Personal Communications AS/400 and 3270
for Windows 95

IBM

Emulator Programming

Version 4.1

SC31-8207-00

Personal Communications AS/400 and 3270
for Windows 95

IBM

Emulator Programming

Version 4.1

© ODBC is copyrighted by Microsoft Corporation.

SC31-8207-00

Note

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

First Edition (May 1996)

This edition applies to Version 4.1 of Personal Communications for AS/400 and 3270. It applies to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

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

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

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

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

© **Copyright International Business Machines Corporation 1995, 1996. All rights reserved.**
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Notices xi
Trademarks xii

About This Book xiii
Who Should Read This Book xiii
How This Book Is Organized xiv
Highlighting Conventions xv
Requirements xvi
  Hardware Requirements xvi
  Software Requirements xvi
  Host Requirements xvii
Compatibility with Other Emulation Products xvii
Related Publications xvii
Notation xvii

Chapter 1. Introduction 1
EHLLAPI/DDE Overview 1
EHLLAPI Interface Description 2
  Function Calls 2
  Dynamic Link Library 3
Getting Started 3

Chapter 2. API Support Files 5
Requirements 5
  Programming Language Support 5
Sample Programs 6
  Compiling and Linking 7
  Using the Header Files in C Language Programs 7
  DLL Interface 7
  Memory Management 8
  Example of Procedure Entry Point 8
  Windows 95 Critical Sections and Signals 8
  Stack Size 9

Chapter 3. Operating the Host System 11
Scenario 1. A Search Function 12
Scenario 2. Sending Keystrokes 13
Scenario 3. Specifying Strings Using EHLLAPI 13
Scenario 4. Distributed Processing 14
Scenario 5. File Transfer 15
Scenario 6. Automation 15
Scenario 7. Keystroke Filtering 16
Scenario 8. Keyboard Enhancement 16

Chapter 4. EHLLAPI Programming Overview 19
HAPI_C32.H 20
Source Code 21
  Initialization 21
  Memory Preallocation 21
  Passing Call Parameter Values 21

© Copyright IBM Corp. 1995, 1996

Call Statement . . . 22
Receiving Return Parameter Values . . . 23
Exiting the Application . . . 24
Compiler . . . 24
Linker . . . 24
Program Execution . . . 25
Debugging . . . 25

Chapter 5. EHLLAPI Programming Reference . . . 27
Summary of Presentation Space Specification . . . 27
Type of Presentation Space . . . 27
Size of Presentation Space . . . 27
Presentation Space Names . . . 27
Host-Connected Presentation Space . . . 28
PSID Handling . . . 28
Sharing EHLLAPI Presentation Space between Processes . . . 30
Summary of Return Codes . . . 33
ASCII Mnemonics . . . 34
General . . . 35
Get Key (51) Function . . . 36
Send Key (3) Function . . . 37

Chapter 6. EHLLAPI Language Syntax and Considerations . . . 39
C and C++ for Windows 95 Guidelines . . . 40
Source Code Preparation . . . 40
Compiling . . . 41
Linking . . . 41

Chapter 7. EHLLAPI Functions . . . 43
Page Layout Conventions . . . 43
Prerequisite Calls . . . 43
Call Parameters . . . 43
Return Parameters . . . 43
Notes on Using This Function . . . 43
Summary of EHLLAPI Functions . . . 44
Allocate Communications Buffer (123) . . . 46
Change Window Name (106) . . . 48
Change Switch List LT Name (105) . . . 50
Connect for Structured Fields (120) . . . 52
Connect Window Services (101) . . . 54
Connect Presentation Space (1) . . . 56
Convert Position or Convert RowCol (99) . . . 58
Copy Field to String (34) . . . 61
Copy OIA (13) . . . 66
Copy Presentation Space (5) . . . 75
Copy Presentation Space to String (8) . . . 80
Copy String to Field (33) . . . 85
Copy String to Presentation Space (15) . . . 88
Disconnect from Structured Fields (121) . . . 91
Disconnect Window Service (102) . . . 93
Disconnect Presentation Space (2) . . . 94
Find Field Length (32) . . . 95
Find Field Position (31) . . . 97
Free Communications Buffer (124) . . . 99

Get Key (51) . . . 100
Get Request Completion (125) . . . 103
LOCK WINDOW SERVICES API (61) . . . 106
LOCK PRESENTATION SPACE API (60) . . . 109
Pause (18) . . . 112
Window Status (104) . . . 114
Post Intercept Status (52) . . . 118
Query Close Intercept (42) . . . 119
Query Communications Buffer Size (122) . . . 120
Query Cursor Location (7) . . . 122
Query Field Attribute (14) . . . 123
Query Host Update (24) . . . 125
Query Window Coordinates (103) . . . 127
Query Session Status (22) . . . 129
Query Sessions (10) . . . 131
Query System (20) . . . 133
Read Structured Fields (126) . . . 135
Receive File (91) . . . 140
Release (12) . . . 142
Reserve (11) . . . 143
Reset System (21) . . . 144
Search Field (30) . . . 145
Search Presentation Space (6) . . . 147
Send File (90) . . . 149
Send Key (3) . . . 151
  Keyboard Mnemonics . . . 153
Set Cursor (40) . . . 159
Set Session Parameters (9) . . . 160
Start Close Intercept (41) . . . 168
Start Host Notification (23) . . . 171
Start Keystroke Intercept (50) . . . 175
Start Playing Macro (110) . . . 178
Stop Close Intercept (43) . . . 179
Stop Host Notification (25) . . . 180
Stop Keystroke Intercept (53) . . . 181
Wait (4) . . . 182
Write Structured Fields (127) . . . 183

Chapter 8. PCSAPI Functions . . . 189
How to Use PCSAPI . . . 189
Page Layout Conventions . . . 189
  Function Type . . . 189
  Parameter Type and Description . . . 189
  Return Code . . . 189
pcsConnect Session . . . 190
pcsDisconnect Session . . . 191
pcsQueryEmulatorStatus . . . 192
pcsQueryWorkstationProfile . . . 193
pcsStartSession . . . 194
pcsStopSession . . . 195

Chapter 9. DDE Functions in a 32-Bit Environment . . . 197
PC/3270 and PC400 DDE Data Items . . . 197
  Using System Topic Data Items . . . 198

Using Session Topic Data Items . . . 199
Using LU Topic Data Items (PC/3270 Only) . . . 199
DDE Functions . . . 200
Naming Conventions for Parameters . . . 201
Find Field . . . 202
Get Keystrokes . . . 204
Get Mouse Input . . . 206
Get Number of Close Requests . . . 209
Get Operator Information Area . . . 210
Get Partial Presentation Space . . . 211
Get Presentation Space . . . 214
Get Session Status . . . 217
Get System Configuration . . . 219
Get System Formats . . . 221
Get System Status . . . 222
Get System SysItems . . . 223
Get System Topics . . . 224
Get Trim Rectangle . . . 225
Initiate Session Conversation . . . 226
Initiate Structured Field Conversation . . . 227
Initiate System Conversation . . . 228
Put Data to Presentation Space . . . 229
Search for String . . . 231
Send Keystrokes . . . 232
Session Execute Macro . . . 234
Set Cursor Position . . . 241
Set Mouse Intercept Condition . . . 242
Set Presentation Space Service Condition . . . 245
Set Session Advise Condition . . . 247
Set Structured Field Service Condition . . . 249
Start Close Intercept . . . 250
Start Keystroke Intercept . . . 252
Start Mouse Input Intercept . . . 253
Start Read SF . . . 257
Start Session Advise . . . 259
Stop Close Intercept . . . 261
Stop Keystroke Intercept . . . 262
Stop Mouse Input Intercept . . . 263
Stop Read SF . . . 264
Stop Session Advise . . . 265
Terminate Session Conversation . . . 266
Terminate Structured Field Conversation . . . 267
Terminate System Conversation . . . 268
Write SF . . . 269
DDE Menu Item API in a 32-Bit Environment . . . 270
DDE Menu Client in a 32-bit Environment . . . 270
DDE Menu Server in a 32-bit Environment . . . 271
DDE Menu Functions in a 32-bit Environment . . . 272
Change Menu Item . . . 273
Create Menu Item . . . 280
Initiate Menu Conversation . . . 282
Start Menu Advise . . . 283
Stop Menu Advise . . . 285
Terminate Menu Conversation . . . 286

Summary of DDE Functions in a 32-bit Environment 287

Chapter 10. Using DDE Functions with a DDE Client Application 293
Using the PC/3270 and PC400 DDE Interface 293
System Conversation 294
Session Conversation 295
Session Conversation (Hot Link) 296
PC/3270 and PC400 DDE Interface 298
DDE Functions for System Conversation 299
Get System Configuration 299
Get System Formats 299
Get System Status 299
Get System SysItems 300
Get System Topics 300
Initiate System Conversation 301
Terminate System Conversation 301
DDE Functions for Session Conversation 302
Find Field 302
Get Operator Information Area 303
Get Partial Presentation Space 304
Get Presentation Space 305
Get Session Status 305
Get Trim Rectangle 306
Initiate Session Conversation 307
Put Data to Presentation Space 307
Search for String 308
Session Execute Macro 308
Set Cursor Position 309
Terminate Session Conversation 309
DDE Functions for Session Conversation (Hot Link) 310
Initiate Session Conversation 310
Start Close Intercept 310
Start Keystroke Intercept 311
Start Session Advise 311
Stop Close Intercept 313
Stop Keystroke Intercept 313
Stop Session Advise 313
Terminate Session Conversation 314
Visual Basic Sample Program 315

Chapter 11. Server-Requester Programming Interface (SRPI) Support . 327
How to Use SRPI 327
SRPI Compatibility 327
Using the Server-Requester Programming Interface 328
SEND_REQUEST Parameters 331
Supplied Parameters 331
Returned Parameters 332
How PC/3270 Applications Use SRPI 332
Invoking SEND_REQUEST 333
Performance Considerations 333
Handling the Interrupt (Ctrl+Break) Key 333
C Requesters 333
C send_request Function 334
SRPI Record Definition 334

SRPI Return Codes 334

Appendix A. Query Reply Data Structures Supported by EHLLAPI . . . 335
The DDM Query Reply 335
DDM Application Name Self-Defining Parameter 336
PCLK Protocol Controls Self-Defining Parameter 336
Base DDM Query Reply Formats 336
The IBM Auxiliary Device Query Reply 338
Direct Access Self-Defining Parameter 340
PCLK Protocol Controls Self-Defining Parameter 340
The OEM Auxiliary Device Query Reply 340
Direct Access Self-Defining Parameter 341
PCLK Protocol Controls Self-Defining Parameter 341
The Cooperative Processing Requester Query Reply 342
The Product-Defined Query Reply 342
Optional Parameters 343
Direct Access Self-Defining Parameter 344
The Document Interchange Architecture Query Reply 344
Direct Access Self-Defining Parameter 345

Appendix B. Compatibility with Personal Communications 4.1 for Windows 3.1 347

Appendix C. Compatibility with Communications Manager/2 349
Set Session Parameter (9) 349
Copy OIA (13) 350
Copy String to PS (15) 350
Storage Manager (17) 351
Copy String to Field (33) 351
Get Key (51) 351
Window Status (104) 351
Query Sessions (10) 351
Connect for Structured Fields (120) 352
Allocate Communications Buffer (123) 352
ASCII Mnemonics 352

Appendix D. SRPI Return Codes 353
Error Handling 353
Transport Layer Errors 353
Application Errors 353
SEND_REQUEST Processing Errors 353
Types of SRPI Return Codes 353
Type 0 Return Code Definitions 354
Type 1 Return Code Definitions 354
Type 2 Return Code Definitions 356
Type 3 Return Code Definitions 356
Class Definitions for Type 2 and Type 3 357
Exception Code Values for Type 2 and Type 3 358
Exception Object Values for Type 2 and Type 3 358
Server Return Codes 359

Appendix E. DDE Functions in a 16-bit Environment 361
PC/3270 and PC400 DDE Data Items in a 16-bit Environment 361
Using System Topic Data Items 362

Using Session Topic Data Items . . . 363
Using LU Topic Data Items (PC/3270 Only) . . . 363
DDE Functions IN A 16-Bit Environment . . . 364
Naming Conventions for Parameters . . . 365
Find Field . . . 366
Get Keystrokes . . . 368
Get Mouse Input . . . 370
Get Number of Close Requests . . . 373
Get Operator Information Area . . . 374
Get Partial Presentation Space . . . 375
Get Presentation Space . . . 378
Get Session Status . . . 381
Get System Configuration . . . 383
Get System Formats . . . 385
Get System Status . . . 386
Get System SysItems . . . 387
Get System Topics . . . 388
Get Trim Rectangle . . . 389
Initiate Session Conversation . . . 390
Initiate Structured Field Conversation . . . 391
Initiate System Conversation . . . 392
Put Data to Presentation Space . . . 393
Search for String . . . 394
Send Keystrokes . . . 395
Session Execute Macro . . . 396
Set Cursor Position . . . 403
Set Mouse Intercept Condition . . . 404
Set Presentation Space Service Condition . . . 407
Set Session Advise Condition . . . 409
Set Structured Field Service Condition . . . 411
Start Close Intercept . . . 412
Start Keystroke Intercept . . . 414
Start Mouse Input Intercept . . . 415
Start Read SF . . . 419
Start Session Advise . . . 421
Stop Close Intercept . . . 423
Stop Keystroke Intercept . . . 424
Stop Mouse Input Intercept . . . 425
Stop Read SF . . . 426
Stop Session Advise . . . 427
Terminate Session Conversation . . . 428
Terminate Structured Field Conversation . . . 429
Terminate System Conversation . . . 430
Write SF . . . 431
Summary of DDE Functions in a 16-Bit Envionment . . . 432
DDE Menu Item API in a 16-bitEnvironment . . . 436
DDE Menu Client in a 16-bit Environment . . . 436
DDE Menu Server, 32-bit . . . 437
DDE Menu Functions in a 16-bit Environment . . . 438
Change Menu Item . . . 439
Create Menu Item . . . 445
Initiate Menu Conversation . . . 447
Start Menu Advise . . . 448
Stop Menu Advise . . . 450

Terminate Menu Conversation 451

Index 453

Notices

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
USA

For license inquiries regarding double-byte character set (DBCS) information, contact:

IBM World Trade Asia Corporation Commercial Relations
IBM Kamiya-cho Building, 4-3-9 Toranomon, Minato-ku
Tokyo, Japan

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

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

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

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

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

© Copyright IBM Corp. 1995, 1996

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

APPN
AS/400
CICS
IBM
OS/2
OS/400
Personal System/2
Presentation Manager
PS/2
RT PC
S/390
TopView
VTAM
XGA
400

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

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

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

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

About This Book

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

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

Note: SRPI is supported by PC/3270 only.

If you want only to get started using PC/3270 or PC400 or have no interest in using the programming interface, see the *Personal Communications AS/400 and 3270 for Windows 95 Version 4.1 Up and Running*.

Who Should Read This Book

This book is intended for workstation users and programmers who write application programs that use EHLLAPI/DDE/PCSAPI/SRPI functions.

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

The programmer must also be familiar with connecting to a host system from a terminal or from a workstation with terminal emulation software.

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

© Copyright IBM Corp. 1995, 1996

How This Book Is Organized

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

Chapter 2, API Support Files, contains information about migrating from previous versions of Personal Communications to Personal Communications Version 4.1 for Windows 95.

Chapter 3, Operating the Host System, contains information about automated operation of the host system using EHLLAPI/DDE functions in applications.

Chapter 4, EHLLAPI Programming Overview, provides general information concerning development of applications that use EHLLAPI functions.

Chapter 5, EHLLAPI Programming Reference, contains reference information when programming with EHLLAPI.

Chapter 6, EHLLAPI Language Syntax and Considerations, provides information concerning syntax and general considerations for using each supported high-level language.

Chapter 7, EHLLAPI Functions, describes each PC/3270 and PC400 EHLLAPI function in detail. The functions are arranged alphabetically by name. See page 287 when you want to find the EHLLAPI function by the function number.

Chapter 8, PCSAPI Functions, describes PCSAPI function supported in the PC/3270 and PC400.

Chapter 9, DDE Functions in a 32-Bit Environment, provides information concerning Dynamic Data Exchange functions of PC/3270 and PC400.

Chapter 10, Using DDE Functions with a DDE Client Application, describes how to use DDE functions with other Windows** application programs (Microsoft** Visual Basic**, Microsoft Excel**, Microsoft Word**) that have DDE client functions. includes information about PC/3270 for Windows 95 support for SRPI functions.

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

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

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

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

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

Highlighting Conventions

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

Type style	Emphasized Information
ROMAN CAPS	ROMAN CAPS are used to emphasize commands, keywords, or other information that the programmer is to type in literally. This type style is not used in lengthy sample program listings.
Bold	A **Bold** type style (Initial Caps) is used to emphasize EHLLAPI function names. Keep in mind that these names are labels used to refer to the function. They are not used *in* an application to *call* an EHLLAPI function; the *function number* parameter is used to specify which EHLLAPI function is to be performed. See Chapter 7, “EHLLAPI Functions.”
Italics	When *italics* or ***bold italics*** appear in text, either a term is being used for the first time or the meaning implied by the word is being stressed.
Bold Monospace	Bold monospace type indicates file and directory names. And it is also used for programmable keywords.
Italic Monospace	Italic monospace type indicates variables for placeholders, parameters, and file names. When you enter a command, replace all variables with the appropriate file name, number, or option. On an example display screen, nonitalic regular monospace type indicates a variable quantity other than a default value.
Monospace	Monospace type is used in the following situations: In sample code listings of more than a few lines. As options for link or compile commands. For names of session options set by the **Set Session Parameters** (9) function. See “Set Session Parameters (9)” on page 160. For ASCII mnemonics in text. For displayed information. For input from either a programmer or an operator.

Requirements

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

Hardware Requirements

PC/3270 and PC400 EHLLAPI/DDE/PCSAPI/SRPI require the storage for Windows 95 and at least 3 MB of additional storage, and are designed to operate on workstations with 80386** or higher processor.

Software Requirements

EHLLAPI/DDE/PCSAPI/SRPI has the following software requirements and limitations:

PC/3270 or PC400 for Windows 95 is supported.

Write your PC/3270 or PC400 EHLLAPI/DDE/PCSAPI/SRPI program in one of the following high-level languages:

- Microsoft Visual C++ Version 4.0

For PC400, to use the file transfer APIs in this book, install one of the following programs on the host system:

- Personal Communication Tools:
  85G9969 for quarter-inch tape
  85G9973 for half-inch tape

Note: For additional information on available hardware and software that can be used with this product, refer to the *Personal Communications for AS/400 and 3270, Version 4.1, Up and Running for Windows 95* and *Personal Communications for AS/400 and 3270, Version 4.1, Reference for Windows 95* for your Personal Communications program.

Host Requirements

EHLLAPI/DDE/PCSAPI/SRPI has the same host requirements as PC/3270 or PC400. Refer to the *Personal Communications for AS/400 and 3270, Version 4.1, Up and Running for Windows 95* and *Personal Communications for AS/400 and 3270, Version 4.1, Reference for Windows 95* for your Personal Communications program.

Compatibility with Other Emulation Products

For PC/3270: Most EHLLAPI applications that are written for the following IBM High-Level Language Application Program Interface (HLLAPI) products can be used with PC/3270 EHLLAPI:

IBM Workstation Program HLLAPI
IBM Entry Emulator HLLAPI
IBM Communications Manager/2 EHLLAPI

For PC400: Most EHLLAPI applications written for the IBM Communications Manager/2 EHLLAPI can be used with PC400 EHLLAPI.

For compatibility with other Emulator programs, refer to Appendix B, “Compatibility with Personal Communications 4.1 for Windows 3.1.”

EHLLAPI does not support DOS.

Related Publications

The following is a list of related publications:

Personal Communications Version 4.1 Up and Running for Windows 95, SC31-8205

Personal Communications Version 4.1 Reference for Windows 95, SC31-8206

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

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

Notation

A table at the beginning of each section explains API or DDE functions in Chapter 7, “EHLLAPI Functions,” Chapter 8, “PCSAPI Functions,” Chapter 9, “DDE Functions in a 32-Bit Environment,” and Appendix E, “DDE Functions in a 16-bit Environment.” It shows the products that provide the function described in the section. ‘X’ means it is supported, and ‘–’ means not supported. For example, the following table indicates that a function is available in the PC/3270 it is not supported in the PC400.

PC/3270	*PC400*
X	–

Chapter 1. Introduction

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

EHLLAPI/DDE Overview

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

You can write workstation programs using PC/3270 or PC400 EHLLAPI/DDE in the supported languages listed in “Software Requirements” on page xvi.

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

- Automate repetitive tasks.
- Mask complex applications from the user.
- Consolidate several complicated tasks into one simple task.
- Simplify existing host applications.
- Provide for unattended operation through a ***programmed operator***, which monitors tasks without human intervention. This programmed operator can:
  - Monitor events that are serial in nature—for example, data center applications.
  - Automate console operation.
  - Monitor response time and availability.
  - Do stress testing.
- Create ***composite screen*** applications using input from the host. The input for a composition screen is formatted and presented within a workstation presentation space.
- Write programs that divide the work between host and workstation sessions. EHLLAPI/DDE lets you write and use an application to access files and programs in either the host or the workstation session. The more trivial tasks, such as file editing, can be done in the workstation session, and the more demanding tasks (database searches, updates, and retrievals) can be done in the host session when needed. The terminal operator does not even need to know whether a workstation or host application is doing the work. The goal is simply to finish the task.

In other words, you can use EHLLAPI/DDE with the functions provided by PC/3270 or PC400 to enhance interaction between a workstation application program and a host session, or also to simplify your programming tasks.

© Copyright IBM Corp. 1995, 1996

EHLLAPI Interface Description

The interface between an EHLLAPI application and the host can be described as follows:

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

PC/3270 or PC400 software includes dynamic link libraries (DLLs), which provide application interface programs to communicate with the terminal emulation software.

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

- The terminal emulation software passes the data on to the host. To the host, this data looks just like it came from a terminal.
- The terminal emulation software receives the host response and passes the result back to the DLL.
- The DLL then converts the data into the required parameters and passes them back to the application.

This entire process is not apparent to the user. To make all this work, the programmer must write, compile, and link an application that contains valid function calls. A valid function call in the correct syntax provides valid data for the required parameters and uses the appropriate entry-point name for the DLL. The entry-point name for function calls is **HLLAPI**.

Function Calls

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

A CALL verb
A DLL entry-point name
Four parameters

The CALL verb is not required in some languages. The four parameters must all be passed by reference, even though some functions do not use all four parameters. If a parameter is not used by a function, the value of that variable name is ignored.

The four parameters that are passed are:

1. function number,
2. data string,
3. length, and
4. return code parameters.

Notes:

1. On call and return, the function number, length, and return code parameters are 2-byte integers in all supported languages.
2. On call and return, the data string parameter contains string (nonnumeric) data. Numbers contained in the data string are handled as string data, not numeric data.

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

As a minimum, all functions require a function number on call, and all functions return a function number and a return code parameter. The function number is always returned unchanged. On return (for all functions except the **Convert Position or RowCol** (99) function), the return code value indicates the status of the function or of the host. Information concerning return parameters and required call parameters for each function is contained in Chapter 7, "EHLLAPI Functions."

The return code parameter indicates whether the function executed successfully and, in some cases, indicates the reason for the failure. For example, if a function number that is not valid is specified, a return code of 10 is returned. Refer to "Summary of Return Codes" on page 33 for more information.

The data string and length parameters are discussed for each function in Chapter 7, "EHLLAPI Functions."

More specific information concerning syntax and content of function calls is provided in "Source Code" on page 21.

Dynamic Link Library

A dynamic link library (DLL) provides an interface between your application and the PC/3270 or PC400 EHLLAPI/DDE software. The DLL is necessary because each high-level language has a unique method of storing data and calling subroutines.

DLLs are provided as part of PC/3270 or PC400 software. Applications are required to call a DLL to use EHLLAPI functions.

PC/3270 and PC400 for Windows 95 provide **PCSHLL.DLL** to support the existing EHLLAPI applications for PC/3270 and PC400 for Windows 3.1 **PCSHLL.DLL** converts 16-bit addressing to 32-bit addressing and passes it to PC/3270 or PC400 EHLLAPI DLL. But there are several restrictions when using the existing EHLLAPI applications for PC/3270 and PC400 for Windows 3.1. For these restrictions, refer to Appendix B, "Compatibility with Personal Communications 4.1 for Windows 3.1."

Getting Started

Before using PC/3270 or PC400 EHLLAPI, customize your workstation for use with EHLLAPI. Refer to the *Personal Communications, Version 4.1, Up and Running for Windows 95* or the *Personal Communications, Version 4.1, Reference for Windows 95*.

For more information about running an application, see "Program Execution" on page 25.

Chapter 2. API Support Files

Requirements

The requirements for a 32-bit API interface are:

4 byte-boundary alignment

Current existing 16-bit API application support for Intel-based machines

Toolkit for 32-bit API interface including header files and LIB file

Sample programs for 32-bit API interface

The extendedname fields of all EHLLAPI structures should be set to binary zero before making EHLLAPI calls to ensure compatibility with future products.

If the EHLLAPI entry point is dynamically loaded using **LoadLibrary** and **GetProcAddress**, the entry point ordinals should be used as defined in the header file, not the entry point names.

Applications should use the header file structures to access buffers sent to or received from EHLLAPI. Applications should not depend on specific hard-coded offsets into EHLLAPI data buffers.

Programming Language Support

Sample programs in the C programming language and related instructions support one or more of the following C compilers:

| Microsoft Visual C++ 4.0

Note: Other compilers *might* work with these header files but could require minor modifications.

The C sample files contain code that allow these programs to be built with a supported 32-bit compiler. These samples are informational and are primarily designed to demonstrate techniques for migrating applications from 16-bit to 32-bit operation.

E IBM ***strongly recommends*** that 32-bit application programs be developed for
E Personal Communications Version 4.1 for Windows 95 if new programs are
E developed.

© Copyright IBM Corp. 1995, 1996

Sample Programs

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

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

The sample programs provided with Personal Communications V4.1 are located on both diskette and CD-ROM media. The sample program files include source and supporting files for the following Personal Communications Version 4.1 for Windows 95 APIs:

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

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

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

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

Filename	Description
DDE_C.H	DDE include file
HAPI_C.H	EHLLAPI include file
PCSAPI.H	PCSAPI include file
PCSCALLS.LIB	Import library for 16-bit interface
PCSCAL32.LIB	Import library for 32-bit interface
UUCCPRB.H	SRPI include file

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

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

Compiling and Linking

Building executable programs (.EXE files) from the sample source code requires the following procedure to use the Personal Communications for Windows 95 API data structures and link libraries:

1. Start Microsoft Visual C++ 4.0.
2. Open makefile (.makfile) of the sample source code.
3. Build executable program (Exe.file).

Using the Header Files in C Language Programs

Each C header file that contains verb structures (For example: HAPI_C.H, UUCCPRB.H, and PCSAPI.H) will include the constant file if it has not yet been included.

If you include windows.h in your program, you should place the include statement for windows.h before any API include statements.

Note: Some Personal Communications Version 4.1 for Windows 95 include files require Windows.H to be previously included.

DLL Interface

Windows 95 provides two ways to dynamically link to a DLL:

Load-time linking
Run-time linking

The PCSCAL32.LIB library contains the 32-bit entry point linkage information for all Personal Communications Version 4.1 for Windows 95 API entry points.

Notes:

1. Run-time linking provides conditional linking to a DLL. In order to find the function address of the API entry point, your application can issue **LoadLibrary** for the module name of PCSHLL32.DLL and then **GetProcAddress** using the ordinal number of the routine. The ordinal number can be found in the HAPI_C16.H or HAPI_C32.H files supplied with Personal Communications Version 4.1 for Windows 95.

Memory Management

Application programs pass two types of memory blocks to HLLAPI; parameter blocks and communication buffers. Parameter blocks for blocking verbs can be located in the application program's private memory. All data buffers must be located in shared memory.

Private Memory

The parameter blocks can be declared as a program variable. Alternatively, your program can allocate or suballocate the verb control block in dynamically allocated read/write memory.

Shared Memory

Because some application processing occurs in a separate Personal Communications Version 4.1 for Windows 95 process, applications require communications buffers to be in shared memory allocated with **Allocate Communications Buffer (123)**.

Example of Procedure Entry Point

The entry point for HLLAPI, in any supported language, is:

Entry Point	Description
HLLAPI	This entry point in PCSHLL32.DLL is for control, conversation, management, and configuration verbs.

The DLL is initially installed in:

PCSHLL32.DLL *d*:\Personal Communications

where *d* is the drive where Personal Communications Version 4.1 for Windows 95 is installed.

Example Call

To call an HLLAPI function in a supported 32-bit C language prgram, the following program statement is needed:

```
extern long WINAPI hllapi(LPINT, LPSTR, LPINT, LPINT);
```

Note: The header file, HAPI_C.H, contains the HLLAPI function prototype.

Windows 95 Critical Sections and Signals

Use **EnterCriticalSection** carefully when your program issues verbs. Do not issue verbs within a critical section. That is, do not issue verbs within a critical section after your program issues **EnterCriticalSection** unless **LeaveCriticalSection** first completes successfully. Similarly, if one thread of a program establishes a critical section and another thread is within a call to a verb, the verb is suspended until the critical section is exited.

During processing of a verb, all signals (except numeric coprocessor signals) are delayed until verb completion or until the verb needs to wait for incoming data. For example, pressing Ctrl+Break ends a transaction program if an application is waiting for data to arrive while it processes a verb. Also, **TerminateProcess** issued from another process is held until the application completes a verb it might be processing or has to wait for incoming data.

Stack Size

APIs use the calling application's stack when it executes a verb. Windows 95, the API, and your program's variables need additional stack space. The recommended stack size is at least 8196 bytes. An example of a STACKSIZE statement for a .DEF file follows:

```
STACKSIZE 8196
```

Specifying STACKSIZE in the .DEF file overrides stack sizes specified in the source code.

Chapter 3. Operating the Host System

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

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

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

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

The sample scenarios presented here 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
  - Specifying strings using EHLLAPI
- Distributed processing, including:
  - Data extraction
  - File transfer
- Integrating interfaces

© Copyright IBM Corp. 1995, 1996

Scenario 1. A Search Function

There are four phases in a typical host system transaction:

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

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

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

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

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

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

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

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

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

No Response From Host. Retry?

In this environment, program revisions become very important considerations, because the “programmed operator” must be reprogrammed for even minor changes in the display messages.

For example, if a terminal operator expects the message:

Enter Part Number:

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

Enter Component Number:

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

Scenario 2. Sending Keystrokes

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

The Tab key mnemonic (@T; see page 35 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. Specifying Strings Using EHLLAPI

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

The calling data string can contain:

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

The return data string can contain:

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

EHLLAPI requires you to indicate the length of all strings.

The length of calling data strings can be specified by either of the two methods described below, except for the **Set Session Parameters** (9) function.

For returned data strings, memory must be preallocated. The method used to do this depends on the high-level language in which your application is written. See “Memory Preallocation” on page 21 for more information.

To specify the length of calling data strings, you can:

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

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

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

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

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

To send the following string to a session:

XREF 12345

followed by the Enter key, you can pass:

XREF 12345

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

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

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

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

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

Scenario 4. 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 5. 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 6. Automation

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

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

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

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

Scenario 7. 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.

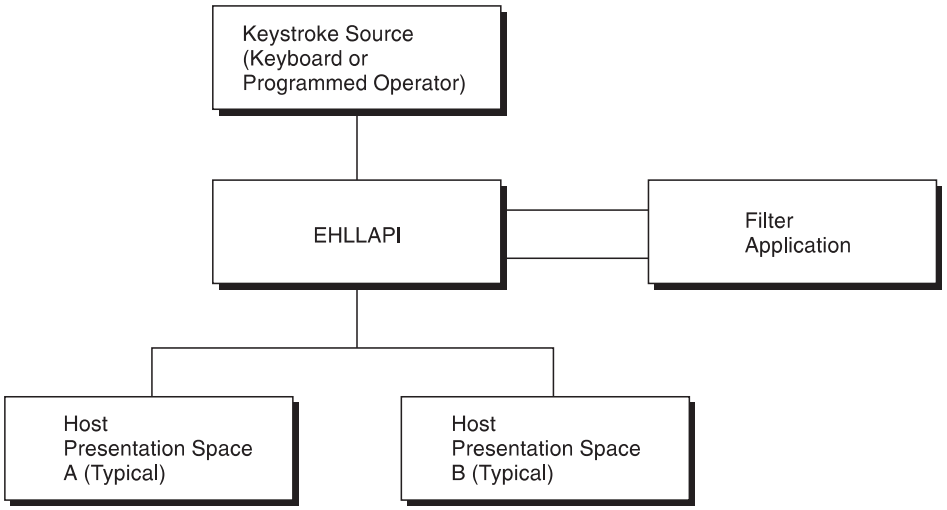

Figure 1. Keystroke Flow

Scenario 8. Keyboard Enhancement

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

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

1. Your EHLLAPI application program calls the **Connect Presentation Space** (1) function to connect to the presentation space whose keystrokes are to be filtered.
2. Your EHLLAPI program next calls the **Start Keystroke Intercept** (50) function specifying the L option. This causes all keystrokes to be routed to the filtering application program.
3. The filtering application program can now define a loop in which:

a. The **Get Key** (51) function intercepts all keystrokes being sent to the target presentation space.

b. The filtering application examines each keystroke and performs a keyboard macro task, such as:

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

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

   Creating **boiler plates** for contracts or frequently used letters

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

   For example, the filtering application might convert a key combination such as **Alt+Y** into a command to move the cursor to column 35 of the second line in presentation space and write the string “XYZ Tool Corporation, Dallas, Texas”.

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

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

Chapter 4. EHLLAPI Programming Overview

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

After the functional requirements for a program have been defined and it is determined that host support is required or desirable, the programmer should identify those tasks that will be performed by the host. A set of calls to EHLLAPI functions necessary to perform the required tasks can then be incorporated in the application. Refer to **Chapter 7, "EHLLAPI Functions."** , for information about each EHLLAPI function.

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

Programs using EHLLAPI functions are usually typed in as source code in a supported high-level language, compiled, and linked. The executable file can be run on the workstation to test and debug the individual functions. The following sections provide information about each of these steps. See Figure 2 below for the Personal Communications Version 4.1 for Windows 95 EHLLAPI component structure.

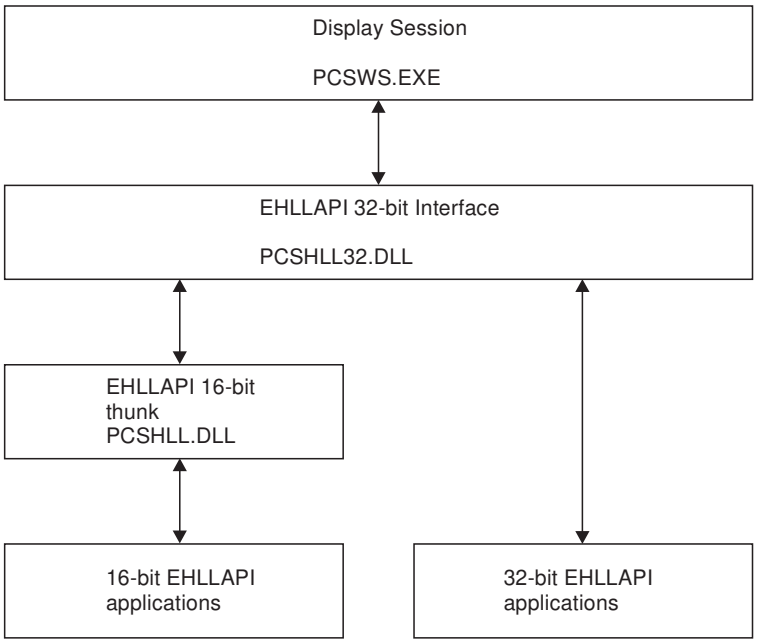

Figure 2. Personal Communications Version 4.1 for Windows 95 EHLLAPI component structure

© Copyright IBM Corp. 1995, 1996

HAPI_C32.H

The following points need to be considered when the 32-bit EHLLAPI data structure is designed:

Each data field must be aligned to a 4-byte boundary. Therefore, data length is always a multiple of 4.

Numeric data should be stored in a 2-byte or 4-byte field, instead of 1-byte field, to avoid incompatibility between machine architectures.

Bit (flag) should be stored in 2-byte or 4-byte field, instead of 1-byte field to avoid incompatibility between machine architectures.

Source Code

This section describes source code statements that are required to use EHLLAPI functions for applications.

Initialization

Generally, the variable names used in PC/3270 and PC400 EHLLAPI function call statements must be declared before the call is executed.

For the applications, you should call the **Reset System** (21) function at the beginning of your application to create a known set of conditions. Also, check the return code; a return code of 1 indicates that EHLLAPI is not loaded. You can include an operator prompt to indicate the remedy. The **Query System** (20) function also generates a return code of 1 if EHLLAPI is not loaded.

Specific initialization statements for each supported high-level language are described beginning on page 39.

Memory Preallocation

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

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

In C or C++ for Windows 95, include a statement such as:

```
char q_buffer[36];              / for query system /
```

to preallocate the required memory.

Passing Call Parameter Values

PC/3270 and PC400 EHLLAPI function calls pass parameter values by reference. Literal values for parameters cannot be entered in the calling statement. Before calling an EHLLAPI function, store the value for each parameter required by the function in the corresponding variable name. Values need not be set for call parameters listed as not applicable (NA) in **Chapter 7, "EHLLAPI Functions."**

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

1. Function Number
2. Data String
3. Length (or a buffer size)
4. Presentation Space Position

Function Number, the first parameter, specifies the EHLLAPI function to be executed and is always required. For the applications, this position must be filled by a 4-byte pointer that points to a 4-byte integer value.

Data String, the second parameter, is used in different ways by different functions. In some functions, the data string is a string of characters. In other functions, it is a string of concatenated data items. When not applicable (NA), this parameter can be any string.

Length, the third parameter, is usually the length (in number of bytes) of the calling data string described in the preceding paragraph. In special cases, this parameter passes information such as buffer size. When used, this position must be filled by a 4-byte pointer that points to a 4-byte integer value. When length is either not applicable (NA) or overridden by the current value of a session parameter defined by the **Set Session Parameters** (9) function, this integer can be any value.

Presentation Space (PS) Position, the fourth parameter, is a value associated with the IBM 3278/79 or 5250 Display Station screen sizes emulated by the workstation. When used, this position must be filled by a 4-byte pointer that points to a 4-byte integer value from 1 to 3564. The valid upper limit for the value of this parameter can be less than 3564 depending on how you configure the host presentation space. See “Summary of Presentation Space Specification” on page 27.

When PS Position is either not applicable (NA) or overridden by the current value of a session parameter defined by the **Set Session Parameters** (9) function, this 4-byte integer can be any value.

Call Statement

After any required initialization statements are executed, calls to EHLLAPI functions can be located anywhere you need them within the application code. The pattern for an EHLLAPI function call is:

hllapi(*FUNC,DATA_STRING,PARM_3,PARM_4***)**

Term	Description
hllapi	hllapi is an entry-point name of EHLLAPI's dynamic link library (**PCSHLL32.DLL**).
FUNC	As the first parameter, *FUNC* is always the EHLLAPI function number on call and on return. The function number is expressed as a 4-byte pointer that points to a 4-byte integer function number.
DATA_STRING	As the second parameter, *DATA_STRING* contains presentation space identification, function options, or data on call or return. *DATA_STRING* is expressed as a 4-byte pointer that points to a string.
PARM_3	As the third parameter, when used on call or return, *PARM_3* contains a data string length, the length of a field in the host presentation space, or a host presentation space position. In **Chapter 7, “EHLLAPI Functions,”** the heading *Length* is used for information concerning use of this parameter during call and return. *PARM_3* is expressed as a 4-byte pointer that points to a 4-byte integer value.

PARM_4 As the fourth parameter, *PARM_4* is a 4-byte pointer that points to a 4-byte integer value on call or return. When used on call, *PARM_4* is usually the host presentation space position. It is always used on return and usually contains the return code. In **Chapter 7, "EHLLAPI Functions,"** the heading *PS Position* is used for information concerning use of this parameter during call. The heading *Return Code* is used for information concerning use of this parameter during return.

The names used for the parameters (*FUNC*, *DATA_STRING*, *PARM_3*, and *PARM_4*) in the preceding example are typical. The parameters can have any name, but the name used must be of the proper type.

The names used to pass the first, third, and fourth parameters must be 4-byte pointer variables that point to a 4-byte integer value.

The name used to pass the second parameter must be a string variable.

A unique name must be used for each parameter.

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

Receiving Return Parameter Values

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

PC/3270 and PC400 EHLLAPI return codes support a common error handling routine. See "Summary of Return Codes" on page 33 for more information.

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

1. Function Number
2. Data String
3. Length
4. Return Code

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

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

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

Preallocate space for returning data strings in your EHLLAPI application program. See “Memory Preallocation” on page 21.

Length, the third parameter, is usually the length (in number of bytes) of the data string described in the immediately preceding paragraph. For certain functions, this parameter returns a presentation space position or an attribute byte, a field length, or a field position.

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

The return code parameter for the **Convert Position or RowCol** (99) function is a special case: It returns data to the application program. If your EHLLAPI application calls the **Convert Position or RowCol** (99) function, your application must provide for the processing of this data to avoid misleading or unpredictable results. Refer to “Convert Position or Convert RowCol (99)” on page 58 for more information.

Detailed explanations of return codes are contained in the descriptions of the individual EHLLAPI functions. A summary of return codes is provided under “Summary of Return Codes” on page 33.

Exiting the Application

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

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

Compiler

A compiler from the list under “Software Requirements” on page xvi should be used.

Linker

Use the Windows 95 linker provided with each of the supported high-level languages.

Note: When linking the EHLLAPI program, use **PCSCAL32.LIB** to resolve the address of the entry-point name of EHLLAPI.

Program Execution

After you have successfully compiled and linked your application, you are ready to run it.

Do as follows:

1. At the Windows 95 command prompt, type:

   d:*path**yourprog*

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

2. Press **Enter**.

Debugging

You can use the Trace Facility of Personal Communications to trace your Windows 95 EHLLAPI programs.

For more information about the Trace Capture program, see *Personal Communications for Windows 95, User's Guide*.

Chapter 5. EHLLAPI Programming Reference

This chapter contains the following information needed to write a program:

Summary of the presentation space specification
Summary of prerequisite calls
Summary of return codes
ASCII mnemonics

Summary of Presentation Space Specification

Some functions interact with host presentation space. The supported type, size, and names of presentation space are discussed in the following paragraphs.

Type of Presentation Space

A session can be declared as a display session or as a printer session during customization. EHLLAPI applications cannot connect to printer sessions, and router sessions of PC400.

PC/3270 and PC400 EHLLAPI support only the host system type of presentation space.

Size of Presentation Space

PC/3270 and PC400 EHLLAPI support:

A minimum presentation space size of 1920 bytes. This is the number of bytes in a 24-row by 80-column screen.

A maximum presentation space size of 9920 bytes. This is the number of bytes in a 62-row by 160-column screen.

You can specify a presentation space size between these two sizes. To specify the number of bytes in a presentation space, use the Customization facility of PC/3270 or PC400.

To find the size of a presentation space, examine bytes 13 and 17 of the returned data string for the **Query Session Status** (22) function. See “Query Session Status (22)” on page 129.

Presentation Space Names

PC/3270 or PC400 EHLLAPI supports up to 26 presentation spaces or sessions. Valid names are the letters *A* through *Z*. To be valid within an application, a name must be declared during customization. See the *Personal Communications/3270 Version 4.1 for Windows 95 Reference* or the *Personal Communications AS/400 Version 4.1 for Windows User's Guide*.

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

© Copyright IBM Corp. 1995, 1996

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

Host-Connected Presentation Space

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

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

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

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

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

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

You cannot connect to a logical printer session.

PSID Handling

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

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

      A letter *A* through *Z*
      A blank or a null
2. The status of the connection to the host presentation space.

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

PSID Handling for Functions Requiring Connect

Some functions interact only with the host-connected presentation space. These functions require the **Connect Presentation Space** (1) function as a prerequisite call. The PSID for these functions is determined by the **Connect Presentation Space** (1) and the **Disconnect Presentation Space** (2) functions as follows:

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

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

PSID Handling for Functions Not Requiring Connect

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

Connect Presentation Space (1)
Convert Position RowCol (99)
Get Key (51)
Post Intercept Status (52)
Query Close Intercept (42)
Query Host Update (24)
Query Session Status (22)
Query Sessions (10)
Query System (20)
Start Close Intercept (41)
Start Host Notification (23)
Start Keystroke Intercept (50)
Stop Close Intercept (43)
Stop Host Notification (25)
Stop Keystroke Intercept (53)

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

A letter *A* through *Z*
A blank or a null

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

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

1. The function can interact with any declared host presentation space if a letter, not a blank or a null, is used to specify the PSID.
2. If a blank or a null is used to specify the PSID, a return code of 1 is generated. The function does not execute.
3. Using a letter to specify the PSID does not establish a host-connected presentation space. Unless a connect is executed:

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

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

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

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

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

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

Sharing EHLLAPI Presentation Space between Processes

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

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

The type of shared access can be defined by setting the following read and write sharing options for each function in the SET SESSION PARAMETERS (9) function call:

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

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

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

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

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

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

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

Table 1. EHLLAPI Read and Write Sharing Option Combinations

Calling Application	Super_Write	Write_Super	Write_Write	Write_Read	Write_None	Read_Write
Super_Write	Yes	Yes	Yes	No	No	Yes
Write_Super (default)	Yes	No	No	No	No	No
Write_Write	Yes	No	Yes	No	No	Yes
Write_Read	No	No	No	No	No	Yes
Write_None	No	No	No	No	No	No
Read_Write	Yes	No	Yes	Yes	No	Yes

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

Notes:

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

Table 2. Prerequisite Functions and Associated Dependent Functions

Prerequisite Call	Functions	Access
Allocate Communications Buffer (120)	Free Communication Buffer (120)	
Connect Window Service (101)	Disconnect Window Service (102) Query Window Service (103) Window Status (104) Change Switch List Name (105) Change PS Window Name (106)	Read Query=Read Set=Write Write Write
Connect Presentation Space (1)	Disconnect Presentation Space (2) Send key (3) Wait (4) Copy Presentation Space (5) Search Presentation Space (6) Query Cursor Location (7) Copy Presentation Space to String (8) Receive File (91) Reserve (11) Release (12) Copy OIA (13) Query Field Attribute (14) Copy String to Presentation Space (15) Search Field (30) Send FIle (90) Find Field Position (31) Find Field Length (32) Copy String to Field (33) Copy Field to String (34) Set Cursor (40)	Write Read Read Read Read Read Read Read Write Write Read Read Write Read Write Read Read Write Read Write
Connect Structured Field (120)	Disconnect Structured Field (121) Get Request Completion (125) Read Structured Field (126) Write Structured Field (127)	
Read Structured Field (126)	Get Request Completion (125)	
Start Close Intercept (41)	Stop Close Intercept (43) Query Close Intercept (42)	
Start Host Notification (23)	Query Host Update (24) Stop Host Notification (25)	
Start Keystroke Intercept (50)	Get Key (51) Post Intercept Status (52) Stop Keystroke Intercept (53)	
Write Structured Field (127)	Get Request Completion (125)	

Locking Presentation Space

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

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

Summary of Return Codes

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

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

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

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

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

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. Whenever you receive a return code 9, you should issue a call to the **Query System** (20) function and exit. Extended error codes 1 and 2 that are returned by this function are vital in helping your service coordinator determine the cause of the system error. When you contact your service coordinator, be prepared to provide the system error information generated by the **Query System** (20) function.
10	This function is not available for EHLLAPI.
11	This resource is not available.
12	This session stopped.
24	The string was not found, or the presentation space is unformatted.
25	Keystrokes were not available on input queue.
26	A host event occurred. See **QUERY HOST UPDATE (24)** for details.
27	File transfer was ended by a Ctrl+Break command.

Return Code	Explanation
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).

ASCII Mnemonics

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

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

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

Mnemonics for unshifted keys consist of the escape character followed by an abbreviation. This is also true for the shift keys themselves, Upper shift, Alt, and Ctrl. Mnemonics for shifted keys consist of the mnemonic for the shift key followed by the mnemonic for the unshifted key. Hence the mnemonic for a shifted key is a 4-character sequence of escape character, abbreviation, escape character, abbreviation.

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

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

Upper shift @S

Alt	@A
Ctrl	@r

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

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

The following text describes the use of these functions.

General

All defined keys are represented by either:

A 1-byte ASCII value that is part of the 256-element ASCII character set, or
A 2-, 4-, or 6-byte ASCII mnemonic

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

To represent a key defined as a function, a 2-, 4-, or 6-byte ASCII mnemonic that corresponds to that function is used. For example, to represent the backtab key, @B is used. To represent PF1, @1 is used. To represent Erase Input, @A@F is used. See the following tables:

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

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

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

Notes:

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

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

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

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

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

Get Key (51) Function

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

If the operator types a key defined as a function, the host application receives a 2-, 4-, or 6-byte ASCII mnemonic that corresponds to that function. For example, if the **Backtab** key is typed, @B is received. If **PF1** is pressed, @1 is received. If **Erase Input** is pressed, @A@F is received.

If the operator types a defined shift key combination, the host application receives the ASCII character, or the 2-, 4-, or 6-byte ASCII mnemonic that corresponds to the defined character or function.

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

The **Get Key** (51) function precedes all characters and mnemonics sent to the host application as described in the preceding text with two ASCII characters. The first ASCII character is the PSID of the host presentation space to which the keystrokes

are sent. The fourth character is an *A* or *M* for ASCII, special shift, or mnemonic, respectively. See page 101.

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.

Chapter 6. EHLLAPI Language Syntax and Considerations

This chapter provides source code syntax, compiler commands and options, and linker commands and options to call EHLLAPI functions in C and C++ for Windows 95.

© Copyright IBM Corp. 1995, 1996

C and C++ for Windows 95 Guidelines

The following guidelines are helpful in preparing EHLLAPI applications that call EHLLAPI functions in C and C++ for Windows 95

Source Code Preparation

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

Required Statements

To call EHLLAPI from C or C++ for Windows 95, the following variables must be declared:

32-Bit Interface

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

USHORT apilen    = 4 ;
USHORT apiret ;
```

This function declaration and the data structures are defined in **HAPI_C.H**.

EHLLAPI Function Call Syntax

The syntax for calls to EHLLAPI functions in C or C++ is shown below. This example calls the **Connect Presentation Space** (1) function.

32-Bit Interface

```
USHORT apifunc = 1 ;
CHAR     apistring[4]  ;
USHORT apilen    = 4 ;
USHORT apiret ;
apistring[ ] = 'A' ;
apistring[1] = apistring[2] = apistring[3] =  ;

hllapi (&apifunc,
          apistring,
          &apilen,
          &apiret)  ;
```

Compiling

Compile your EHLLAPI program (for the compile options, refer to the manual of the compiler):

Linking

Link the resultant .OBJ files with the appropriate object file or libraries.

E You must also link it with the EHLLAPI import library, PSCAL32.LIB for 32-bit and
E PSCALLS.LIB for 16-bit applications.

Chapter 7. EHLLAPI Functions

This chapter describes each individual PC/3270 and PC400 EHLLAPI function in detail and explains how to use the EHLLAPI program sampler. The functions are arranged alphabetically by name. The functions are explained for both the 16-bit and 32-bit environments.

Page Layout Conventions

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

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

Prerequisite Calls

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

Call Parameters

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

Return Parameters

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

Notes on Using This Function

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

© Copyright IBM Corp. 1995, 1996

Summary of EHLLAPI Functions

The following table is the summary of the EHLLAPI functions:

Number	Function	PC/3270	PC400	Page
1	Connect Presentation Space (1)	X	X	56
2	Disconnect Presentation Space (2)	X	X	94
3	Send Key (3)	X	X	151
4	Wait (4)	X	X	182
5	Copy Presentation Space (5)	X	X	75
6	Search Presentation Space (6)	X	X	147
7	Query Cursor Location (7)	X	X	122
8	Copy Presentation Space to String (8)	X	X	80
9	Set Session Parameters (9)	X	X	160
10	Query Sessions (10)	X	X	131
11	Reserve (11)	X	X	143
12	Release (12)	X	X	142
13	Copy OIA (13)	X	X	66
14	Query Field Attribute (14)	X	X	123
15	Copy String to Presentation Space (15)	X	X	88
18	Pause (18)	X	X	112
20	Query System (20)	X	X	133
21	Reset System (21)	X	X	144
22	Query Session Status (22)	X	X	129
23	Start Host Notification (23)	X	X	171
24	Query Host Update (24)	X	X	125
25	Stop Host Notification (25)	X	X	180
30	Search Field (30)	X	X	145
31	Find Field Position (31)	X	X	97
32	Find Field Length (32)	X	X	95
33	Copy String to Field (33)	X	X	85
34	Copy Field to String (34)	X	X	61
40	Set Cursor (40)	X	X	159
41	Start Close Intercept (41)	X	X	168
42	Query Close Intercept (42)	X	X	119
43	Stop Close Intercept (43)	X	X	179
50	Start Keystroke Intercept (50)	X	X	175

Number	Function	PC/3270	PC400	Page
51	Get Key (51)	X	X	100
52	Post Intercept Status (52)	X	X	118
53	Stop Keystroke Intercept (53)	X	X	181
60	LOCK PRESENTATION SPACE API (60)	X	X	109
61	LOCK WINDOW SERVICES API (61)	X	X	106
90	Send File (90)	X	X	149
91	Receive File (91)	X	X	140
99	Convert Position or Convert RowCol (99)	X	X	58
101	Connect Window Services (101)	X	X	54
102	Disconnect Window Service (102)	X	X	93
103	Query Window Coordinates (103)	X	X	127
104	Window Status (104)	X	X	114
105	Change Switch List LT Name (105)	X	X	50
106	Change Window Name (106)	X	X	48
110	Start Playing Macro (110)	X	X	178
120	Connect for Structured Fields (120)	X	–	52
121	Disconnect from Structured Fields (121)	X	–	91
122	Query Communications Buffer Size (122)	X	–	120
123	Allocate Communications Buffer (123)	X	–	46
124	Free Communications Buffer (124)	X	–	99
125	Get Request Completion (125)	X	–	103
126	Read Structured Fields (126)	X	–	135
127	Write Structured Fields (127)	X	–	183

Allocate Communications Buffer (123)

PC/3270	PC400
X	–

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

Prerequisite Calls

None.

Call Parameters

Table 3. Call Parameters

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

The calling data string can contain:

Table 4. Calling Data String

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

Return Parameters

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

Notes on Using This Function

1. The EHLLAPI obtains a buffer from Windows 95 memory management and places the buffer address into the return parameter string. The requested buffer size (length) is also passed in the parameter string. The buffer size can be from 1 byte to 64 KB minus 256 bytes (X'FF00' bytes) in length.

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

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

Change Window Name (106)

PC/3270	*PC400*
X	X

The **Change PS Window Name** function allows the application to specify a new name for the presentation space window or reset the presentation space window to the default name.

Prerequisite Calls

Connect Window Services (101).

Call Parameters

Table 5. Call Parameters

	16-bit	**32-bit**
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 32-bit).

The calling data string can contain:

Table 6. Calling Data String

Byte		**Definition**
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2	5	A change request option value, select one of: X'01' for changing the presentation space window name. X'02' for resetting the presentation space window name.
3–63	6–66	An ASCII string of from 1 (for PC/3270) or 2 (for PC400) to 61 bytes including a terminator byte. The ASCII string must end with a NULL character. This string must contain at least one non-NULL character followed by a NULL character.
	67–68	Reserved

Return Parameters

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

Notes on Using This Function

A string is ended at the first NULL character found. The NULL character overrides the specified string length. If the NULL character is not at the end of the specified length, the last byte at the specified length is replaced by a NULL character, and the remainder of the data string is lost. If the NULL character is found before the specified length, the string is truncated at that point, and the remainder of the data string is lost.

If the application fails to reset the presentation space name before exiting, the exit list processing resets the name.

Change Switch List LT Name (105)

PC/3270	*PC400*
X	X

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

Note: This is for compatibility with Communications Manager EHLLAPI, and has the same result as the **Change PS Window Name** (106) function.

Prerequisite Calls

Connect Window Services (101).

Call Parameters

Table 7. Call Parameters

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

The calling data string can contain:

Table 8. Calling Data String

Byte		**Definition**
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2	5	A change request option value, select one of: X'01' for changing a switch list LT name. X'02' for resetting a switch list LT name.
3–63	6–66	An ASCII string of 2 to 61 bytes including a terminator byte. The ASCII string must end with a NULL character. This string must contain at least one non-NULL character followed by a NULL character.
	67–68	Reserved

Return Parameters

Return Code	Explanation
0	The **Change Switch List LT Name** function was successful.
1	An incorrect host presentation space short session ID was specified, or the host presentation space was not connected.
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)

PC/3270	PC400
X	–

The **Connect for Structured Fields** function allows an application to establish a connection to the emulation program to send and receive structured field data to or from a host application. The workstation application must provide the Query Reply data field and must point to it with in the parameter string. The destination/origin ID returned by the emulator will be returned to the application.

Prerequisite Calls

None.

Call Parameters

Table 9. Call Parameters

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

The calling data string can contain:

Table 10. Calling Data String

Byte		Definition
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2–5	5–8	Address of the Query Reply data buffer.
6–7	9–10	Destination/origin unique ID. (16-bit word, returned)
	11–12	Reserved
8–11	13–16	The data in these position is ignored by EHLLAPI. However, no error is caused if the migrating program has data in these positions. This data is accepted to provide compatibility with migrating applications.

Return Parameters

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

Notes on Using This Function

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

Connect Window Services (101)

PC/3270	PC400
X	X

The **Connect Window Services** function allows the application to manage the presentation space windows. Only one EHLLAPI application at a time can be connected to a presentation space for window services.

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

Prerequisite Calls

None.

Call Parameters

Table 11. Call Parameters

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

The calling data string can contain:

Table 12. Calling Data String

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

Return Parameters

Return Code	Explanation
0	The **Connect Window Services** function was successful.
1	An incorrect host presentation space short session ID was specified, or the host presentation space was not connected.
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.

   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 160.

   A **CONNECT WINDOW SERVICES** function is sufficient for the process. However, each thread that requests a **CONNECT WINDOW SERVICES** must have a corresponding **DISCONNECT WINDOW SERVICES** (102), or one of the threads must issue a **RESET SYSTEM** (21), which affects all threads and disconnects any remaining connections.

Connect Presentation Space (1)

PC/3270	PC400
X	X

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

Prerequisite Calls

None.

Call Parameters

Table 13. Call Parameters

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

The calling data string can contain:

Table 14. Calling Data String

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

Return Parameters

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

Return Code	Explanation
0	The **Connect Presentation Space** function was successful; the host presentation space is unlocked and ready for input.
1	An incorrect host presentation space ID was specified. The specified session either does not exist or is a logical printer session.
4	Successful connection was achieved, but the host presentation space is busy.
5	Successful connection was achieved, but the host presentation space is locked (input inhibited).
9	A system error was encountered.
11	This resource is unavailable. The host presentation space is already being used by another system function.

Notes on Using This Function

1. The **Connect Presentation Space** function is affected by the CONLOG/CONPHYS session option.
2. An EHLLAPI application cannot be connected to multiple presentation spaces concurrently; however, the connections are sequential and the application cannot go back and forth between connections. Calls requiring the **Connect Presentation Space** function as a prerequisite use the currently connected presentation space. For example, if an application is connected to presentation space A, B, and C in that order, the application must connect to B or A again to issue functions.
3. Each thread that requests a **CONNECT PRESENTATION SPACE** must have a corresponding **DISCONNECT PRESENTATION SPACE** (2), or one of the threads must issue a **RESET SYSTEM** (21), which affects all threads and disconnects any remaining connections.
4. More than one EHLLAPI application can share a presentation space, if the applications support sharing (that is, if they were developed to work together and if they exhibit predictable behavior) and have compatible read/write access and keyword options as set in the **SET SESSIONS PARAMETERS** (9) function. For more information, see “Set Session Parameters (9)” on page 160.
5. Because the **CONNECT PRESENTATION SPACE** and **START KEYSTROKE INTERCEPT** (50) functions share common subsystem functions, successful requests by an application to share either of these functions for the same session can affect the request of these two functions by other applications. For example, if application A successfully requests a **CONNECT PRESENTATION SPACE** for a session with Write_Read access and KEY$abcdefgh as the keyword, a request by application B to **CONNECT PRESENTATION SPACE** for a session and **START KEYSTROKE INTERCEPT** is successful only if both applications have set compatible read/write options.
6. To exist, a session must be declared during customization. You cannot connect to a session that is not declared or to a session that is defined as a logical printer session. See the *Personal Communications/3270 Version 4.1 for Windows 95 Up and Running*.

Convert Position or Convert RowCol (99)

PC/3270	*PC400*
X	X

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

Prerequisite Calls

None.

Call Parameters

Function Number: Must be 99.

Data String: Host presentation space short name *and* P for the **Convert Position** function (for example, AP converts the presentation space position of session A),

or

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

Length: Row when R, above, is specified. 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 60.

NA when P, above, is specified.

PS Position: Column when R, above, is specified. 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 60.

Host presentation space position when P, above, is specified. 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 60.

The calling data string can contain:

Table 15. Call Parameters		
Byte		**Definition**
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2	5	**Convert** option P or R
	6–8	Reserved

Return Parameters

This function returns a length and a return code.

Length:

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

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

Return Code:

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

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

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

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

The following status codes are valid:

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

Notes on Using This Function

1. To configure your presentation space, refer to the *Personal Communications/3270 Version 4.1 for Windows 95 Up and Running* or the *Personal Communications AS/400 Version 4.1 for Windows User's Guide*.
2. To find out how many rows and columns are in your presentation space, examine bytes 12 and 14 of the returned data string parameter for the **Query Session Status** (22) function. See “Query Session Status (22)” on page 129.

Copy Field to String (34)

PC/3270	PC400
X	X

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

Function Number: Must be 34.

Data String: Preallocated target data string. When the **Set Session Parameters** (9) function with Extended Attribute Bytes (EAB) option is issued, the length of the data string must be at least twice the length of the field.

DBCS Only: When Extended Attributes Double-byte (EAD) option is specified, the length of the data string must be at least three times the length of the field. When both EAB and EAD options are specified, the length of the data string must be at least four times the length of the field.

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

PS Position: Identifies the target field. This can be the PS position of any byte within the target field. Copy always starts at the beginning of the field.

Return Parameters

This function returns a data string and a return code.

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

Number of bytes specified in the calling length parameter

Number of bytes in the identified field in the host presentation space

Return Code: The following codes are valid:

Return Code	Explanation
0	The **Copy Field to String** function was successful.
1	Your program is not connected to a host session.
2	An error was made in specifying parameters.
6	The data to be copied and the target field are not the same size. The data is truncated if the string length is smaller than the field copied.
7	The host presentation space position is 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 ATTRB/NOATTRB/NULLATTRB (NULLATTRB is valid for Windows 95 only), the EAB/NOEAB, the XLATE/NOXLATE, the DISPLAY/NODISPLAY, the DISPLAY/NODISPLAY (for Windows 95 only), the EAD/NOEAD (for DBCS only), and the NOSO/SPACESO/SO (for DBCS only) session options. Refer to items 5 on page 162; 13 and 14 on page 164; 17 on page 165; and 20 and 21 on page 166 for more information.

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

Set Session Parameter Effect on the COPY Function

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

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

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

Bit Position	Meaning
0–1	Character highlighting 00 = Normal 01 = Blink 10 = Reverse video 11 = Underline
2–4	Character color (Color remap can override this color definition.) 000 = Default 001 = Blue 010 = Red 011 = Pink 100 = Green 101 = Turquoise 110 = Yellow 111 = White
5–6	Character attributes 00 = Default value 11 = Double byte character
7	Reserved

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

Bit Position	Meaning
0	Reverse image 0 = Normal image 1 = Reverse image
1	Underline 0 = No underline 1 = Underline

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

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

Bit Position	Meaning
0–3	Background character colors 0000 = Black 0001 = Blue 0010 = Green 0011 = Cyan 0100 = Red 0101 = Magenta 0110 = Brown (3270), Yellow (5250) 0111 = White
4–7	Foreground character colors 0000 = Black 0001 = Blue 0010 = Green 0011 = Cyan 0100 = Red 0101 = Magenta 0110 = Brown (3270), Yellow (5250) 0111 = White 1000 = Gray 1001 = Light blue 1010 = Light green 1011 = Light cyan 1100 = Light red 1101 = Light magenta 1110 = Yellow 1111 = White (high intensity)

Double-byte character set attributes (for DBCS only)

– The first byte

Bit Position	Character Position	Field Attribute Position
0	Double-byte character	Reserved
1	The first byte of the double-byte character	Reserved
2	SO	Reserved

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.

7. To use this function, preallocate memory to receive the returned data string parameter. The statements required to preallocate this memory vary depending on the language in which your application is written. Refer to “Memory Preallocation” on page 21 for more information.

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

Copy OIA (13)

PC/3270	PC400
X	X

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

Function Number: Must be 13.
Data String: Preallocated target data string.
Data String Length: 103 bytes (104 for 32-bit).
PS Position: NA.

Return Parameters

This function returns a data string and a return code.

Data String: A 103-byte string. See “Format of the Returned OIA Data String” on page 67 for more information.
Return Code: The following codes are valid:

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

Notes on Using This Function

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

depending on the language in which your application is written. Refer to “Memory Preallocation” on page 21 for more information.

Format of the Returned OIA Data String

The OIA data string contains the following information:

Table 16. Calling Data String

Byte		Definition
16-bit	32-bit	
1	1	The OIA format byte. The value is 1 (PC/3270) or 9 (PC400).
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 3 on page 68 shows the hexadecimal codes found in the host presentation space, and the characters they represent. The returned data can be translated into OIA graphics characters. Refer to the *Personal Communications/3270 Version 4.1 for Windows 95 Up and Running* or *Personal Communications AS/400 Version 4.1 for Windows User's Guide* 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 3 on page 68, 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 3 on page 68). 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.

Copy OIA

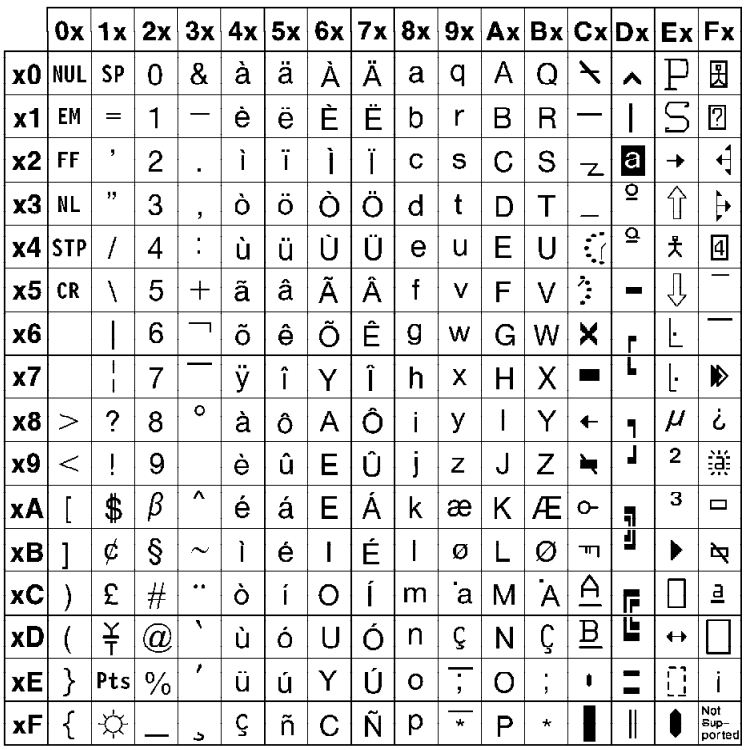

Figure 3. Host Presentation Space Characters

Group 1 (Offset 82): Online and Screen Ownership

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

Group 2 (Offset 83): Character Selection

Bit	Meaning
0	Reserved
1	APL
2	Katakana (Japan only)
3	Alphanumeric
4–5	Reserved
6	Hiragana (Japan only)
7	Double byte character

Group 3 (Offset 84): Shift State

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

Group 4 (Offset 85): PSS Group 1

Bit	Meaning
0–7	Reserved

Group 5 (Offset 86): Highlight Group 1

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

Group 6 (Offset 87): Color Group 1

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

Group 7 (Offset 88): Insert

Bit	Meaning
0	Insert mode
1–7	Reserved

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

– Byte 1 (Offset 89)

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

– Byte 2 (Offset 90)

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

– Byte 3 (Offset 91)

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

– Byte 4 (Offset 92)

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

– Byte 5 (Offset 93)

Bit	Meaning
0–7	Reserved

Group 9 (Offset 94): PSS Group 2

Bit	Meaning
0–7	Reserved

Group 10 (Offset 95): Highlight Group 2

Bit	Meaning
0–7	Reserved

Group 11 (Offset 96): Color Group 2

Bit	Meaning
0–7	Reserved

Group 12 (Offset 97): Communication Error Reminder

Bit	Meaning
0	Communications error

Bit	Meaning
1–7	Reserved

Group 13 (Offset 98): Printer State

Bit	Meaning
0–7	Reserved

Group 14 (Offset 99): Graphics

Bit	Meaning
0–7	Reserved

Group 15 (Offset 100): Reserved

Group 16 (Offset 101): Automatic Key Play/Record State

Bit	Meaning
0–7	Reserved

Group 17 (Offset 102): Automatic Key Quit/Stop State

Bit	Meaning
0–7	Reserved

Group 18 (Offset 103): Expanded State

Bit	Meaning
0–7	Reserved

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

Group 1 (Offset 82): Online and Screen Ownership

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

Group 2 (Offset 83): Character Selection

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

Group 3 (Offset 84): Shift State

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

Group 4 (Offset 85): PSS Group 1

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Group 5 (Offset 86): Highlight Group 1

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Group 6 (Offset 87): Color Group 1

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Group 7 (Offset 88): Insert

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

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

– Byte 1 (Offset 89)

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

– Byte 2 (Offset 90)

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

– Byte 3 (Offset 91)

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

– Byte 4 (Offset 92)

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

– Byte 5 (Offset 93)

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Group 9 (Offset 94): PSS Group 2

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Group 10 (Offset 95): Highlight Group 2

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Group 11 (Offset 96): Color Group 2

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Group 12 (Offset 97): Communication Error Reminder

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

Group 13 (Offset 98): Printer State

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Group 14 (Offset 99): Graphics

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Group 15 (Offset 100): Reserved

Group 16 (Offset 101): Automatic Key Play/Record State

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Group 17 (Offset 102): Automatic Key Quit/Stop State

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Group 18 (Offset 103): Expanded State

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Copy Presentation Space (5)

PC/3270	*PC400*
X	X

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

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

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Function Number: Must be 5.

Data String: Preallocated target string the size of your host presentation space. This can vary depending on how your host presentation space is configured. When the **Set Session Parameters** (9) function with the EAB option is issued, the length of the data string must be at least twice the length of the field.

DBCS Only: When the EAD option is specified, the length of the data string must be at least three times the length of the field. 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 field.

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

PS Position: NA.

Return Parameters

This function returns a data string and a return code.

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

Return Code	Explanation
0	The host presentation space contents were copied to the application program. The target presentation space was active, and the keyboard was unlocked.
1	Your program is not connected to a host session.
4	The host presentation space contents were copied. The connected host presentation space was waiting for host response.
5	The host presentation space was copied. The keyboard was locked.
9	A system error was encountered.

Notes on Using This Function

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

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

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

Refer to items 5 on page 162; 13 and 14 on page 164; 15 on page 164; 17 on page 165; and 20 and 21 on page 166 Refer to items 5, 13, 14, 15, 17, 20,and 21 on page 166 for more information.

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

As previously stated, the return of attributes by the various **COPY** (5, 8, and 34) functions is affected by the **SET SESSION PARAMETERS** (9) function. The involved set session parameters have the following effect:

Set Session Parameter Effect on the COPY Function

NOEAB and NOEAD	Attributes are not returned. Only text is copied from the presentation space to the user buffer.
EAB and NOXLATE	Attributes are returned as defined in the following tables.
EAB and XLATE	The colors used for the presentation space display are returned. Colors can be remapped; so the attribute colors are not the ones returned by the **COPY** functions when XLATE and EAB are on at the same time.
EAD	Double-byte character set attributes are returned as shown in the following tables.
NOSO/SPACESO/SO	When NOSO is specified, it works as SPACESO. The size of the presentation space is not changed.

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

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

Bit Position	Meaning
0–1	Character highlighting 00 = Normal 01 = Blink 10 = Reverse video 11 = Underline
2–4	Character color (Color remap can override this color definition.) 000 = Default 001 = Blue 010 = Red 011 = Pink 100 = Green 101 = Turquoise 110 = Yellow 111 = White
5–6	Character attribute 00 = Default value 11 = Double byte character
7	Reserved

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

Bit Position	Meaning
0	Reverse image 0 = Normal image 1 = Reverse image
1	Underline 0 = No underline 1 = Underline

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

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

Bit Position	Meaning
0–3	Background character colors 0000 = Black 0001 = Blue 0010 = Green 0011 = Cyan 0100 = Red 0101 = Magenta 0110 = Brown (3270), Yellow (5250) 0111 = White
4–7	Foreground character colors 0000 = Black 0001 = Blue 0010 = Green 0011 = Cyan 0100 = Red 0101 = Magenta 0110 = Brown (3270), Yellow (5250) 0111 = White 1000 = Gray 1001 = Light blue 1010 = Light green 1011 = Light cyan 1100 = Light red 1101 = Light magenta 1110 = Yellow 1111 = White (high intensity)

Double-byte character set attributes (for DBCS only)

– The first byte

Bit Position	Character Position	Field Attribute Position
0	Double-byte character	Reserved
1	The first byte of the double-byte character	Reserved
2	SO	Reserved

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

– The second byte

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

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

If you want to copy only a portion of the host presentation space, use the **Copy Presentation Space to String** (8) function.

To use this function, preallocate memory to receive the returned data string parameter. The statements required to preallocate this memory vary depending on the language in which your application is written. Refer to “Memory Preallocation” on page 21 for more information.

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

Copy Presentation Space to String (8)

PC/3270	PC400
X	X

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

Function Number: Must be 8.

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

DBCS Only: When the EAD option is specified, the length of the data string must be at least three times the length of the field. 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 field.

Length: Length of the target data string.

PS Position: Position within the host presentation space of the first byte in your target data string.

Return Parameters

This function returns a data string and a return code.

Data String: Contents of the host presentation space.
Return Code: The following codes are valid:

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

Notes on Using This Function

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

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

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

   Refer to items 5 on page 162; 13 and 14 on page 164; 15 on page 164; 17 on page 165; and 20 and 21 on page 166 for more information.

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

As previously stated, the return of attributes by the various **COPY** (5, 8, and 34) functions is affected by the **SET SESSION PARAMETERS** (9) function. The involved set session parameters have the following effect:

Set Session Parameter Effect on the COPY Function

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

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

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

Bit Position	Meaning
0–1	Character highlighting 00 = Normal 01 = Blink 10 = Reverse video 11 = Underline
2–4	Character color (Color remap can override this color definition.) 000 = Default 001 = Blue 010 = Red 011 = Pink 100 = Green 101 = Turquoise 110 = Yellow 111 = White
5–7	Reserved

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

Bit Position	Meaning
0	Reverse image 0 = Normal image 1 = Reverse image

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

Bit Position	Meaning
0–3	Background character colors 0000 = Black 0001 = Blue 0010 = Green 0011 = Cyan 0100 = Red 0101 = Magenta 0110 = Brown (3270), Yellow (5250) 0111 = White
4–7	Foreground character colors 0000 = Black 0001 = Blue 0010 = Green 0011 = Cyan 0100 = Red 0101 = Magenta 0110 = Brown (3270), Yellow (5250) 0111 = White 1000 = Gray 1001 = Light blue 1010 = Light green 1011 = Light cyan 1100 = Light red 1101 = Light magenta 1110 = Yellow 1111 = White (high intensity)

Double-byte character set attributes

– The first byte

Bit Position	Character Position	Field Attribute Position
0	Double-byte character	Reserved
1	The first byte of the double-byte character	Reserved
2	SO	Reserved

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.

4. To use this function, preallocate memory to receive the returned data string parameter. The statements required to preallocate this memory vary depending on the language in which your application is written. Refer to “Memory Preallocation” on page 21 for more information.

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

Copy String to Field (33)

PC/3270	*PC400*
X	X

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

Function Number: Must be 33.

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

Length: Length, in number of bytes, of the source data string. Overridden if in EOT mode.

PS Position: Identifies the target field. This can be the PS position of any byte within the target field. Copy always starts at the beginning of the field.

Return Parameters

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

Notes on Using This Function

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

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

   Refer to items 1 and 2 on page 161; 13 and 14 on page 164; 18 on page 165; and 20 and 21 on page 166 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 160).

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

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

   Note: If the field at the end of the host presentation space wraps, wrapping occurs when the end of the presentation space is reached.

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

4. The first byte of the data to be transferred is always placed at the beginning of the field that contains the specified PS position.

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

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

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

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

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

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

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

Copy String to Presentation Space (15)

PC/3270	*PC400*
X	X

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

Function Number: Must be 15.

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

Length: Length, in number of bytes, of the source data string. Overridden if in EOT mode.

PS Position: Position in the host presentation space to begin the copy, a value between 1 and the configured size of your host presentation space.

Return Parameters

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

Notes on Using This Function

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

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

   Refer to items 1 and 2 on page 161; 13 and 14 on page 164; 18 on page 165; and 20 and 21 on page 166 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 160).
4. Although the **Send Key** (3) function accomplishes the same purpose, this function responses with the prompt and enters a command more quickly. Because the **Send Key** (3) function emulates the terminal operator typing the data from the keyboard, its process speed is slow for an application operating with a lot of data. This function provides a faster input path to the host.
5. The original data (the copied string) cannot exceed the size of the presentation space.
6. If there is a field attribute or a protected field in the target area, the string and the area are shifted by an amount that equals their length.
7. **DBCS Only:** Double-byte characters can be included as a part of the string.

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

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

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

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

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

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

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

Disconnect from Structured Fields (121)

PC/3270	*PC400*
X	–

The **Disconnect from Structured Fields** function drops the connection between the emulation program and the EHLLAPI application. The EHLLAPI application must disconnect from the emulation program before exiting from the system. The EHLLAPI application should issue this function request.

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

Prerequisite Calls

Connect for Structured Fields (120).

Call Parameters

Table 17. Call Parameters

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

Data String Contents

Table 18. Calling Data String

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

Return Parameters

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

Notes on Using This Function

1. When a **Disconnect from Structured Fields** function is called, any outstanding asynchronous **Read Structured Fields** (126) or **Write Structured Fields** (127) function requests are returned if the application issues the **Get Request Completion** (125) function call. Use the asynchronous form of this function when cleaning up after issuing a Disconnect call.
2. The **Reset System** (21) function will also free any outstanding asynchronous requests (requests that have not been retrieved by the application using the **Get Request Completion** (125) function).

Disconnect Window Service (102)

PC/3270	PC400
X	X

The **Disconnect Window Service** function disconnects the window services connection between the EHLLAPI program and the specified host presentation space window.

Prerequisite Calls

Connect Window Services (101).

Call Parameters

Table 19. Call Parameters

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

Data String Contents

Table 20. Calling Data String

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

Return Parameters

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

Notes on Using This Function

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

Before exiting the application, you should request a **Disconnect Window Service** function for all presentation spaces that have been connected for Presentation Manager services. If the application exits with an outstanding connection for window services, the subsystem cancels the outstanding connection.

Disconnect Presentation Space (2)

PC/3270	PC400
X	X

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

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Table 21. Call Parameters

	16-bit	32-bit
Function Number	Must be 2	
Data String	NA	
Length	NA	NA
PS Position	NA	

Return Parameters

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

Notes on Using This Function

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

Find Field Length (32)

PC/3270	PC400
X	X

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

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

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Table 22. Call Parameters

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

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

The calling 2-character data string can contain:

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

Note: The symbol represents a required blank.

Return Parameters

This function returns a length and a return code.

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 valid:

Return Code	Explanation
0	The **Find Field Length** function was successful.
1	Your program is not connected to a host session.
2	A parameter error was encountered.
7	The host presentation space position is 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 or T is used as the calling data string, if the field found is the same as the field from which the Find started, a return code of 24 is returned.

Find Field Position (31)

PC/3270	*PC400*
X	X

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

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Table 23. Call Parameters

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

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

The calling 2-character data string can contain:

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

Note: The symbol represents a required blank.

Return Parameters

This function returns a length and a return code.

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 valid:

Return Code	Explanation
0	The **Find Field Position** function was successful.
1	Your program is not connected to a host session.
2	A parameter error was encountered.
7	The host presentation space position is 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 or T is used as the calling data string, if the field found is the same as the field from which the Find started, a return code of 24 is returned.

Free Communications Buffer (124)

PC/3270	*PC400*
X	–

The **Free Communications Buffer** function returns to Windows 95 management memory a buffer that is no longer required by the application. The application should free the buffer prior to exiting the system.

Prerequisite Calls

Allocate Communications Buffer (123).

Call Parameters

Table 24. Call Parameters

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

Data String Contents

Table 25. Calling Data String

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

Return Parameters

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

Notes on Using This Function

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

Get Key (51)

PC/3270	PC400
X	X

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

Prerequisite Calls

Start Keystroke Intercept (50).

Call Parameters

Table 26. Call Parameters

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

Data String Contents

Table 27. Calling Data String

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

Return Parameters

This function returns a data string and a return code.

Data String: See the following table:

Table 28 (Page 1 of 2). Calling Data String

Byte		Definition
16-bit	32-bit	

Table 28 (Page 2 of 2). Calling Data String		
Byte		**Definition**
1	1	One of the following values: A 1-character presentation space short name (PSID) A blank or null indicating a function call for the host-connected presentation.
	2–4	Reserved
2	5	An option code character, one of the following characters: A for ASCII returned M for keystroke mnemonic
3–8	6–11	These 6 bytes of the preallocated buffer space are used internally to enqueue and dequeue keystrokes. Possible combinations include: Byte 3 contains an ASCII character and byte 4 contains X'00'. Bytes 3 and 4 contains a double-byte character. Bytes 3 contains the escape character (either @ or another character specified using the ESC=c option of function 9) and byte 4 contains a 1-byte abbreviation for a function. (See "ASCII Mnemonics" on page 34). 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

E At 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').

E M@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 Preallocation” on page 21 for more information.

Get Request Completion (125)

PC/3270	PC400
X	–

The **Get Request Completion** function allows an application to determine the status of a previous asynchronous function request issued to the EHLLAPI and to obtain the function parameter list before using the data string again. This function is valid only if the user specified asynchronous (“A”) completion on a previous function call such as **Read Structured Fields** (126) or **Write Structured Fields** (127).

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

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

Prerequisite Calls

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

Call Parameters

Table 29. Call Parameters

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

Data String Contents

Table 30. Calling Data String

Byte		Definition
16-bit	32-bit	
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	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)

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

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

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

Return Parameters

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

The differences between return codes 38 and 42 are:

1. Return code 38 has three subtle differences:
   a. If a specific request ID and session were requested, both the session and ID were found but the request is pending (not in a completed state).
   b. If a zero request ID and a specific session were requested, the specified session has pending requests, but they are not satisfied (complete).
   c. If a zero request ID and a blank session were requested, pending requests were found but none were satisfied (complete).
2. Return code 42 has three subtle differences:
   a. If a specific request ID and session were requested, the specific request ID was not found in either a pending or a completed state.
   b. If a zero request ID and a specific session were requested, the specific session contains no pending or completed requests.
   c. If a zero request ID and a blank session were requested, no pending or completed requests were found.

Notes on Using This Function

1. This function is valid only if the user specified asynchronous completion ("A" 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 WINDOW SERVICES API (61)

PC/3270	*PC400*
X	–

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

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

Prerequisite Calls

Connect Window Services (101)

Call Parameters

Table 31. Call Parameters

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

Data String Contents

Table 32. Calling Data String

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

Return Parameters

Return Code	Explanation
0	The **LOCK Window Services API** function was successful.
1	An incorrect host presentation space short session ID was specified or was not connected.
2	An error was made in specifying parameters.
9	A system error was encountered.
38	Requested function was not complete.
43	The API was already locked by another EHLLAPI application (on LOCK) or API not locked (on UNLOCK).

Notes on Using This Function

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

WINDOW STATUS (104)
CHANGE SWITCH LIST NAME (105)
CHANGE PS WINDOW NAME (106)

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

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

Successfully issuing the **LOCK Window Services API** with th 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.

LOCK PRESENTATION SPACE API (60)

PC/3270	*PC400*
X	–

The **LOCK PRESENTATION SPACE API** function allows the application to obtain or release exclusive control of the presentation space window over other Windows 95 applications. While locked, no other application can connect to the presentation space window.

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

Prerequisite Calls

CONNECT TO PRESENTATION SPACE (1)

Call Parameters

Table 33. Call Parameters

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

Data String Contents

Table 34. Calling Data String

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

Return Parameters

Return Code	Explanation
0	The **LOCK PRESENTATION SPACE API** function was successful.
1	An incorrect host presentation space short session ID was specified or was not connected.
2	An error was made in specifying parameters.
9	A system error was encountered.
43	The API was already locked by another EHLLAPI application (on LOCK) or API not locked (on UNLOCK).

Notes on Using This Function

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

SEND KEY (3)
COPY PRESENTATION SPACE (5)
SEARCH PRESENTATION SPACE (6)
COPY PRESENTATION SPACE TO STRING (8)
RELEASE (11)
RESERVE (12)
QUERY FIELD ATTRIBUTE (14)
COPY STRING TO PRESENTATION SPACE (15)
SEARCH FIELD (30)
FIND FIELD POSITION (31)
FIND FIELD LENGTH (32)
COPY STRING TO FIELD (33)
COPY FIELD TO STRING (34)
SET CURSOR (40)
SEND FILE (90)
RECEIVE FILE (91)
CONNECT TO PRESENTATION SPACE (1) with the CONPHYS parameter set in a previous SET SESSIONS PARAMETER (9) function call.

These queued requests are not serviced until the lock is removed. When the lock is removed, the queued requests are processed in first-in-first-out (FIFO) order. EHLLAPI functions not listed are run as if there was no lock.

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

Disconnecting from the presentation space while still owning the LOCK.
Issuing the **RESET SYSTEM** (21) function while still owning the LOCK.
Stopping the application while still owning the LOCK.
Stopping the session.
Successfully issuing the **LOCK PRESENTATION SPACE API** with the UNLOCK option.

Before exiting the application, you should unlock any presentation space windows that have been locked with the **LOCK PRESENTATION SPACE API** function. If the application exits with outstanding locks, or a **RESET SYSTEM** (21), or **DISCONNECT PRESENTATION SPACE** (2) function is issued, the locks are released.

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

Pause (18)

PC/3270	PC400
X	X

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

Connect Presentation Space (1).

Call Parameters

Table 35. Call Parameters

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

The calling data string can contain:

Table 36. Calling Data String

Byte		Definition
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–8	Reserved

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 162 for more information.
2. The value entered in the calling length parameter is the maximum number of half-second intervals that the **Pause** function waits. For a pause of 20 seconds, a hex value of 28 (decimal 40) must be passed in the calling length parameter.
3. If you use the IPAUSE option, once a pause has been satisfied by a host event, you should call the **Query Host Update** (24) function to clear the queue prior to the next **Pause** function. The **Pause** function will continue to be satisfied with the pending event until the **Query Host Update** (24) function is completed.
4. A practical maximum value for the **Pause** function is 24 . You should not use the **Pause** function for these kinds of tasks:

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

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

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

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

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

   By using IPAUSE when the **START** function is called, you can make an application wait until the host presentation space or OIA (or both) receives an update. When the receive is completed and the application can issue the **Query** function to determine the changes, PAUSE terminates. Then the application issues the **Search Presentation Space** (6) to check whether the expected update occurred.

Window Status (104)

PC/3270	PC400
X	X

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

Prerequisite Calls

Connect Window Services (101).

Call Parameters

Table 37. Call Parameters

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

The calling data string can contain:

Table 38. Calling Data String

Byte		Definition
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2	5	A request option value, select one of the following values: X'01' for set status X'02' for query for status X'03' for query for extended status
	6	Reserved

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

Table 39. Calling Data String

Byte		Definition
16-bit	32-bit	
3–4	7–8	A 16- or 32-bit word containing the status set bits if the request option is 1 (set status). The following codes are valid return values if the request option is set status: X'0001' Change the window size. (Not valid with minimize, maximize, restore, or move.) X'0002' Move the window. (Not valid with minimize, maximize, size, or restore.) X'0004' ZORDER window replacement. X'0008' Set the window to visible. X'0010' Set the window to invisible. X'0080' Activate the window. (Sets focus to window and places it in the foreground unless ZORDER is specified. In this case, the ZORDER placement is used.) X'0100' Deactivate the window. (Deactivates the window and makes the window the bottom window unless ZORDER is also specified. In this case, the ZORDER placement is used.) X'0400' Set the window to minimized. (Not valid with maximize, restore, size, or move.) X'0800' Set the window to maximized. (Not valid with minimize, restore, size, or move.) X'1000' Restore the window. (Not valid with minimize, maximize, size, or move.)
5–6	9–12	A 16- or 32-bit word containing the X window position coordinate. (Ignored if the move option is not set.)
7–8	13–16	A 16- or 32-bit word containing the Y window position coordinate. (Ignored if the move option is not set.)
9–10	17–20	A 16- or 32-bit word containing the X window size in device units. (Ignored if the size option is not set.)
11–12	21–24	A 16- or 32-bit word containing the Y window size in device units. (Ignored if the size option is not set.)
13–16	25–28	A 16- or 32-bit word containing a window handle for relative window placement. These two words are only for the set option. (Ignored if the ZORDER option is not set.) Valid values are as follows: X'00000003' Place in front of all sibling windows. X'00000004' Place behind all sibling windows.

Window Status

If the request option value is X'02' (query for status):

Table 40. Calling Data String		
Byte		**Definition**
16-bit	32-bit	
3–4	7–8	A 16- or 32-bit word containing X'0000' if the request option is 2 (query for status). The following codes are possible return values if the request option is query for status. More than one state is possible. X'0008' The window is visible. X'0010' The window is invisible. X'0080' The window is activated. X'0100' The window is deactivated. X'0400' The window is minimized. X'0800' The window is maximized.
5–6	9–12	A 16- or 32-bit word containing the X window position coordinate. (Ignored if the move option is not set.)
7–8	13–16	A 16- or 32-bit word containing the Y window position coordinate. (Ignored if the move option is not set.)
9–10	17–20	A 16- or 32-bit word containing the X window size in device units. (Ignored if the size option is not set.)
11–12	21–24	A 16- or 32-bit word containing the Y window size in device units. (Ignored if the size option is not set.)
13–16	25–28	A 16- or 32-bit word containing a window handle for relative window placement. These two words are only for the set option. (Ignored if the ZORDER option is not set.) Valid values are as follows: X'00000003' Place in front of all sibling windows. X'00000004' Place behind all sibling windows.

If the request option value is X'03' (query for extended status) the return status will be the handle of the emulator presentation space window:

Table 41 (Page 1 of 2). Calling Data String		
Byte		**Definition**
16-bit	32-bit	
3–4	7–8	A 16- or 32-bit word containing X'0000' if the request option is 3 (query for extended status). The following codes are possible return values if the request option is query for extended status. More than one state is possible. X'0008' The window is visible. X'0010' The window is invisible. X'0080' The window is activated. X'0100' The window is deactivated. X'0400' The window is minimized. X'0800' The window is maximized.
5–6	9–10	A 16- or 32-bit word containing the current font size in the X-dimension. The value is in screen pels.
7–8	11–12	A 16- or 32-bit word containing the current font size in the Y-dimension. The value is in screen pels.

Table 41 (Page 2 of 2). Calling Data String		
Byte		**Definition**
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.

Post Intercept Status (52)

PC/3270	PC400
X	X

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

Prerequisite Calls

Start Keystroke Intercept (50).

Call Parameters

Table 42. Call Parameters

	16-bit	32-bit
Function Number	Must be 52	
Data String	See the following table	
Length	Must be 2	Must be 8
PS Position	NA	

The calling data string can contain:

Table 43. Calling Data String

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

Return Parameters

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

Query Close Intercept (42)

PC/3270	*PC400*
X	X

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

Prerequisite Calls

Start Close Intercept (41).

Call Parameters

Table 44. Call Parameters

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

The calling data string can contain:

Table 45. Calling Data String

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

Return Parameters

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

Query Communications Buffer Size (122)

PC/3270	PC400
X	–

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

Prerequisite Calls

None.

Call Parameters

Table 46. Call Parameters

	16-bit	32-bit
Function Number	Must be 122	
Data String	See the following table	
Length	Must be 9	Must be 20
PS Position	NA	

The calling data string can contain:

Table 47. Calling Data String

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

Return Parameters

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

Notes on Using This Function

1. There is no way to require the user to use this function. It is not a required function so that the application can be tailored to run on any system.
2. The buffer sizes returned represent the record sizes that are actually transmitted across the medium. For a DDM connection, the 8-byte header supplied in the **Read** and **Write Structured Fields** data buffer is stripped off and 1 byte containing the structured field AID value is prefixed. The application should compare the size of the actual data in the data buffer (which does not include the 8-byte header) with the buffer sizes returned by the **Query Communications Buffer Size** minus 1 byte. For destination/origin connections, the 8-byte header supplied in the **Read** and **Write Structured Fields** data buffer is stripped off and 9 bytes are then prefixed to the data. The application should compare the size of the actual data in the data buffer (which does not include the 8-byte header) with the buffer size returned by the **Query Communications Buffer Size** minus 9 bytes.
3. The maximum buffer sizes returned represent the maximum number of bytes supported by the workstation hardware and by the emulator. The maximum buffer size can be used only if the host is also configured to accept at least these maximum sizes.
4. The optimum buffer sizes returned represent the optimum number of bytes supported by the both the workstation hardware and the emulator. Some network configurations might set transmission limits smaller than these values. In these cases, the data transfer buffer size override value in the emulator configuration profile will be used for structured field support. The **Query Communications Buffer Size** will reflect any buffer size override values entered in the emulator configuration profile.

Query Cursor Location (7)

PC/3270	*PC400*
X	X

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

Function Number: Must be 7.
Data String: NA.
Length: NA.
PS Position: NA.

Return Parameters

This function returns a length and a return code.

Length: Host presentation space position of the cursor.

Return Code: The following codes are valid:

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

Query Field Attribute (14)

PC/3270	*PC400*
X	X

The **Query Field Attribute** function returns the attribute byte of the field containing the input host presentation space position. This information is returned in the returned length parameter.

For the PC/3270, note also that:

The returned length parameter is set to 0 if the screen is unformatted. Attribute bytes are equal to or greater than hex C0.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Table 48. Call Parameters

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

Return Code	**Explanation**
0	The **Query Field Attribute** was successful.
1	Your program is not currently connected to a host session.
7	The host presentation space position is not valid.
9	A system error was encountered.
24	Attribute byte not found or unformatted host presentation space.

Notes on Using This Function

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

3270 field attribute:

Query Field Attribute

Bit Position	Meaning
0–1	Both = 1, field attribute byte.
2	Unprotected/protected 0 = Unprotected data field 1 = Protected field
3	A/N 0 = Alphanumeric data 1 = Numeric data only
4–5	I/SPD 00 = Normal intensity, pen not detectable 01 = Normal intensity, pen detectable 10 = High intensity, pen detectable 11 = Nondisplay, pen not detectable
6	Reserved
7	MDT 0 = Field has not been modified 1 = Field has been modified

5250 field attributes:

Bit Position	Meaning
0	Field attribute flag 0 = Nonfield attribute flag 1 = Field attribute flag
1	Visibility 0 = Nondisplay 1 = Display
2	Unprotected/protected 0 = Unprotected data field 1 = Protected field
3	Intensity 0 = Normal intensity 1 = High intensity
4–6	Field type 000 = Alphanumeric data: All characters are available 001 = Alphabet only: Uppercase and lowercase, comma, period, hyphen, blank, or Dup key are available 010 = Numeric shift: Automatic shift for number 011 = Numeric data only: 0–9, comma, period, plus, minus, blank, or Dup key are available 101 = Numeric data only: 0–9, or Dup key are available 110 = Magnetic stripe reading device data only 111 = Signed-numeric data: 0–9, plus, minus, or Dup key are available
7	MDT 0 = Field has not been modified 1 = Field has been modified

Query Host Update (24)

PC/3270	*PC400*
X	X

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

The **Start Host Notification** (23) function was called (on first call to the **Query Host Update** function only)

The previous call to the **Query Host Update** function (for all calls to the **Query Host Update** function except the first).

Prerequisite Calls

Start Host Notification (23).

Call Parameters

Table 49. Call Parameters

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

The calling data string can contain:

Table 50. Calling Data String

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

Return Parameters

Return Code	Definition
0	No updates have been made since the last call.
1	An incorrect host presentation space was specified.
8	No prior **Start Host Notification** (23) function was called for the host presentation space ID.
9	A system error was encountered.
21	The OIA was updated.
22	The presentation space was updated.
23	Both the OIA and the host presentation space were updated.

Notes on Using This Function

The target presentation space must be specified in the data string, even though a connection to the host presentation space is not necessary to check for updates.

Query Window Coordinates (103)

PC/3270	PC400
X	X

The **Query Window Coordinates** function requests the coordinates for the window of a presentation space. The window coordinates are returned in pels.

Note: (0,0) indicates the top-left of the window.

Prerequisite Calls

Connect Window Services (101).

Call Parameters

Table 51. Call Parameters

	16-bit	32-bit
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:

Table 52. Calling Data String

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

Return Parameters

This function returns a data string and a return code.

Table 53. Calling Data String

Byte		Definition
16-bit	32-bit	
1	1	One of the following values: A 1-character presentation space short session ID. A blank or null indicating a function call for the current connection presentation space.
	2–4	Reserved
2–5	5–8	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 valid:

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

Query Session Status (22)

PC/3270	PC400
X	X

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

Prerequisite Calls

None.

Call Parameters

Table 54. Call Parameters

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

Return Parameters

This function returns a data string and a return code.

Table 55 (Page 1 of 2). Calling Data String

Byte		Definition	
16-bit		32-bit	
1	1	A 1-character presentation space short name (PSID)	
	2–4	Reserved	
2–9	5–12	Session long name (same as short name).	
10	13	Session type (F = 5250 display, G = 5250 printer).	
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.	

Table 55 (Page 2 of 2). Calling Data String			
Byte		**Definition**	
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	21	Reserved.	

Return Code: The following codes are valid:

Return Code	Explanation
0	The **Query Session Status** function was successful.
1	An incorrect host presentation space was specified.
2	An incorrect string length was made.
9	A system error was encountered.

Notes on Using This Function

1. To use this function, preallocate memory to receive the returned data string parameter. The statements required to preallocate this memory vary depending on the language in which your application is written. Refer to “Memory Preallocation” on page 21 for more information.

Query Sessions (10)

PC/3270	PC400
X	X

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

Prerequisite Calls

None.

Call Parameters

Table 56. Call Parameters

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

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

Return Parameters

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

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

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

Table 57. Calling Data String

Byte		Definition
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2–9	5–12	Session long name (same as short name).
10	13	Session type H=host
	14	Reserved
11–12	15–16	Host presentation space size (this is a binary number and is not in display format). This is 0 when the session type is H for a printer session.

Length: The number of host sessions started.
Return Code: The following codes are valid:

Return Code	Explanation
0	The **Query Sessions** function was successful.
2	An incorrect string length was made.
9	A system error was encountered.

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. Refer to item 16 on page 164 for more information.

Notes:

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

Query System (20)

PC/3270	*PC400*
X	X

The **Query System** function can be used by an EHLLAPI application program to determine the level of PC/3270 or PC400 support and other system-related values. This function returns a string that contains the appropriate system data. Most of this information is for use by a service coordinator when you call the IBM Support Center after receiving a return code 9 (a system error was encountered).

The bytes in this returned string are defined below under “Return Parameters.”

Prerequisite Calls

None.

Call Parameters

Table 58. Call Parameters

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

Return Parameters

This function returns a data string and a return code.

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

Table 59 (Page 1 of 2). Calling Data String

Byte		**Definition**
16-bit	32-bit	
1	1	EHLLAPI version number.
2–3	2–3	EHLLAPI level number.
4–9	4–9	EHLLAPI date (month, date, year—for service purposes only).
10–12	10–12	Reserved.
13	13	Hardware base, where: U= Unable to determine, Z= Interrogate positions 28 and 29 for hardware base.
14	14	Program type, where P=PC/3270 or PC400.
15–16	15–16	PC/3270 or PC400 sequence number as a 2-byte ASCII value.
17–18	17–18	PC/3270 or PC400 version as a 2-byte ASCII value.
19	19	Reserved.

Table 59 (Page 2 of 2). Calling Data String

Byte		Definition
20–23	20–23	Extended Error Code 1. This is a printable ASCII string representing a hexadecimal word giving the EHLLAPI component ID and system error number for that function (for service purposes only).
24–27	24–27	Extended Error Code 2. This is a printable ASCII string representing a fault symptom code for the last internal system error. In general, extended error codes represent internal diagnostic information, such as the contents of the computer register that indicated the system error and contained the internal error code.
28–29	28–29	Reserved
	30	Reserved
30–31	31–32	NLS type expressed as a 2-byte binary number.
32	33	1-byte printable ASCII code representing type of monitor being used as follows: A=Personal System/2 Monochrome, B=Personal System/2 Monochrome Model 30, C=CGA, E=EGA, G=MCGA, H=Personal System/2 Model 8514, M=Monochrome, V=VGA, U=Unknown.
33–35	34–36	Reserved.

Return Code

The following codes are valid:

Return Code	Explanation
0	The **Query System** function was successful; data string has been returned.
1	EHLLAPI is not loaded. (PC/3270 only)
2	An incorrect string length was 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. Refer to “Memory Preallocation” on page 21 for more information.

Read Structured Fields (126)

PC/3270	PC400
X	–

The **Read Structured Fields** function allows an application to read structured field data from the host application. If the call specifies “S” (for Synchronous), the application does not receive control until the **Read Structured Fields** is completed. If the call specifies “A” (for Asynchronous), the application receives control immediately after the call. If the call specifies “M” (for Asynchronous, message mode), the application receives control immediately after the call. The application can wait for the message. In any case (“S”, “A”, or “M”), the application provides the buffer address in which the data from the host is to be placed.

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

The return code field in the parameter list might not contain the results of the requested I/O. If the return code is not 0, the request failed. The application must take the appropriate action based on the return code.

If the return code for this request is 0, the application must use the request ID returned with this function call to issue the **Get Request Completion** function call to determine the completion results of the function associated with the request ID. The **Get Request Completion** function call returns the following information:

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

Prerequisite Calls

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

Call Parameters

Table 60. Call Parameters

	16-bit	32-bit
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:

Table 61. Calling Data String		
Byte		**Definition**
32-bit	16-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2	5	S or A or M S = Synchronous. Control is not returned to the application until the read is satisfied. A = Asynchronous. Control is returned immediately to the application, can wait for the event object. M = Asynchronous. Control is returned immediately to the application, can wait for the message.
	6	Reserved
3–4	7–8	2-byte destination/origin ID.
5–8	9–12	4-byte address of the buffer into which the data is to be read. The buffer must be obtained using the **Allocate Communications Buffer** (123) function.
9–10	13–16	Reserved
11–12	17–20	When “M” is specified in position 2 the window handle of the window that receives the message should be set. The message is a return value of RegisterWindowMessage (“PCSHLL”)(not equal 0).
13–14		The data in these positions is ignored by EHLLAPI. However, no erro is caused if the migrating program has data in these positions. This data is accepted to provide compatibility with migrating applications.

Return Parameters

This function returns a data string and a return code.

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

Table 62. Call Parameters		
Byte		**Definition**
16-bit	32-bit	
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 th duration of that request.

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

Table 63. Return Parameters

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

Note: If the function is completed successfully, an application window receive a message. The message is a return value of RegisterWindowMessage ("PCSHLL"). wParam parameter contains Task ID returned by the function call. HIWORD of lParam parameter contains Return Code 0, which shows the function was successful, and LOWORD of lParam parameter contains function number 126.

Return Code: The following codes are valid:

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).
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	1 word	Length of the first (or only) structured field message.
10	1 byte	First nonlength byte of the structured field message.
		⋮
m+7	1 byte	Last byte in the structured field message.

Bytes 0 through 7 are the buffer header. These first 8 bytes are used by the emulation program. The user section of the buffer begins with offset 8. Bytes 8 and 9 contain the number of bytes in the first structured field (a structured field message can contain multiple structured fields), including 2 bytes for bytes 8 and 9. Bytes 8 through *m*+7 are used for the structured field message received from the host (which could contain multiple structured fields).

The using application must furnish the complete buffer with the word at offset 0 set to zero. The buffer length must be in the word at offset 4. The word at offset 6 must be X'C000'. The emulation program will place the data message beginning at offset 8 and place the length of the message in the word at offset 2. The buffer length is not disturbed by EHLLAPI.

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

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

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

The return code is correct.

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

When requested asynchronously, EHLLAPI returns the following values:

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

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

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

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

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

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

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

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

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

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

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

Receive File (91)

PC/3270	PC400
X	X

The **Receive File** function is used to transfer a file from the host session to the workstation session. It is used the same way as the RECEIVE command is used in the PC/3270. The **Receive File** function can be called by an EHLLAPI application program.

Prerequisite Calls

None.

Call Parameters

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.

3270 Session

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

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

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

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

To receive the file from the CICS host system:

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

5250 Session

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

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

Note: Parameters within [ ] can be abbreviated. Available options are listed below. For more information of the options, refer to the online help, *Personal Communications/3270 Version 4.1 for Windows 95 Up and Running*, or *Personal Communications AS/400 Version 4.1 for Windows User's Guide*.

Host System	Options
VM/CMS	ASCII, JISCII, CRLF, APPEND, TIME n, CLEAR, NOCLEAR, PROGRESS, QUIET
MVS/TSO	ASCII, JISCII, CRLF, APPEND, TIME (n), CLEAR, NOCLEAR, PROGRESS, QUIET, AVBLOCK|TRACKS|CYLINDERS
CICS	ASCII, JISCII, CRLF, NOCRLF, BINARY, TIME n, CLEAR, NOCLEAR, PROGRESS, QUIET
OS/400	ASCII, JISCII, CRLF, APPEND, TIME n, CLEAR, NOCLEAR, PROGRESS, QUIET
Note: ASCII is valid in an SBCS session and JISCII is valid in a DBCS session.	

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.

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

Win32 error codes reported by EHLLAPI are greater than 300. To determine the Win32 error code, subtract 300 and refer to the *Win32 Programmer's Reference*.

Notes on Using This Function

1. Four sets of parameters under the **Set Session Parameters** (9) function are related to this function. They are the STRLEN/STREOT, EOT=c, QUIET/NOQUIET and the TIMEOUT=c/TIMEOUT= session options. See items 1 and 2 on page 161 and items 7 and 8 on page 162 for more information.
2. If no path is specified when the **Receive File** function is executed, the received file is stored in the current subdirectory, which is the directory in which your application is running.
3. For the PC/3270, this function cannot interact with a session that is not declared or with a session that is defined as a logical printer session. See the *Personal Communications/3270 Version 4.1 for Windows 95 Up and Running*.

Release (12)

PC/3270	PC400
X	X

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

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)

PC/3270	PC400
X	X

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 things occurs:

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

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)

PC/3270	PC400
X	X

The **Reset System** function reinitializes EHLLAPI to its starting state. The session parameter options are reset to their defaults. Event notification is stopped. The reserved host session is released. The host presentation space is disconnected. Keystroke intercept is disabled.

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

Prerequisite Calls

None.

Call Parameters

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	EHLLAPI is not loaded.
9	A system error was encountered.

Notes on Using this Function

For the PC/3270, this function can be used to check whether EHLLAPI is loaded. Place a call to this function at the start of your application and check for a return code of 1.

Search Field (30)

PC/3270	PC400
X	X

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

Function Number: Must be 30.

Data String: Target string for search.

Length: Length of the target data string. Overridden in EOT mode.

PS Position: Identifies the target field. For SRCHALL, 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 146.

Return Parameters

This function returns a length and a return code.

Length: The following codes are valid:

Length	Explanation
= 0	The string was not found.
> 0	The string was found at the indicated host presentation space position.

Return Code: The following codes are valid:

Return Code	Explanation
0	The **Search Field** function was successful.
1	Your program is not connected to a host session.
2	Parameter error. Either 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

1. Four sets of parameters under the **Set Session Parameters** (9) function are related to this function. They are the SRCHALL/SRCHFROM, STRLEN/STREOT, SRCHFRWD/SRCHBKWD, and the EOT=c session options. See items 1 on page 161 through 4 on page 162 for more information.
2. You can use the **Set Session Parameters** (9) function to determine whether your searches proceed forward (SRCHFRWD) or backward (SRCHBKWD) in a field.
3. The **Search Field** function normally checks the entire field (SRCHALL default mode). However, you can use the function 9 to specify SRCHFROM. In this mode, the calling PS position parameter does more than identify the target field. It also provides a beginning or ending point for the search.

   If the SRCHFRWD option is in effect, the search for the designated string begins at the specified PS position and proceeds toward the end of the field.

   If the SRCHBKWD option is in effect, the search for the designated string begins at the end of the field and proceeds backward toward the specified PS position. If the target string is not found, the search ends at the PS position specified in the calling PS position parameter.
4. **DBCS Only:** If the start position of the specified search function is the second byte in a double-byte character, the search is started from the next character for SRCHFRWD and from the character for SRCHBKWD. If the last character of the specified string is the first byte of a double-byte character, the character is not searched for.

   The search ignores a pair of SO and SI in the presentation space. When you search a double-byte control character, put SO (X'0E') before the character and SI (X'0F') after it. For example, X'0E000C0F' in the data string is treated as a double-byte character FF (X'000C').

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

Search Presentation Space (6)

PC/3270	PC400
X	X

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

Function Number: Must be 6.

Data String: Target string of search.

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

PS Position: Position within the host presentation space where the search is to begin (SRCHFRWD option) or to end (SRCHBKWD option). Overridden in SRCHALL (default) mode.

Return Parameters

This function returns a length and a return code.

Length: The following codes are valid:

Length	Explanation
= 0	The string was not found.
> 0	The string was found at the indicated host presentation space position.

Return Code: The following codes are valid:

Return Code	Explanation
0	The **Search Presentation Space** function was successful.
1	Your program is not connected to a host session.
2	An error was made in specifying parameters
7	The host presentation space position is not valid.
9	A system error was encountered.
24	The search string was not found.

Notes on Using This Function

1. Four sets of parameters under the **Set Session Parameters** (9) function are related to this function. They are the SRCHALL/SRCHFROM, STRLEN/STREOT, SRCHFRWD/SRCHBKWD, and the EOT=c session options. See items 1 on page 161 through 4 on page 162 for more information.
2. You can use the **Set Session Parameters** (9) function to specify SRCHBKWD. When this option is in effect, the search operation locates the *last* occurrence of the string.
3. The **Search Presentation Space** function normally checks the entire host presentation space. However, you can use the **Set Session Parameters** (9) function to specify SRCHFROM. In this mode, the calling PS position parameter specifies a beginning or ending point for the search.

   If the SRCHFRWD option is in effect, the search for the designated string begins at the specified PS position and proceeds toward the end of the host presentation space.

   If the SRCHBKWD option is in effect, the search for the designated string begins at the end of the PS and proceeds backward toward the specified PS position. If the target string is not found, the search ends at the PS position specified in the calling PS position parameter.
4. The SRCHFROM option is also useful if you are looking for a keyword that might occur more than once in the host presentation space.
5. The **Search Presentation Space** function is useful in determining when the host presentation space is available. If your EHLLAPI application is expecting a specific prompt or message before sending data, the **Search Presentation Space** function allows you to check for a prompt message before continuing.
6. **DBCS Only:** If the start position of the specified search function is the second byte in a double-byte character, the search is started from the next character for SRCHFRWD and from the character for SRCHBKWD. If the last character of the specified string is the first byte of a double-byte character, the character is not searched for.

   The search ignores a pair of SO and SI in the presentation space. When you search a double-byte control character, put SO (X'0E') before the character and SI (X'0F') after it. For example, X'0E000C0F' in the data string is treated as a double-byte character FF (X'000C').

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

Send File (90)

PC/3270	PC400
X	X

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

Prerequisite Calls

None.

Call Parameters

Function Number: Must be 90.
Data String: Refer to the examples.
Length: Length of the data string. Overridden if in EOT mode.
PS Position: Must be 0.

Following are examples of the data strings.

3270 Session

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

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

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

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

To send the file to the CICS host system:

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

5250 Session

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

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

Note: Parameters within [ ] can be abbreviated. Available options are listed below. For more information of the options, refer to the online help, the *Personal Communications AS/400 Version 4.1 for Windows User's Guide*.

Host System	Options
VM/CMS	ASCII, JISCII, CRLF, APPEND, LRECL n, RECFM v|f, TIME n, CLEAR, NOCLEAR, PROGRESS, QUIET
MVS/TSO	ASCII, JISCII, CRLF, APPEND, LRECL (n), RECFM (v|f|u), TIME (n), CLEAR, NOCLEAR, PROGRESS, QUIET, BLKSIZE (n), SPACE (n[,m]), AVBLOCK|TRACKS|CYLINDERS
CICS	ASCII, JISCII, CRLF, BINARY, TIME n, CLEAR, NOCLEAR, PROGRESS, QUIET
OS/400	ASCII, JISCII, CRLF, APPEND, SRC, LRECL n, TIME n, CLEAR, NOCLEAR, PROGRESS, QUIET **Note:** ASCII is valid in an SBCS session and JISCII is valid in a DBCS session.

Return Parameters

Return Code	Explanation
2	Parameter error or you have specified a length that is too long (more than 255 bytes) for the EHLLAPI buffer. The file transfer was unsuccessful.
3	File transfer complete.
4	File transfer complete with segmented records.
5	Workstation file name is not valid or not found. File transfer was canceled.
9	A system error was encountered.
27	File transfer terminated because of either a Cancel button or the timeout set by the **Set Session Parameter** (9) function.

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

Win32 error codes reported by EHLLAPI are greater than 300. To determine the Win32 error code, subtract 300 and refer to the *Win32 Programner's Reference*.

Notes on Using This Function

1. Four sets of parameters under the **Set Session Parameters** (9) function are related to this function. They are the QUIET/NOQUIET, STRLEN/STREOT, TIMEOUT=c/TIMEOUT= , and the EOT=c session options. See items 1 and 2 on page 161 plus items 7 and 8 on page 162 for more information.
2. For the PC/3270, this function cannot interact with a session that is not declared or with a session that is defined as a logical printer session. See the *Personal Communications/3270 Version 4.1 for Windows 95 Up and Running*.

Send Key (3)

PC/3270	PC400
X	X

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

Function Number: Must be 3.

Data String: A string of keystrokes, maximum 255. Uppercase and lowercase ASCII characters are represented literally. Function keys and shifted function keys are represented by mnemonics. See “Keyboard Mnemonics” on page 153.

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

1. The parameters under the **Set Session Parameters** (9) function are related to this function. They are the AUTORESET/NORESET, STRLEN/STREOT, EOT=c, ESC=c, and RETRY/NORETRY session options. See items 1 and 2 on page 161, 9 and 10 on page 163, and 19 on page 165 for more information.
2. Keystrokes cannot be sent to the host session when the keyboard is locked or busy. You can check this condition with the **Wait** (4) function.
3. If the host is busy, input might be rejected.

4. The length of the data string must be explicitly defined by the default length parameter, but it can be defined implicitly by the EOT=c option of the **Set Session Parameters** (9) function.

   When explicitly defining length (see item 1), the value for the length parameter passed by the application must be calculated. For this calculation, allow 2 bytes for compound keystrokes such as @E and allow 4 bytes for compound keystrokes such as @A@C.

5. To send special control keys, a compound character coding scheme is used. In this coding scheme, one keystroke is represented by a sequence of two to four ASCII characters. The first and third character are always the escape character. The second and fourth character are always a keycode.

   To send the sequence LOGON ABCDE followed by the Enter key, you would code the string LOGON ABCDE@E. A complete list of these keycodes is represented in "Keyboard Mnemonics" on page 153.

   This compound coding technique allows an ASCII string representation of all necessary keystroke codes without requiring the use of complex hexadecimal key codes.

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

6. Users needing higher levels of performance should use the **Copy String to Field** (33) or **Copy String to Presentation Space** (15) function rather than send keystrokes with the **Send Key** (3) function. But remember, only the **Send Key** (3) function can send the special control keys.

7. Refer to "Set Session Parameters (9)" session option 10 on page 163 (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 64. Mnemonics with Uppercase Alphabetic Characters

Mnemonic	Meaning	Windows	Win95
@B	Left Tab	X	X
@C	Clear	X	X
@D	Delete	X	X
@E	Enter	X	X
@F	Erase EOF	X	X
@H	Help	X	X
@I	Insert	X	X
@J	Jump (Set Focus)	–	X
@L	Cursor Left	X	X
@N	New Line	X	X
@O	Space	X	X
@P	Print	X	X
@R	Reset	X	X
@T	Right Tab	X	X
@U	Cursor Up	X	X
@V	Cursor Down	X	X
@X	DBCS (Reserved)	–	X
@Y	Caps Lock (No action)	–	X
@Z	Cursor Right	X	X

Table 65 (Page 1 of 2). Mnemonics with Uppercase Alphabetic Characters

Mnemonic	Meaning	PC/3270	PC400
@B	Left Tab	X	X
@C	Clear	X	X
@D	Delete	X	X
@E	Enter	X	X
@F	Erase EOF	X	X
@H	Help	–	X
@I	Insert	X	X
@J	Jump (Set Focus)	X	X
@L	Cursor Left	X	X
@N	New Line	X	X
@O	Space	X	X
@P	Print	X	X
@R	Reset	X	X
@T	Right Tab	X	X
@U	Cursor Up	X	X

Table 65 (Page 2 of 2). Mnemonics with Uppercase Alphabetic Characters			
Mnemonic	**Meaning**	**PC/3270**	**PC400**
@V	Cursor Down	X	X
@X	DBCS (Reserved)	X	X
@Y	Caps Lock (No action)	X	X
@Z	Cursor Right	X	X

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

Table 66. Mnemonics with Numbers or Lowercase Characters			
Mnemonic	**Meaning**	**Windows**	**Win95**
@0	Home	X	X
@1	PF1/F1	X	X
@2	PF2/F2	X	X
@3	PF3/F3	X	X
@4	PF4/F4	X	X
@5	PF5/F5	X	X
@6	PF6/F6	X	X
@7	PF7/F7	X	X
@8	PF8/F8	X	X
@9	PF9/F9	X	X
@a	PF10/F10	X	X
@b	PF11/F11	X	X
@c	PF12/F12	X	X
@d	PF13	X	X
@e	PF14	X	X
@f	PF15	X	X
@g	PF16	X	X
@h	PF17	X	X
@i	PF18	X	X
@j	PF19	X	X
@k	PF20	X	X
@l	PF21	X	X
@m	PF22	X	X
@n	PF23	X	X
@o	PF24	X	X
@q	End	–	X
@s	ScrLk (No action)	–	X
@t	Num Lock (No action)	–	X
@u	Page Up	X	X
@v	Page Down	X	X
@x	PA1	–	X
@y	PA2	–	X
@z	PA3	–	X

Table 67. Mnemonics with Numbers or Lowercase Characters

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

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

Table 68 (Page 1 of 2). Mnemonics with @A and @ Uppercase Alphabetic Characters

Mnemonic	Meaning	PC/3270	PC400
@A@C	Test	–	X
@A@D	Word Delete	X	X
@A@E	Field Exit	X	X
@A@F	Erase Input	X	X

Table 68 (Page 2 of 2). Mnemonics with @A and @ Uppercase Alphabetic Characters			
Mnemonic	**Meaning**	**PC/3270**	**PC400**
@A@H	System Request	X	X
@A@I	Insert Toggle	X	X
@A@J	Cursor Select	X	X
@A@L	Cursor Left Fast	X	X
@A@Q	Attention	X	X
@A@R	Device Cancel (Cancels Print Presentation Space)	X	X
@A@T	Print Presentation Space	X	X
@A@U	Cursor Up Fast	X	X
@A@V	Cursor Down Fast	X	X
@A@Z	Cursor Right Fast	X	X

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

Table 69. Mnemonics with @A and @ Lowercase Alphabetic Characters			
Mnemonic	**Meaning**	**PC/3270**	**PC400**
@A@9	Reverse Video	X	X
@A@b	Underscore	X	–
@A@c	Reset Reverse Video	X	–
@A@d	Red	X	–
@A@e	Pink	X	–
@A@f	Green	X	–
@A@g	Yellow	X	–
@A@h	Blue	X	–
@A@i	Turquoise	X	–
@A@j	White	X	–
@A@l	Reset Host Colors	X	–
@A@t	Print (Personal Computer)	X	X
@A@y	Forward Word Tab	X	X
@A@z	Backward Word Tab	X	X

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

Table 70. Mnemonics with @A and @ Alphanumeric (Special) Characters			
Mnemonic	**Meaning**	**PC/3270**	**PC400**
@A@–	Field –	–	X
@A@+	Field +	–	X
@A@<	Record Backspace	–	X

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

Table 71. Mnemonics with @S (Shift) and @ Alphabetic Characters			
Mnemonic	**Meaning**	**PC/3270**	**PC400**
@S@E	Print Presentation Space on Host	–	X
@S@x	Dup	X	X
@S@y	Field Mark	X	X

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

Table 72. Mnemonics Using @X and @Alphabetic Lowercase (For DBCS Only)			
Mnemonic	**Meaning**	**PC/3270**	**PC400**
@X@1	Display SO/SI	X	X
@X@5	Generate SO/SI	–	X
@X@6	Display Attribute	–	X
@X@7	Forward Character	–	X
@X@c	Split vertical bar (¦)	–	X

The following character keys are interpreted as they are.

Table 73 (Page 1 of 2). Alphabetic Keys			
Mnemonic	**Meaning**	**PC/3270**	**PC400**
a–z	a–z	X	X
A–Z	A–Z	X	X
0–9	0–9	X	X
~	~	X	X
#	#	X	X
!	!	X	X
$	$	X	X
%	%	X	X
&	&	X	X
"	"	X	X
'	'	X	X
(	(	X	X
)	)	X	X
*	*	X	X
+	+	X	X
'	'	X	X
–	–	X	X
.	.	X	X
/	/	X	X
:	:	X	X
;	;	X	X
<	<	X	X
>	>	X	X
=	=	X	X
?	?	X	X

Table 73 (Page 2 of 2). Alphabetic Keys			
Mnemonic	**Meaning**	**PC/3270**	**PC400**
{	{	X	X
}	}	X	X
[	[	X	X
]	]	X	X
|	|	X	X

The following table shows the mnemonics using a special character.

Table 74. Mnemonics with Special Character Keys			
Mnemonic	**Meaning**	**PC/3270**	**PC400**
@@	@	X	X
@$	Alternate Cursor (The Presentation Manager Interface only)	X	X
@<	Backspace	X	X

Set Cursor (40)

PC/3270	*PC400*
X	X

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

Function Number: Must be 40.

Data String: NA.

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.

Set Session Parameters (9)

PC/3270	*PC400*
X	X

The **Set Session Parameters** function lets you change certain default session options in EHLLAPI. When EHLLAPI is loaded, the default settings for session options are as indicated by the underscored entries in the tables that appear beginning on page 161. 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 161.

Function Number	Function Name	See Items
1	**Connect Presentation Space**	12,
2	**Disconnect Presentation Space**	12
3	**Send Key**	5, 1, 4, 6, 17,
4	**Wait**	12
5	**Copy Presentation Space**	8, 13, 14, 15, 19, 24 20*, 21*
6	**Search Presentation Space**	5, 1, 2, 7
8	**Copy Presentation Space to String**	8, 13, 14, 15, 19, 24** 20*, 21*
10	**Query Sessions**	18, 24
15	**Copy String to Presentation Space**	5, 1, 13, 14, 16, 24
18	**Pause**	9
30	**Search Field**	5, 1, 2, 7, 24
33	**Copy String to Field**	5, 1, 13, 14, 16, 24*
34	**Copy Field to String**	8, 13, 14, 18, 24 20*, 21*
51	**Get Key**	4
90	**Send File**	5, 1, 10, 3
91	**Receive File**	5, 1, 10, 3
Note: * DBCS only		

Prerequisite Calls

None.

Call Parameters

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.

1. The values in the following table determine how the data string length is defined for functions **Send Key** (3), **Search Presentation Space** (6), **Copy String to Presentation Space** (15), **Search Field** (30), **Copy String to Field** (33), **Send File** (90), and **Receive File** (91).

Value	Explanation
STRLEN	An explicit length is passed for all strings.
STREOT	Lengths are not explicitly coded. Calling (source) data strings are terminated with an EOT character.

2. The statement in the following table is used to specify the character that is used as the end-of-text (EOT) delimiter in the calling (source) data string for EHLLAPI functions **Send Key** (3), **Search Presentation Space** (6), **Copy String to Presentation Space** (15), **Search Field** (30), **Copy String to Field** (33), **Send File** (90), and **Receive File** (91).

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

   To be valid, c must be entered as a 1-byte string literal character with no preceding blanks. The EOT character specified by this statement is used to determine the length of a calling data string only when the STREOT option (see item 1) is in effect.

3. The values in the following table affect the **Search Presentation Space** (6) and **Search Field** (30) search functions.

Value	Explanation
SRCHALL	The **Search Presentation Space** (6) function and **Search Field** (30) function scan the entire host presentation space or field.
SRCHFROM	The **Search Presentation Space** (6) function and **Search Field** (30) function start from a specified PS position (for SRCHFRWD) or end at a specified PS position (for SRCHBKWD).

4. The values in the following table affect the **Search Presentation Space** (6) and **Search Field** (30) search functions. They determine the direction for the search.

Value	Explanation
SRCHFRWD	The **Search Presentation Space** (6) function and **Search Field** (30) function perform in an ascending direction.
SRCHBKWD	The **Search Presentation Space** (6) function and **Search Field** (30) function perform in a descending direction. A search is satisfied if the first character of the requested string starts within the bounds specified for the search.

5. The values in the following table determine how attribute bytes are treated for functions **Copy Presentation Space** (5), **Copy Presentation Space to String** (8), and **Copy Field to String** (34).

Value	Explanation
NOATTRB	Convert all unknown values to blanks.
ATTRB	Pass back all codes that do not have an ASCII equivalent as their original values.
NULLATTRB	Convert all field attributes to null characters.

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

Value	Explanation
FPAUSE	A full-duration pause lasts for however long you specified in the **Pause** (18) function.
IPAUSE	Interruptible pause. After the **Start Host Notification** (23) function is executed, a host event satisfies a pause.

7. The values in the following table determine whether messages generated by file transfer functions **Send File** (90) and **Receive File** (91) are displayed.

Value	Explanation
NOQUIET	SEND and RECEIVE messages are displayed.
QUIET	SEND and RECEIVE messages are not displayed.

8. The statements in the following table determine how long PC/3270 or PC400 EHLLAPI waits before it automatically issues a Cancel during execution of file transfer functions **Send File** (90) and **Receive File** (91). To be valid, c must be an Arabic number 0–9 or a capital letter J–N and must not be preceded by a blank.

Value	Explanation
TIMEOUT=0	A Cancel is automatically issued following a 20-second (approximate) delay.

Value	Explanation
TIMEOUT=c	A Cancel is automatically issued following a specified delay. A 1-character indicator from the table below tells PC/3270 or PC400 how many 30-second cycles it should accept before issuing a Cancel itself. Character / Value (in minutes) / Character / Value (in minutes) 1 0.5 — 8 4.0 2 1.0 — 9 4.5 3 1.5 — J 5.0 4 2.0 — K 5.5 5 2.5 — L 6.0 6 3.0 — M 6.5 7 3.5 — N 7.0

9. The statement in the following table is used to define the escape character for keystroke mnemonics. This session option affects functions **Send Key** (3) and **Get Key** (51). The value of c must be entered as a 1-byte literal character string with no preceding blanks.

Value	Explanation
ESC=c	Specifies the escape character for keystroke mnemonics (@ is the default). Do not leave a blank after the equal sign. A blank is not a valid escape character.

10. The values in the following table determine whether EHLLAPI automatically precedes strings sent using the **Send Key** (3) function with a reset.

Value	Explanation
AUTORESET	EHLLAPI 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 whic the Connect Presentation Space (1) and Disconnect Presentation Space(2) commands function.

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

12. The values in the following table affect the **Wait** (4) function and **Get Key** (51) function. For each value, there are two different effects, one for each function.

Value	Explanation
TWAIT	For the **Wait** (4) function, waits up to a minute before timing out on XCLOCK (X []) or XSYSTEM. For the **Get Key** (51) function, does not return control to your EHLLAPI application program until it has intercepted a key (normal or AID key based on the option specified under the **Start Keystroke Intercept** (50) function).

Value	Explanation
LWAIT	For the **Wait** (4) function, waits until XCLOCK (X [])/XSYSTEM clears. This option is not recommended, because control does not return to your application until the host is available. For the **Get Key** (51) function, does not return control to your EHLLAPI application program until it has intercepted a key (normal or AID key based on the option specified under the **Start Keystroke Intercept** (50) function).
NWAIT	For the **Wait** (4) function, checks status and returns immediately (no wait). For the **Get Key** (51) function, returns return code 25 (keystrokes not available) in the fourth parameter if nothing is queued matching the option specified under the **Start Keystroke Intercept** (50) function.

Note: Use of NWAIT is recommended.

13. The values in the following table affect **Copy Presentation Space** (5), **Copy Presentation Space to String** (8), **Copy String to Presentation Space** (15), **Copy String to Field** (33), and **Copy Field to String** (34). Extended attribute bytes (EAB) include extended character attributes and extended field attributes.

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

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

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

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

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

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

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

Value	Explanation
DISPLAY	Copy nondisplay fields in the presentation space to the target buffer area in the same manner as display fields. Current applications function normally.
NODISPLAY	Do not copy nondisplay fields in the presentation space to the target buffer area. Copy the nondisplay fields to the target buffer as a string of null characters. This allows applications to display the copied buffers in the presentation widow without displaying confidential information, such as passwords.

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

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

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

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

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

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

Value	Explanation
NOEAD	DBCS attribute characters are not passed.

Value	Explanation
EAD	Pass the presentation space data and two attribute characters for the double-byte character set (DBCS). (Users receive 2 bytes for each character other than the data. Therefore, a buffer twice the size of the presentation space must be preallocated.)

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

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

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

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

23. The values in the following table affect the **CONNECT PRESENTATION SPACE**(1) and **CONNECT WINDOW SERVICES**(101) functions. The options specify whether an application can or will share the presentation space to which it is connected with another application. Only one of the following values can be specified with each **SET SESSION PARAMETER** call.

Value	Explanation
SUPER_WRITE	The application allows other applications that allow sharing and have write access permissions to concurrently connect to the same presentation space. The originating application performs supervisory-type functions but does not create errors for other applications that share the presentation space.
WRITE_SUPER	The application requires write access and allows only supervisory application to concurrently connect to its presentation space. This is the default value.
WRITE_WRITE	The application requires write access and allows partner or other applications with predictable behavior to share the presentation space.

Value	Explanation
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 valid:

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

Start Close Intercept (41)

PC/3270	PC400
X	X

The **Start Close Intercept** function allows the application to intercept close requests generated when a user selects the close option from the emulator session window. This function intercepts the close request and discards it until a **Stop Close Intercept** (43) function is requested.

After using this function, your application program can use the **Query Close Intercept** (42) function to determine when a close request has occurred.

Prerequisite Calls

None.

Call Parameters

Table 75. Call Parameters

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

The data string contains the following items.

Table 76. Calling Data String

Byte		Definition
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
6	5	If you request the asynchronous message mode, specify “M”.
	6–8	Reserved
2–3	9–12	When ‘M’ is specified in position 5(6 for 16-bit), the window handle of the window that receives the message should be set. The message is a return value of RegisterWindowMessage (“PCSHLL”) (not equal 0).
	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.

Return Parameters

This function returns a data string and a return code.

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

Table 77. Calling Data String

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

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

Table 78. Calling Data String

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

Note: If a user selects the close option, an application window receives a message. The message is a return value of RegisterWindowMessage("PCSHLL"). wParam parameter contains Task ID returned by the function call. HIWORD of lParam parameter contains Return Code 26, which shows a close intercept occurred, and LOWORD of lParam parameter contains function number 41.

Return Code: The following codes are valid:

Return Code	Explanation
0	The **Start Close Intercept** function was successful.
1	An incorrect host presentation space was specified.
2	A parameter error occurred.
9	A system error occurred.
10	The function is not supported by the emulation program.

Notes on Using This Function

1. The returned event object is in a non-signaled state when the start request is in the signaled state. The event object is in the signaled state each time a close request occurs. To receive notification of multiple close request events, put the event object into the signaled state each time using **SetEvent** or the **Query Close Intercept** (42) function.
2. After using this function, your application program can use the **Query Close Intercept** (42) function to determine when a close request has occurred. The application can wait on the returned event object to determine when the event has occurred.
3. This is not an exclusive call. Multiple applications can request this function for the same short session ID.
4. If there are no applications intercepting close requests for a logical terminal, any subsequent close requests selected by the user from the emulator operations dialog result in a normal stop requested for that logical terminal.

Start Host Notification (23)

PC/3270	PC400
X	X

The **Start Host Notification** function begins the process by which your EHLLAPI application program determines if the host presentation space or OIA have been updated.

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

Prerequisite Calls

None.

Call Parameters

Table 79. Call Parameters

	16-bit	32-bit
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:

Table 80 (Page 1 of 2). Calling Data String

Byte		Definition
16-bit	32-bit	
1	1	One of the following values: A 1-character presentation space short name (PSID) A blank or null indicating a request for the host-connected host presentation space
	2–4	Reserved

Table 80 (Page 2 of 2). Calling Data String

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

Return Parameters

This function returns a data string and a return code.

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

Table 81. Calling Data String

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

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

Table 82. Calling Data String

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

Note: If OIA or presentation space is updated, an application window receives a message. The message is a return value of RegisterWindowMessage ("PCSHLL"). The wParam parameter contains the Task ID returned by the function call. HIWORD of lParam contains Return Code 21(shows the OIA is updated), 22(shows the host presentation space is updated), or 23(shows both the OIA and the host presentation space are updated), and LOWORD of lParam parameter contains function number 23.

Return Code: The following codes are valid:

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

Notes on Using This Function

1. For the PC/3270, this function cannot interact with a session that is not declared or with a session that is defined as a logical printer session. See the *Personal Communications/3270 Version 4.1 for Windows 95 Up and Running*.
2. An application program can issue this function in some host sessions. The **Pause** (18) function can notify the application when one or more host sessions (PS, OIA, or both of them) are updated. The **Query Host Update** (24) function can be used to determine whether a PS, OIA, or both of them have been updated.
3. If the application chooses the asynchronous option, it can wait for the returned event object to determine when a host event has occurred.
4. The event object is initially in a non-signaled state and is signaled each time an appropriate event occurs. To receive notification for multiple events, the application must put the event object into the non-signaled state each time it has been signaled using either the **ResetEvent** or the **Query Host Update** (24) function.
5. The next **Start Host Notification** function is not valid while the option is the same as one of the first **Start Host Notification** function calls.
6. This is not an exclusive call. Multiple applications can request this function for the same short session ID.

Start Keystroke Intercept (50)

PC/3270	PC400
X	X

The **Start Keystroke Intercept** function allows a workstation application to filter any keystrokes sent to a session by a terminal operator. After a call to this function, keystrokes are intercepted and saved until the keystroke queue overflows or until the **Stop Keystroke Intercept** (53) function or **Reset System** (21) function is called. The intercepted keystrokes can be:

Received through the **Get Key** (51) function and sent to the same or another session with the **Send Key** (3) function

Accepted or rejected through the **Post Intercept Status** (52) function

Replaced by other keystrokes with the **Send Key** (3) function

Used to trigger other processes

Prerequisite Calls

None.

Call Parameters

Table 83. Call Parameters

	16-bit	32-bit
Function Number	Must be 50	
Data String	See the following table	
Length	6 or 7	Must be 12
PS Position	NA	

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

The calling data string can contain:

Table 84 (Page 1 of 2). Calling Data String

Byte		Definition
16-bit	32-bit	
1	1	One of the following values: A specific host presentation space short name (PSID) A blank or null indicating a request for the host-connected host presentation space.
	2–4	Reserved

Table 84 (Page 2 of 2). Calling Data String

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

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

Table 85. Calling Data String

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

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

Return Parameters

Return Code	Explanation
0	The **Start Keystroke Intercept** function was successful.
1	An incorrect presentation space was specified.
2	An incorrect option was specified.
4	The execution of the function was inhibited because the target presentation space was busy.
9	A system error was encountered. Release is being used.

Notes on Using This Function

1. If a return code of 31 occurs for the **Get Key** (51) function, either:

   Increase the value of the calling length parameter for this function, or
   Execute the **Get Key** (51) function more frequently.

   An intercepted keystroke occupies 3 bytes in the buffer. The next intercepted keystroke is placed in the adjacent 3 bytes. When the **Get Key** (51) function retrieves a keystroke (first-in first-out, or FIFO), the 3 bytes that it occupied are made available for another keystroke. By increasing the size of the buffer or the rate at which keystrokes are retrieved from the buffer, you can eliminate buffer overflow.

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

2. If option code *D* is provided, EHLLAPI writes intercepted non-AID keys to the presentation space to which they were originally intended, and returns only AID keys to the application.
3. Call the **Stop Keystroke Intercept** (53) function before exiting your EHLLAPI application. Otherwise, keystroke interception remains enabled with unpredictable results.
4. For the PC/3270, this function cannot interact with a session that is not declared or with a session that is defined as a logical printer session. See the *Personal Communications/3270 Version 4.1 for Windows 95 Up and Running*.

Start Playing Macro (110)

PC/3270	*PC400*
X	X

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

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Function Number: Must be 110.
Data String: See the following table.
Length: Must be 10.
PS Position: NA.

Byte	Definition
Positions 1–2	Reserved.
Positions 3–10	Null terminated macro name.

Return Parameters

Return Code	Explanation
0	The **Start Playing Macro** function was successful.
1	The programs is not connected to a host session.
2	An error was made in specifying parameters.
9	A system error was encountered.

Stop Close Intercept (43)

PC/3270	PC400
X	X

The **Stop Close Intercept** function allows the application to turn off the **Start Close Intercept** (41) function. After the application has issued the **Stop Close Intercept** function, subsequent close requests result in a normal stop sent to the logical terminal session.

Prerequisite Calls

Start Close Intercept (41).

Call Parameters

Table 86. Call Parameters

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

The calling data string can contain:

Table 87. Calling Data String

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

Return Parameters

Return Code	Explanation
0	The **Stop Close Intercept** function was successful.
1	An incorrect host presentation space was specified.
2	An error was made in specifying parameters.
8	No previous **Start Close Intercept** (41) function was issued.
9	A system error occurred.
12	The session stopped.

Stop Host Notification (25)

PC/3270	PC400
X	X

The **Stop Host Notification** function disables the capability of the **Query Host Update** (24) function to determine if the host presentation space or OIA has been updated. This function also stops host events from affecting the **Pause** (18) function.

Prerequisite Calls

Start Host Notification (23).

Call Parameters

Table 88. Call Parameters

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

The calling data string can contain:

Table 89. Calling Data String

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

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

Return Parameters

Return Code	Definition
0	The **Stop Host Notification** function was successful.
1	An incorrect host presentation space was specified.
8	No previous **Start Host Notification** (23) function was issued.
9	A system error was encountered.

Stop Keystroke Intercept (53)

PC/3270	*PC400*
X	X

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

Prerequisite Calls

Start Keystroke Intercept (50).

Call Parameters

Table 90. Call Parameters

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

The calling data string can contain:

Table 91. Calling Data String

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

Return Parameters

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

Wait (4)

PC/3270	PC400
X	X

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

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 163.
2. You can use this function to see if the host OIA is inhibited.
3. The **Wait** function is satisfied by the host unlocking the keyboard. Therefore, a return code of 0 does not necessarily mean that the transaction has been completed. To verify completion of the transaction, you should use the **Search Field** (30) function or **Search Presentation Space** (6) function combined with the **Wait** function to look for expected keyword prompts.

Write Structured Fields (127)

PC/3270	PC400
X	–

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 afte the call. The application may wait for the message. In any case (S, A or M), the application provides the buffer address in which data to the host is to be placed.

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

The return code field in the parameter list might not contain the results of the requested I/O. If the return code is not 0, then the request failed. The application must take the appropriate action based on the return code.

If the return code for this request is 0, the application must use the request ID returned with this function call to issue the **Get Request Completion** function call to determine the completion results of the function associated with the request ID. The **Get Request Completion** function call returns the following information:

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

Prerequisite Calls

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

Call Parameters

Table 92. Call Parameters

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

The calling data string can contain:

Table 93. Calling Data String

Byte		Definition
16-bit	32-bit	&rblk.
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 written. The buffer must be obtained using the **Allocate Communications Buffer** (123) function.
9–10	13–16	Reserved.
11–12	17–20	When "M" is specified in position 5 (2 for 16-bit), the window handle of the window that receives the message should be set, The message is a return value of RegisterWindowMessage("PCSHLL") (not equal 0).
13–14		The data in these positions is ignored by EHLLAPI However, no error is caused if the migrating program has data in these positions. This data is accepted to provide compatibility with migrating applications.

Return Parameters

This function returns a data string and a return code.

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

Table 94. Return Parameters

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

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

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

Table 95. Return Parameters

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

Note: If the function is completed successfully, an application window receive a message. The message is a return value of RegisterWindowMessage ("PCSHLL"). The wParam parameter contains the Task ID returned by the function call. HIWORD of lParam parameter contains Return Code 0 which shows the function was successful, and LOWORD of lParam parameter contains function number 127.

Return Code: The following codes are valid:

Return Code	Explanation
0	The **Write Structured Fields** function was successful.
1	A specified host presentation space short session ID was not valid or was not connected.
2	An error was made in specifying parameters.
9	A system error occurred.
11	Resource unavailable (memory unavailable).
34	The message sent inbound to the host was canceled.
35	An outbound transmission from the host was canceled.
36	Request rejected. Lost contact with the host.
37	Failed. The host is inbound disabled.

Notes on Using This Function

1. Return code 35 will be returned when the first **Read Structured Fields** or **Write Structured Fields** is requested after an outbound transmission from the host is canceled. Corrective action is the responsibility of the application.
2. Return code 36 requires that the application disconnect from the emulation program and then reconnect to reestablish communications with the host. Corrective action is the responsibility of the application.
3. Return code 37 will be returned if the host is inbound disabled.

4. The EHLLAPI allows for a maximum of 20 asynchronous requests per application to be outstanding. A return code for unavailable resources (RC=11) is returned if more than 20 asynchronous requests are attempted.
5. If you are using IBM Global Network connections, the maximum number of asynchronous requests is 10.

The structured field data format is as follows:

Offset	Length	Contents
0	1 word	X'0000'
2	1 word	m (message length: the number of bytes of data in the message, the number does not include the buffer header prefix, which contains 8 bytes). This value must be set by the application.
4	1 word	X'0000'
6	1 word	X'0000'
8	1 word	Length of the first (or only) structured field message.
10	1 byte	First nonlength byte of the structured field message.
		⋮
m+7	1 byte	Last byte in the structured field message.

Bytes 0 through 7 are the buffer header. These first 8 bytes are used by the emulation program. The user section of the buffer begins with offset 8. Bytes 8 and 9 contain the number of bytes in the first structured field (a structured field message can contain multiple structured fields) including 2 bytes for bytes 8 and 9. Bytes 8 through *m*+7 are used for the structured field message sent to the host.

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

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

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

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

When requested asynchronously, EHLLAPI returns the following values:

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

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

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

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

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

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

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

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

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

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

These are used to complete the asybchronous Write Structured Fields call

Write Structured Fields

Chapter 8. PCSAPI Functions

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

This chapter describes each individual PCSAPI function in detail. The functions are arranged alphabetically by name.

How to Use PCSAPI

You can write the application program using the PCSAPI in C or C++ for Windows 95. To develop a PCSAPI application, do as follows:

1. Prepare a source code and add the appropriate PCSAPI calls.
2. Include the header file **PCSAPI.H** in the application program.
3. Compile the source code.
4. Link the resultant **.OBJ** files with the appropriate object file or libraries.

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

Page Layout Conventions

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

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” describes the type and description of the parameters to be specified in the PCSAPI function call.

Return Code

“Return Code” lists the codes that must be received by your program after a call to the discussed PCSAPI function.

© Copyright IBM Corp. 1995, 1996

pcsConnect Session

PC/3270	*PC400*
X	X

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

Function Type

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

Parameter Type and Description

char cShortID Presentation space short session ID.

Return Code

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

pcsDisconnect Session

PC/3270	*PC400*
X	X

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

Function Type

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

Parameter Type and Description

char cShortID Presentation space short session ID.

Return Code

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

pcsQueryEmulatorStatus

PC/3270	*PC400*
X	X

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. When this bit is off, the specified session is offline.
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.

pcsQueryWorkstationProfile

PC/3270	*PC400*
X	X

The **pcsQueryWorkstationProfile** function returns the workstation profile name that has been used to invoke the host session. To specify the host session, the short session ID must be used. The workstation profile name is copied to the work buffer prepared by the user.

Function Type

BOOL WINAPI pcsQueryWorkstationProfile(*char cShortSessionID, PSZ lpBuffer***)**

Parameter Type and Description

char cShortSessionID Presentation space short session ID.

PSZ lpBuffer Work buffer to copy a null-terminated workstation profile name.

Return Code

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

pcsStartSession

PC/3270	PC400
X	X

The **pcsStartSession** function starts a host session by using a specified workstation profile (**.WS**). 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 name of the profile to load.

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. This parameter can be one of the following values:

fuCmdShow	Value	Meaning
PCS_HIDE	0	Hides the window.
PCS_SHOW	1	Activates and displays the window.
PCS_MINIMIZE	2	Activates the window and displays it as an icon.
PCS_MAXIMIZE	3	Activates the window and displays it as a maximized window.

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

PC/3270	*PC400*
X	X

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

Function Type

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

Parameter Type and Description

char cShortSessionID Presentation space short session ID.

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

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

Return Code

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

pcsStopSession

Chapter 9. DDE Functions in a 32-Bit Environment

PC/3270 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. &pc3a5is. a DDE server for the open sessions that PC/3270 and PC400 are managing. For more information about DDE, refer to *Microsoft Windows Software Development Kit Guide to Programming*.

Note: If you use DDE functions with Visual Basic, see Chapter 10, “Using DDE Functions with a DDE Client Application” on page 293.

PC/3270 and PC400 DDE Data Items

Microsoft Windows DDE uses a three-level naming scheme to identify data items: application, topic, and item. Table 96 on page 197 describes these levels.

Table 96. Naming Scheme for Data Items

Level	Description	Example
Application	A Windows task or a particular task of an application. In this book, the application is PC/3270 or PC400.	IBM327032
Topic	A specific part of an application.	SessionA
Item	A data object that can be passed in a data exchange. An item is an application-defined data item that conforms to one of the Windows clipboard formats or to a private, application-defined, clipboard format. For more information regarding Windows clipboard formats, refer to *Microsoft Windows Software Development Kit Guide to Programming*.	PS (presentation space)

PC/3270 and PC400 support one application name as Windows DDE server.

Table 97. Applications for PC/3270 and PC400

Application
IBM327032 IBM525032

© Copyright IBM Corp. 1995, 1996

The following table lists the topics you can use.

Table 98. Topics for PC/3270 and PC400

Topics
System SessionA, SessionB, ..., SessionZ LUA_xxxx, LUB_xxxx, ..., LUZ_xxxx

In DDE, *atoms* identify application names, topic names, and data items. Atoms represent a character string that is reduced to a unique integer value. The character string is added to an atom table, which can be referred to for the value of the string associated with an atom. Atoms are created with the GlobalAddAtom function call. Refer to *Microsoft Windows Software Development Kit Guide to Programming* for more information about how to create and use atoms.

Using System Topic Data Items

Applications that provide a DDE interface should also provide a special topic SYSTEM. This topic provides a context for items of information that might be of general interest to an application. The SYSTEM topic for PC/3270 or PC400 contains these associated data items:

Item	Function
Formats	Returns the list of clipboard formats (numbers) that PC/3270 or PC400 is capable of rendering.
Status	Returns information about the status of each PC/3270 or PC400 session.
SysCon	Returns the level of PC/3270 or PC400 support and other system related values.
SysItems	Returns the list of data items that are available when connected to the PC/3270 or PC400 system topic.
Topics	Returns the list of PC/3270 or PC400 topics that are available.

Using Session Topic Data Items

For each Session topic, the following data items are supported:

Item	Function
CLOSE	Retrieves the window close requests.
EPS	Retrieves the session presentation space with additional data.
EPSCOND	Retrieves the presentation space service condition.
FIELD	Retrieves the field in the presentation space of the session.
KEYS	Retrieves the keystrokes.
MOUSE	Retrieves the mouse input.
OIA	Retrieves the operator information area status line.
PS	Retrieves the session presentation space.
PSCOND	Retrieves the session advise condition.
SSTAT	Retrieves the session status.
STRING	Retrieves the ASCII string data.
TRIMRECT	Retrieves the session presentation space within the current trim rectangle.

Using LU Topic Data Items (PC/3270 Only)

For each LU topic, the following data items are supported:

Item	Function
SF	Retrieves the destination/origin structured field data.
SFCOND	Retrieves the query reply data.

DDE Functions

The following table lists the DDE functions that are available for use with PC/3270 or PC400 and the page in this section, where each function is more fully documented.

Function	PC/3270 Windows	PC400	Page
Find Field	X	X	202
Get Keystrokes	X	X	204
Get Mouse Input	X	X	206
Get Number of Close Requests	X	X	209
Get Operator Information Area	X	X	210
Get Partial Presentation Space	X	X	211
Get Presentation Space	X	X	214
Get Session Status	X	X	217
Get System Configuration	X	X	219
Get System Formats	X	X	221
Get System Status	X	X	222
Get System SysItems	X	X	223
Get System Topics	X	X	224
Get Trim Rectangle	X	X	225
Initiate Session Conversation	X	X	226
Initiate Structured Field Conversation	X	–	227
Initiate System Conversation	X	X	228
Put Data to Presentation Space	X	X	229
Search for String	X	X	231
Send Keystrokes	X	X	232
Session Execute Macro	X	X	234
Set Cursor Position	X	X	241
Set Mouse Intercept Condition	X	X	242
Set Presentation Space Service Condition	X	X	245
Set Session Advise Condition	X	X	247
Set Structured Field Service Condition	X	–	249
Start Close Intercept	X	X	250
Start Keystroke Intercept	X	X	252
Start Mouse Input Intercept	X	X	253
Start Read SF	X	–	257
Start Session Advise	X	X	259
Stop Close Intercept	X	X	261
Stop Keystroke Intercept	X	X	262
Stop Mouse Input Intercept	X	X	263
Stop Read SF	X	–	264
Stop Session Advise	X	X	265
Terminate Session Conversation	X	X	266
Terminate Structured Field Conversation	X	–	267
Terminate System Conversation	X	X	268
Write SF	X	–	269

Refer to “Summary of DDE Functions in a 32-bit Environment” on page 287 for a summary of the DDE functions.

Naming Conventions for Parameters

Most DDE parameter names and local variables. These variables have a prefix that indicates the general type of the parameter, followed by one or more words that describe the content of the parameter. Prefixes presented in this book are:

a	Atom
c	Character (a 1-byte value)
f	Bit flags packed into a 16-bit integer
h	16-bit handle
p	Short (16-bit) pointer
lp	Long (32-bit) pointer
w	Short (16-bit) unsigned integer
u	Unsigned integer
sz	Null-terminated character string

Find Field

PC/3270	PC400
X	X

The find field function returns to the client the information of the field specified by the set presentation space service condition function.

Note: The client must set the PS position by using the set presentation service condition function before using this function.

The client sends this following message to receive the field information.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELPARAM(cfFormat, aFIELD) );
```

where:

cfFormat Identifies the format for the field information.
Valid value must be CF_DSPTEXT.

aFIELD Identifies field data item.

The Find Field function supports a new format like Visual Basic. Using the new format, the Find Field function can find a field with specifying its type. The new format is:

FIELD (pos, type)

pos Position where PC/3270 or PC400 starts to search a target field.
type Target field type. The field type are:

Type	Meaning
or T	This field.
P	The previous field, either protected or unprotected.
N	The next field, either protected or unprotected.
NP	The next protected field.
NU	The next unprotected field.
PP	The previous protected field.
PU	The previous unprotected field.

Note: The symbol represents a required blank.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the following information of the field in a DDE data message,

Start PS position
Length
Attribute value

WM_DDE_DATA(hData, aFIELD)

or responds with an ACK message containing status information.

WM_DDE_ACK(wStatus, aFIELD)

If PC/3270 or PC400 cannot return the field information, the following status codes will be returned in the low order byte of the wStatus word:

Return Code	Explanation
1	PS position is not valid.
2	PS is unformatted.
6	The specified format is not valid.
9	A system error occurred.

Structure of the Field Information

PC/3270 or PC400 returns the field information in the following structure:

```
typedef struct tagFINDFIELD
{
  unsigned  char   data[(sizeof(DDEDATA)-1)];
   unsigned short uFieldStart;       /  Field start offset
   unsigned short uFieldLength;     /  Field Length
   unsigned char    cAttribute;        /  Attribute character
   unsigned char    ubReserved;        /     Reserved
} FINDFIELD;

typedef union tagDDE_FINDFIELD
{
  DDEDATA         DDEdata;
  FINDFIELD     DDEfield;
} DDE_FINDFIELD,  lpDDE_FINDFIELD;
```

Get Keystrokes

PC/3270	PC400
X	X

The Get Keystrokes function returns to the client the keystrokes that are intercepted by the Start Keystroke Intercept function. The client sends the following message to receive the keystroke information.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELPARAM(cfFormat, aKEYS) );
```

where:

cfFormat Identifies the format for the keystroke information. Valid value must be CF_DSPTEXT.

aKEYS Identifies keystroke data item.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the keystrokes in a DDE data message, or responds with one of these ACK messages containing status information:

WM_DDE_DATA(hData, aKEYS)
WM_DDE_ACK(wStatus, aKEYS)

If PC/3270 or PC400 cannot return the keystroke information, the following status codes will be returned in the low-order byte of the wStatus word:

Return Code	Explanation
2	No keystroke was intercepted.
6	The specified format is not valid.
9	A system error occurred.

Structure of the Keystroke Information

PC/3270 or PC400 returns the keystroke information in the following structure:

```
typedef struct tagKEYSTROKE
{
  unsigned  char   data[(sizeof(DDEDATA)-1)];
   unsigned short uTextType;          /  Type of keystrokes
  unsigned  char   szKeyDataffl1“;     /   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                /  Pure text, no HLLAPI commands
PCS_HLLAPITEXT 1            /  Text, including HLLAPI tokens
```

Get Mouse Input

PC/3270	PC400
X	X

The Get Mouse Input function returns the latest mouse input that are intercepted by Start Mouse Input Intercept function to the client.

Note: The client must call the Start Mouse Input Intercept function before using this function.

The client sends the following command to receive the mouse input information.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELPARAM(cfFormat, aMOUSE) );
```

where:

cfFormat Identifies the format for the presentation space. Valid values are CF_TEXT or CF_DSPTEXT. The structure of the mouse input data, in these two formats, is shown below.

aMOUSE Identifies the mouse input as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the mouse input data in a DDE data message, or responds with one of these ACK messages:

WM_DDE_DATA(hData, aMOUSE)
WM_DDE_ACK(wStatus, aMOUSE)

If PC/3270 or PC400 cannot return the mouse input information, the following status codes are returned in the low-order byte of the wStatus word:

Return Code	Explanation
2	No mouse input information was intercepted.
6	The specified format is not valid.
9	A system error occurred.

Structure of the Mouse Input Information

If the format is CF_TEXT, PC/3270 or PC400 returns the mouse input information in the following format:

```
typedef struct tagMOUSE_CF_TEXT
{
   unsigned 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 positio
   unsigned char Tab2[1];             / Tab character
   unsigned char PSColPos[4];       / Col number of Mouse positio
   unsigned char Tab3[1];             / Tab character
   unsigned char PSSize[4];         / Size od Presentation Space
   unsigned char Tab4[1];             / Tab character
   unsigned char PSRows[4];          / Row number of PS
   unsigned char Tab5[1];             / Tab character
   unsigned char PSCols[4];         / Column number of PS
   unsigned char Tab6[1];             / Tab character
   unsigned char Button[1];         / Type of clicked mouse butto
   unsigned char Tab7[1];             / Tab character
   unsigned char Click[1];          / Type of clicking
   unsigned char Tab8[1];             / Tab character
   unsigned char zClickString[1];/ Retrived 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	Size of the presentation space	
PSRows	Number of rows of presentation space	
PSCols	Number of columns of presentation space	
ButtonType	Type of the clicked mouse button	**L** Left button **M** Middle button **R** Right button
ClickType	Type of clicking	**S** Single click **D** Double click
ClickString	Retrieved string to which the mouse pointed	A character string terminated with a '\0'
Tab1–8	A tab character for delimiter	'\t'

If the format is CF_DSPTEXT, PC/3270 or PC400 returns the mouse input information in the following format:

```
typedef struct tagMOUSE_CF_DSPTEXT
{
  unsigned  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]; / Retrived 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 mouse was clicked	0 ... (uPSSize – 1)
uPSRowPos	Row number of the position where mouse was clicked	0 ... (uPSRows – 1)
uPSColPos	Column number of the position where mouse was clicked	0 ... (uPSCols – 1)
uPSSize	Size of the presentation space	
uPSRows	Number of rows of the presentation space	
uPSCols	Number of columns of the presentation space	
uButtonType	Type of the clicked mouse button	**0x0001** Left button **0x0002** Middle button **0x0003** Right button
uClickType	Type of clicking	**0x0001** Single click **0x0002** Double click
szClickString	Retrieved string that the mouse pointed to	A character string terminated with a '\0'

Get Number of Close Requests

PC/3270	PC400
X	X

The Get Number of Close Requests function returns to the client the number of the close requests that are intercepted by the Start Close Intercept function. The client sends the following message to receive the number of the close requests.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELPARAM(cfFormat, aCLOSE) );
```

where:

cfFormat Identifies the format for the close intercept information. Valid value must be CF_DSPTEXT.

aCLOSE Identifies close intercept data item.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the number of the close requests in a DDE data message, or responds with one of these ACK messages:

WM_DDE_DATA(hData, aCLOSE)
WM_DDE_ACK(wStatus, aCLOSE)

If PC/3270 or PC400 cannot return the close intercept information, the following status codes will be returned in the low order byte of the wStatus word:

Return Code	Explanation
6	The specified format is not valid.
9	A system error occurred.

Structure of the Number of the Close Requests Information

PC/3270 or PC400 returns the close intercept information in the following structure:

```
typedef struct tagCLOSEREQ
{
  unsigned  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

PC/3270	PC400
X	X

The Get Operator Information Area (OIA) function returns a copy of the OIA to the client. The client sends the following message to request the OIA.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELPARAM(cfFormat, aOIA) );
```

where:

cfFormat Identifies the format for the OIA. For the OIA this format must be CF_DSPTEXT.

aOIA Identifies the operator information area as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the OIA in a DDE data message, or responds with one of these ACK messages:

WM_DDE_DATA(hData, aOIA)
WM_DDE_ACK(wStatus, aOIA)

If PC/3270 or PC400 cannot return the OIA, the following status codes will be returned in the low order byte of the wStatus word:

Return Code	Explanation
6	The specified format is not valid.
9	A system error occurred.

Structure of the Operator Information Area

PC/3270 or PC400 returns the operator information area in the following structure:

```
typedef struct tagOIADATA
{
  unsigned  char   data[(sizeof(DDEDATA)-1)];
  unsigned  char   OIA[8 ];
} OIADATA;

typedef union tagDDE_OIADATA
{
  DDEDATA      DDEdata;
  OIADATA      DDEoia;
} DDE_OIADATA,  lpDDE_OIADATA;
```

Get Partial Presentation Space

PC/3270	PC400
X	X

The Get Partial Presentation Space function returns all or part of the session presentation space to the client.

Note: The client must set the start PS position and the length (or set the EOF flag) by using the Set Presentation Space Service Condition function before using this function.

The client sends the following command to get the presentation space.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              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 presentation space atom as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the presentation space data, or responds with one of these ACK messages containing an error code in the low order byte of the wStatus word:

WM_DDE_DATA(hData, aEPS)
WM_DDE_ACK(wStatus, aEPS)

If PC/3270 or PC400 cannot return the presentation space, the following status codes will be returned in the low order byte of the wStatus word:

Return Code	Explanation
1	No prior Set Presentation Space Service Condition function was called, or an incorrect parameter was set.
6	The specified format is not valid.
9	A system error occurred.

Structure of the Presentation Space

PC/3270 or PC400 returns the part of the presentation space in the format specified in the Get Partial Presentation Space request.

If the format is CF_DSPTEXT, PC/3270 or PC400 returns the presentation space in the following format:

```
typedef struct tagEPS_CF_DSPTEXT
{
  unsigned  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&rbk.;/  PS + Field list Array(lpPSFIELDS)
} EPS_CF_DSPTEXT;

typedef union tagDDE_EPS_CF_DSPTEXT
{
  DDEDATA                DDEdata;
  EPS_CF_DSPTEXT   DDEeps;
} DDE_EPS_CF_DSPTEXT,  lpDDE_EPS_CF_DSPTEXT;

# The PSFIELDS structure is replaced with below structure.

typedef struct tagPSFIELDS
{
   unsigned short uFieldStart;       /  Field start offset
   unsigned short uFieldLength;      /  Field Length
   unsigned char    cAttribute;         /  Attribute character
   unsigned char    ubReserved;         /     Reserved
} PSFIELDS,  lpPSFIELDS;
```

Note: The following examples show how to obtain long pointers to the PS and the PSFIELDS array.

```
lpDDE = (lpDDE_EPS_CF_DSPTEXT)GlobalLock(hData);
lpps = lpDDE->DDEeps.PSData + lpDDE->DDEeps.uPSOffset;
lppsfields = lpDDE->DDEeps.PSData + lpDDE->DDEeps.uFieldOffset;
```

If the format is CF_TEXT, PC/3270 or PC400 returns the part of the presentation space in the following format:

```
typedef struct tagEPS_CF_TEXT
{
  unsigned  char   data[(sizeof(DDEDATA)-1)];
   unsigned char    PSPOSITION[4];/  Postion of part of the PS
   unsigned char    Tab1[1];          /  Tab character
  unsigned  char   PSLENGTH[4];  /  Length  of  the  part  of  the  PS
   unsigned char    Tab2[1];          /  Tab character
   unsigned char    PSROWS[4];       /  Number of rows in the PS
   unsigned char    Tab3[1];          /  Tab character
   unsigned char    PSCOLS[4];       /  Number of Cols in the PS
   unsigned char    Tab4[1];          /  Tab character
  unsigned  char   PS[1];             /   PS
} EPS_CF_TEXT;

typedef union tagDDE_EPS_CF_TEXT
{
  DDEDATA          DDEdata;
  EPS_CF_TEXT   DDEeps;
} DDE_EPS_CF_TEXT,  lpDDE_EPS_CF_TEXT;
```

Following the PS in the buffer is the following additional structure of fields that compose the field list.

```
typedef struct tagFL_CF_TEXT
{
   unsigned char    Tab5[1];             /  Tab character
  unsigned  char   PSFldCount[4];  /  Number  of  fields  in  the  PS
   unsigned char    Tab6[1];             /  Tab character
   PS_FIELD           Field[1];           /   Field  List  Array
} FL_CF_TEXT,  lpFL_CF_TEXT;

typedef struct tagPS_FIELD
{
  unsigned  char   FieldStart[4];
  unsigned  char   TabF1[1];
  unsigned  char   FieldLength[4];
  unsigned  char   TabF2[1];
```

Note: The following examples show how to obtain long pointers to the PS and the PS_FIELD array.

```
lpDDE = (lpDDE_EPS_CF_TEXT)GlobalLock(hData);
lpps = lpDDE->DDEeps.PS;
lpps_field = lpDDE->DDEeps.PS
             +  atoi(lpDDE->DDEeps.PSLENGTH)
              + ((atoi(lpDDE->DDEeps.PSROWS) -1)   2) // CR/LF
              + 1 + 1 + 4 + 1;    // Tabs + size of field count
```

Get Presentation Space

PC/3270	PC400
X	X

The Get Presentation Space function returns the session presentation space to the client. The client sends the following command to get the presentation space.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              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 presentation space atom as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the presentation space and a list of the fields that comprise the presentation space, or responds with one of these ACK messages containing an error code in the low order byte of the wStatus word:

WM_DDE_DATA(hData, aPS)
WM_DDE_ACK(wStatus, aPS)

If PC/3270 or PC400 cannot return the presentation space, the following status codes will be returned in the low order byte of the wStatus word:

Return Code	Explanation
4	The specified format is not valid.
9	A system error occurred.

Structure of the Presentation Space

PC/3270 or PC400 returns the presentation space in the format specified in the Get Presentation Space request.

If the format is CF_DSPTEXT, PC/3270 or PC400 returns the presentation space in the following format:

```
typedef struct tagPS_CF_DSPTEXT
{
  unsigned  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    PSDataffl1";          /  PS and Field list Array(lpPSFIELDS)
} PS_CF_DSPTEXT;

typedef union tagDDE_PS_CF_DSPTEXT
{
  DDEDATA               DDEdata;
  PS_CF_DSPTEXT   DDEps;
} DDE_PS_CF_DSPTEXT,  lpDDE_PS_CF_DSPTEXT;

typedef struct tagPSFIELDS
{
   unsigned short uFieldStart;       /  Field start offset
   unsigned short uFieldLength;     /  Field Length
   unsigned char    cAttribute;        /  Attribute character
   unsigned char    ubReserved;        /     Reserved
} PSFIELDS,  lpPSFIELDS;
```

Note: The following examples show how to obtain long pointers to the PS and the PSFIELDS array.

```
lpDDE = (lpDDE_PS_CF_DSPTEXT)GlobalLock(hData);
lpps = lpDDE->DDEps.PSData + lpDDE->DDEps.uPSOffset;
lppsfields = lpDDE->DDEps.PSData + lpDDE->DDEps.uFieldOffset;
```

If the format is CF_TEXT, PC/3270 or PC400 returns the presentation space in the following format:

```
typedef struct tagPS_CF_TEXT
{
  unsigned  char   data[(sizeof(DDEDATA)-1)];
   unsigned char    PSSIZE[4];          /  Size of the PS
   unsigned char    Tab1[1];            /  Tab character
   unsigned char    PSROWS[4];            /  Number of rows in the PS
   unsigned char    Tab2[1];            /  Tab character
   unsigned char    PSCOLS[4];           /  Number of Cols in the PS
   unsigned char    Tab3[1];            /  Tab character
  unsigned  char   PS[1];               /   PS
} PS_CF_TEXT;

typedef union tagDDE_PS_CF_TEXT
{
  DDEDATA         DDEdata;
  PS_CF_TEXT   DDEps;
} DDE_PS_CF_TEXT,  lpDDE_PS_CF_TEXT;
```

Following the PS in the buffer is the following additional structure of fields that compose the field list.

```
typedef struct tagPS_FIELD
{
  unsigned  char   FieldStart[4];
  unsigned  char   TabF1[1];
  unsigned  char   FieldLength[4];
  unsigned  char   TabF2[1];
  unsigned  char   Attribute;
  unsigned  char   TabF3[1];
} PS_FIELD,  lpPS_FIELD;
```

Note: The following example shows how to obtain long pointers to the PS and the PS_FIELD array.

```
lpDDE = (lpDDE_PS_CF_TEXT)GlobalLock(hData);
lpps = lpDDE->DDEps.PS;
lpps_field = lpDDE->DDEps.PS
               +  atoi(lpDDE->DDEps.PSSIZE)
                + ((atoi(lpDDE->DDEps.PSROWS) -1)   2) // CR/LF
                + 1 + 1 + 4 + 1;    // Tabs + size of field count
```

Get Session Status

PC/3270	PC400
X	X

The Get Session Status function returns the status of the connected session. The client sends the following message to request session status:

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELPARAM(cfFormat, aSSTAT) );
```

where:

cfFormat Identifies the DDE format for the status information. The value used is CF_TEXT.

aSSTAT Identifies session status as the data item requested.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the session status in a DDE data message, or responds with one of these ACK messages containing status information:

WM_DDE_DATA(hData, aSSTAT)
WM_DDE_ACK(wStatus, aSSTAT)

If PC/3270 or PC400 cannot return the session status, the following status codes will be returned in the low order byte of the wStatus word:

Return Code	Explanation
6	The specified format is not valid.
9	A system error occurred.

Format of Status Information

PC/3270 or PC400 returns the session status as text in CF_TEXT format. The following fields are returned with the following possible values:

Fields	Returned values	Description
Status	Closed, Invisible, Maximized, Minimized, Normal	The window is in one of these states.
Usage	DDE, User	The session is connected in either a DDE session or a user session.
ScreenX	NN	Defines the horizontal size of the screen.
ScreenY	NN	Defines the vertical size of the screen.
CursorX	NN	Defines the horizontal position of the cursor. (0 ... ScreenX – 1)

Fields	Returned values	Description
CursorY	NN	Defines the vertical position of the cursor. (0 ... ScreenY – 1)
TrimRect Status	Closed, Moved, Sized	The current status of the trim rectangle.
Trim Rectangle X1	N	The top-left corner X position of the trim rectangle in character coordinates.
Trim Rectangle Y1	N	The top-left corner Y position of the trim rectangle in character coordinates.
Trim Rectangle X2	N	The lower-right corner X position of the trim rectangle in character coordinates.
Trim Rectangle Y2	N	The lower-right corner Y position of the trim rectangle in character coordinates.
Session Presentation Space Status	N	The current status of the presentation space. The following values are possible: 1. 0: The presentation space is unlocked. 2. 4: The presentation space is busy. 3. 5: The presentation space is locked.
Session Window Handle	XXXX	Window handle of the session.

Notes:

The status of each field is updated each time the status is requested.
A new field might be added in a future version of PC/3270 or PC400.

Get System Configuration

PC/3270	PC400
X	X

The Get System Configuration function returns the level of PC/3270 or PC400 support and other system-related values. Most of this information is for use by a service coordinator when a customer calls the IBM Support Center after receiving a system error.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELPARAM(cfFormat, aSYSCON) );
```

where:

cfFormat Identifies the DDE format for the data item requested. The value used is CF_TEXT.

aSYSCON
Identifies system configuration as the data item requested.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the system configuration data item in a DDE DATA message, or responds with one of these ACK messages containing status information:

WM_DDE_DATA(hData, aSYSCON)
WM_DDE_ACK(wStatus, aSYSCON)

If PC/3270 or PC400 cannot return the system configuration, a DDE ACK message will be returned with an error code in the low-order byte of the wStatus word:

WM_DDE_ACK(wStatus, aSYSCON)

Return Code	Explanation
9	A system error occurred.

Format of System Configuration information

PC/3270 or PC400 returns the system configuration as text in CF_TEXT format. The following fields are returned with the following possible values:

Fields	Returned values	Description
Version	N	The version of PC/3270 or PC400
Level	NN	The level of PC/3270 or PC400
Date	MMDDYY	The date of PC/3270 or PC400
System model base number and submodel number	XXXX	The following numbers are examples only: FC00 or FC01 = PC AT FC02 = PC XT Model 286 FC04 = PS/2 Model 50 FC05 = PS/2 Model 60 F800 or F801 = PS/2 Model 80 For more information, see *IBM Personal System/2 and Personal Computer BIOS Interface Technical Reference*.
Monitor Type	MONO, CGA, EGA, VGA, XGA	Type of the monitor
Country Code	NNNN	Country code used with 3270 or 5250 terminal session

Get System Formats

PC/3270	*PC400*
X	X

The Get System Formats function returns the list of Windows clipboard formats supported by PC/3270 or PC400. The client application sends the following message to retrieve the format list supported by PC/3270 or PC400:

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELPARAM(cfFormat, aFORMATS) );
```

where:

cfFormat Identifies the DDE format for the data item requested. The value used is CF_TEXT.

aFORMATS
Identifies formats as the data item requested.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the list of supported Windows clipboard formats in CF_TEXT format in a DDE DATA message.

```
WM_DDE_DATA(hData, aFORMATS)
```

The following Windows Clipboard formats are supported by PC/3270 or PC400:

```
CF_TEXT
CF_DSPTEXT
```

If PC/3270 or PC400 cannot return the formats data item, a DDE ACK message will be returned with an error code in the low-order byte of the wStatus word:

```
WM_DDE_ACK(wStatus, aFORMATS)
```

Return Code	Explanation
9	A system error occurred.

Get System Status

PC/3270	*PC400*
X	X

The Get System Status function returns the status of each 3270 or 5250 session that is available with the current PC/3270 or PC400 configuration. The client application sends the following message to retrieve the status data item:

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELPARAM(cfFormat, aSTATUS) );
```

where:

cfFormat Identifies the DDE format for the data item requested. The value used is CF_TEXT.

aSTATUS
Identifies status as the data item requested.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the status data item in CF_TEXT format in a DDE DATA message:

```
WM_DDE_DATA(hData, aSTATUS)
```

For each opened session, PC/3270 or PC400 returns a line of status information. Each line contains a series of fields with the following range of values:

Fields	Range of values	Description
Session ID	A, B, ..., Z	The short ID of the session.
Host Type	370, 400	The host system currently supported by PC/3270 or PC400.
Emulation Type	3270, 5250	The emulation type supported by PC/3270 or PC400.
Session Status	Closed, Invisible, Normal, Minimized, Maximized	The current status of the session's window.

If PC/3270 or PC400 cannot return the status data item, a DDE ACK message will be returned with an error code in the low-order byte of the wStatus word:

```
WM_DDE_ACK(wStatus, aSTATUS)
```

Return Code	Explanation
9	A system error occurred.

Get System SysItems

PC/3270	*PC400*
X	X

PC/3270 or PC400 supports the DDE system topic so that a client application can connect to the system topic and retrieve information about PC/3270 or PC400 and the status of the sessions that PC/3270 or PC400 is managing.

The Get System SysItems function returns the list of data items available in the PC/3270 or PC400 system topic. The client application sends the following message to get the system topic data items:

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELPARAM(cfFormat, aSYSITEMS) );
```

where:

cfFormat Identifies the DDE format for the data item requested. The value used is CF_TEXT.

aSYSITEMS
Identifies SysItems as the data item requested.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the list of system topic data items in CF_TEXT format in a DDE DATA message.

```
WM_DDE_DATA(hData, aSYSITEMS)
```

The following list of data items are supported by PC/3270 or PC400:

SysItems
Topics
Status
Formats
SysCon

If PC/3270 or PC400 cannot return the system data items, a DDE ACK message will be returned with an error code in the low-order byte of the wStatus word:

```
WM_DDE_ACK(wStatus, aSYSITEMS)
```

Return Code	Explanation
9	A system error occurred.

Get System Topics

PC/3270	*PC400*
X	X

The Get System Topics function returns the list of active DDE topics currently supported by PC/3270 or PC400. The client application sends the following message to the system topic to retrieve the list of topics that are currently active:

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELPARAM(cfFormat, aTOPICS) );
```

where:

cfFormat Identifies the DDE format for the data item requested. The value used is CF_TEXT.

aTOPICS Identifies topics as the data item requested.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the list of DDE topics in CF_TEXT format in a DDE DATA message.

```
WM_DDE_DATA(hData, aTOPICS)
```

The following list of topics are supported by PC/3270 or PC400:

System – System Topic
SessionA – Session A Topic[1]
⋮
SessionZ – Session Z Topic[1]

If PC/3270 or PC400 cannot return the list of topics, a DDE ACK message will be returned with an error code in the low-order byte of the wStatus word:

```
WM_DDE_ACK(wStatus, aTOPICS)
```

Return Code	Explanation
9	A system error occurred.

[1] The actual number of session topics supported is dependent on the number of sessions currently opened. The client program should always query the topics data item of the system topic to obtain the list of sessions currently opened.

Get Trim Rectangle

PC/3270	*PC400*
X	X

The Get Trim Rectangle function returns to the client the area of the presentation space that is within the current trim rectangle. The client sends the following message to receive the trim rectangle.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELPARAM(cfFormat, aTRIMRECT) );
```

where:

cfFormat Identifies the format for the trim rectangle.
Valid value is CF_TEXT.

aTRIMRECT
Identifies trim rectangle as the data item requested.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns trim rectangle in a DDE data message, or responds with one of these ACK messages:

WM_DDE_DATA(hData, aTRIMRECT)
WM_DDE_ACK(wStatus, aTRIMRECT)

If PC/3270 or PC400 cannot return the trim rectangle, the following status codes will be returned in the low-order byte of the wStatus word:

Return Code	Explanation
6	The specified format is not valid.
9	A system error occurred.

Initiate Session Conversation

PC/3270	*PC400*
X	X

The Initiate Session Conversation function connects a client application to an available session of PC/3270 or PC400. Once a session conversation has been established, the session is reserved for exclusive use by the client until the conversation is terminated.

The client application sends the following message to initiate a DDE conversation with a session:

```
SendMessage( -1,
             WM_DDE_INITIATE,
             hClientWnd,
              MAKELPARAM(aIBM327 32, aSessionN) );
```

where:

aIBM327032
Identifies the application atom. The string used to create atom aIBM327032 is “IBM327032”. In the PC400, the application atom is aIBM525032 and the string “IBM525032” is used to create it.

aSessionN
Identifies the topic atom. The string used to create atom aSessionN is either NULL or “Session” appended with the session ID “A”, “B”, ..., “Z”.

PC/3270 or PC400 Response

If a specific topic is selected and PC/3270 or PC400 can support a conversation with the client application, PC/3270 or PC400 acknowledges the INITIATE transaction with:

```
WM_DDE_ACK(aIBM327 32, aSessionN)
```

If a topic is not selected (aSessionN = NULL), PC/3270 or PC400 responds by acknowledging all topics that are currently available:

```
WM_DDE_ACK(aIBM327 32, aSystem)
WM_DDE_ACK(aIBM327 32, aSessionA)
⋮
WM_DDE_ACK(aIBM327 32, aSessionZ)
```

The client application selects the conversation it wishes to communicate with from the returned list of topics and terminates all other unwanted conversations.

Initiate Structured Field Conversation

PC/3270	*PC400*
X	X

The Initiate Structured Field Conversation function connects a client application and a host application. This allows the applications to send data to each other and to receive data from each other.

The client sends the following command to initiate a structured field conversation:

```
SendMessage( -1,
             WM_DDE_INITIATE,
             hClientWnd,
              MAKELPARAM(aIBM327 32, aLUN_xxxx) );
```

Where:

aIBM327032
Identifies the application atom.

aLUN_xxxx
Identifies the topic atom. The string used to create atom aLUN_xxxx is "LU" appended with the session ID "A", "B", ..., "Z", appended with "_", and appended with the user-defined string of any length.

PC/3270 response

If PC/3270 can support a structured field conversation with the client application, it will return an acknowledgment message with the following parameter:

```
WM_DDE_ACK(aIBM327 32, aLUN_xxxx)
```

Initiate System Conversation

PC/3270	*PC400*
X	X

The Initiate System Conversation function connects a client application to the system conversation. Only one client can be connected to the system conversation at a given time. The client sends the following command to initiate a system conversation:

```
SendMessage( -1,
             WM_DDE_INITIATE,
             hClientWnd,
              MAKELPARAM(aIBM327 32, aSystem) );
```

where:

aIBM327032
Identifies the application atom.

aSystem Identifies the topic atom.

PC/3270 or PC400 Response

If PC/3270 or PC400 can support a system topic conversation with the client application, it returns an acknowledgment message with the following parameters:

```
WM_DDE_ACK(aIBM327 32, aSystem)
```

Put Data to Presentation Space

PC/3270	PC400
X	X

The Put Data to Presentation Space function sends an ASCIIZ data string to be written into the host presentation space at the location specified by the calling parameter. The client sends the following message to the session to send the string.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             PackDDElParam(WM_DDE_POKE,
              hdata, aEPS) );
```

where:

hData

Identifies a handle to a Windows global memory object that contains the string to be sent to the session. The global memory object contains the following structure:

```
typedef struct tagPutString
{
  unsigned  char   poke[(sizeof(DDEPOKE)-1)];
   unsigned short uPSStart;                          / PS Position
   unsigned short uEOFflag;                           / EOF effective switch
   unsigned char    szStringData[1];          / String Data
} PUTSTRING;

typedef union tagDDE_PUTSTRING
{
  DDEPOKE       DDEpoke;
  PUTSTRING   DDEputstring;
} DDE_PUTSTRING,  lpDDE_PUTSTRING;
```

These values are valid at the uEOFflag field:

```
PCS_UNEFFECTEOF            / The string is not truncated at EOF.
PCS_EFFECTEOF     1        / The string is truncated at EOF.
```

aEPS Identifies the presentation space atom as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 receives the string data and sends them to the presentation space, and returns a positive ACK message.

If the presentation space does not accept the string data, PC/3270 or PC400 returns a negative ACK message containing the following status codes in the low-order byte of the wStatus word:

```
WM_DDE_ACK(wStatus, aEPS)
```

Return Code	Explanation
1	PS position is not valid.
2	Length is not valid.
3	The value of EOF flag is not valid.
5	Input to the target PS was inhibited.
6	The specified format is not valid.
7	The string was truncated (successful putting).
9	A system error occurred.

Search for String

PC/3270	PC400
X	X

This function allows a client application to examine the presentation space for a specified string in a specified area.

Note: The client must set the start PS position, search direction, a string to be searched, and EOF flag by using the Set Presentation Space Service Condition function before using this function.

The client sends the following message to search for the string.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELPARAM(cfFormat, aSTRING) );
```

where:

cfFormat Identifies the format for the search information. Valid value is CF_DSPTEXT.

aSTRING Identifies the search data item.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the start position of the string in a DDE data message if the string was found in the specified area:

WM_DDE_DATA(hData, aSTRING)
WM_DDE_ACK(wStatus, aSTRING)

If PC/3270 or PC400 cannot return the start position of the string, the following status codes will be returned in the low-order byte of the wStatus word:

Return Code	Explanation
1	PS position is not valid or the string is too long.
2	The string cannot be found.
6	The specified format is not valid.
9	A system error occurred.

Structure of the Search Information

PC/3270 or PC400 returns the Search information in the following structure:

```
typedef struct tagSEARCH
{
  unsigned  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

PC/3270	PC400
X	X

The Send Keystrokes function sends keystrokes to the connected session. The client sends the following message to the session to send keystrokes.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             PackDDElParam(WM_DDE_POKE,
              hData, aKEYS) );
```

where:

hData Identifies a handle to a Windows global memory object that contains the keystrokes to be sent to the session. The global memory object contains the following structure:

```
typedef struct tagKeystrokes
{
  unsigned  char   poke[(sizeof(DDEPOKE)-1)];
   unsigned short uTextType;                    / Type of keystrokes
   unsigned short uRetryCount;                  / Retry count 1 .. 16
  unsigned  char   szKeyData[1];          /   Keystrokes
} KEYSTROKES;

typedef union tagDDE_SENDKEYSTROKES
{
  DDEPOKE        DDEpoke;
  KEYSTROKES   DDEkeys;
} DDE_SNEDKEYSTROKES,  lpDDE_SENDKEYSTROKES;
```

The following key text types are supported:

```
PCS_PURETEXT              / Pure text, no HLLAPI commands
PCS_HLLAPITEXT 1          / Text, including HLLAPI tokens
```

Note: If the keystrokes are pure text then specifying WC_PURETEXT will transfer the keystrokes to the host in the fastest possible manner. If WC_HLLAPITEXT is specified then the keystroke data can contain HLLAPI commands interspersed with the text.

aKEYS Identifies keystrokes as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 receives the keystrokes and sends them to the presentation space. If the presentation space does not accept the keystrokes, a reset is sent to the presentation space and the keystrokes are sent again. This procedure continues until the presentation space accepts the keystrokes or the retry count is reached. If PC/3270 or PC400 cannot send the keystrokes to the host, PC/3270 or PC400 returns a negative ACK message containing the following status codes in the low-order byte of the wStatus word. Otherwise, PC/3270 or PC400 returns a positive ACK message signalling the keystrokes have been sent.

WM_DDE_ACK(wStatus, aKEYS)

Return Code	Explanation
1	Retry count is not valid.
2	Type of key strokes is not valid.
6	The specified format is not valid.
9	A system error occurred.

Session Execute Macro

PC/3270	PC400
X	X

You can issue commands and macros with the DDE_EXECUTE function. The DDE_EXECUTE function passes command strings to PC/3270 or PC400. The command strings must conform to DDE specifications. Refer to *Microsoft Windows Software Development Kit Guide to Programming* for more information about the DDE command syntax.

The client sends the following command to issue a DDE_EXECUTE function.

```
PostMessage ( hServerWnd,
              WM_DDE_EXECUTE,
              hClientWnd,
              (LPARAM)hCommands)  );
```

where:

hCommands
Identifies a handle to a Windows global memory object containing PC/3270 or PC400 commands. For a list of commands you can issue, see “Issuing Commands with the Session Execute Macro Function.”

PC/3270 or PC400 Response

If PC/3270 or PC400 can process the command string, PC/3270 or PC400 returns an ACK message containing positive status information to the client. If PC/3270 or PC400 cannot perform the command string, PC/3270 or PC400 returns an ACK message containing this error code in the low-order word of the wStatus word:

Return Code	Explanation
9	A system error occurred.

Issuing Commands with the Session Execute Macro Function

You can issue the following commands with the Session Execute Macro function:

WINDOW
KEYBOARD
SEND
RECEIVE
SENDKEY
WAIT

Use a separate DDE_EXECUTE message for each command.

Notes:

Enclose values that contain nonalphanumeric characters or blanks in double quotation marks ("*value value*").
To include a double quotation mark within a string, type it twice (for example, "This is a double quotation mark:""").
The maximum length for any command is 255 characters.

WINDOW Command

[WINDOW(action, "name")]

Performs window actions, where:

action = HIDE|RESTORE|MAXIMIZE|MINIMIZE|
SHOW|CNGNAME
name = LT name or Switch List Entry name

Note: *name* should be specified when CNGNAME is specified at *action*. If *name* is a NULL string, the default caption will be displayed.

KEYBOARD Command

[KEYBOARD(action)]

Enables or disables the session keyboard, including the mouse, where:

action = LOCK|UNLOCK

SEND Command

[SEND("pcfilename","hostfilename","options")]

Sends files to the host, where:

pcfilename = [path]filename[.ext]
hostfilename =
For VM system:
filename filetype[filemode]
For MVS system:
[']filename[(membername)][']
For CICS system:
filename or _
For OS/4 system:
library name filename member name

options includes any combination of the following file transfer options: MVS, VM, CICS, QUIET, OS/400, and emulation-specific transfer options, separated by spaces.

Refer to *Personal Communications/3270 Version 4.1 for Windows 95 Reference* or *Personal Communications/3270 Version 4.1 for Windows 95 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:
filename or _
For OS/4 system:
library name filename member name

options includes any combination of the following file transfer options: MVS, VM, CICS, QUIET, OS/400, and emulation-specific transfer options, separated by spaces.

Refer to *Personal Communications/3270 Version 4.1 for Windows 95 Reference* or *Personal Communications/3270 Version 4.1 for Windows 95 Reference* for more information about the transfer options.

SENDKEY Command

[SENDKEY(token,token)]

Sends keystrokes to PC/3270 or PC400, where:

token = text string | command | macro macroname

Notes:

Text strings are enclosed in double quotation marks.
Macros are prefixed with “macro”.
The argument string for SENDKEY must be 255 characters or less in size.
The following commands are supported.

Table 99 (Page 1 of 3). SENDKEY Command List

Command Name	Token	PC/3270	PC400
Alternate Cursor	[alt cursor]	X	X
Alternate Viewing Mode	[alt view]	X	X
Attention	[sys attn]	X	X
Backspace	[backspace]	X	X
Back Tab	[backtab]	X	X
Backtab Word	[backtab word]	X	X
Character Advance	[character advance]	–	X
Character Backspace	[backspace valid]	–	X
Clear Screen	[clear]	X	X
Clicker	[click]	X	X
Color Blue	[blue]	X	–
Color Field Inherit	[field color]	X	–
Color Green	[green]	X	–
Color Pink	[pink]	X	–
Color Red	[red]	X	–
Color Turquoise	[turquoise]	X	–
Color White	[white]	X	–
Color Yellow	[yellow]	X	–
Cursor Blink	[cursor blink]	X	X
Cursor Down	[down]	X	X
Cursor Left	[left]	X	X
Cursor Right	[right]	X	X
Cursor Select	[cursor select]	X	X
Cursor Up	[up]	X	X
Delete Character	[delete char]	X	X
Delete Word	[delete word]	X	X
Device Cancel	[device cancel]	X	X
Dup Field	[dup]	X	X
Edit Clear	[edit-clear]	X	X
Edit Copy	[edit-copy]	X	X

Table 99 (Page 2 of 3). SENDKEY Command List

Command Name	Token	PC/3270	PC400
Edit Cut	[edit-cut]	X	X
Edit Paste	[edit-paste]	X	X
Edit Undo	[edit-undo]	X	X
End Field	[end field]	X	X
Enter	[enter]	X	X
Erase EOF	[erase eof]	X	X
Erase Field	[erase field]	X	–
Erase Input	[erase input]	X	X
Fast Cursor Down	[fast down]	X	X
Fast Cursor Left	[fast left]	X	X
Fast Cursor Right	[fast right]	X	X
Fast Cursor Up	[fast up]	X	X
Field Exit	[field exit]	–	X
Field Mark	[field mark]	X	X
Field +	[field +]	–	X
Field –	[field -]	–	X
Graphic Cursor	[+cr]	X	–
Help	[help]	X	X
Highlighting Field Inherit	[field hilight]	X	–
Highlighting Reverse	[reverse]	X	–
Highlighting Underscore	[underscore]	X	–
Home	[home]	X	X
Host Print	[host print]	X	–
Input	[input]	X	X
Input nondisplay	[input nd]	X	X
Insert Toggle	[insert]	X	X
Lower case	[to lower]	X	–
Mark Down	[mark down]	X	X
Mark Left	[mark left]	X	X
Mark Right	[mark right]	X	X
Mark Up	[mark up]	X	X
Move Mark Down	[move down]	X	X
Move Mark Left	[move left]	X	X
Move Mark Right	[move right]	X	X
Move Mark Up	[move up]	X	X
New Line	[newline]	X	X
Next Page	[page down]	–	X
Pause 1 second	[pause]	X	X
Previous Page	[page up]	–	X
Print Screen	[local copy]	X	X
Program Attention Key 1	[pa1]	X	–
Program Attention Key 2	[pa2]	X	–
Program Attention Key 3	[pa3]	X	–

Table 99 (Page 3 of 3). SENDKEY Command List

Command Name	Token	PC/3270	PC400
Program Function Key 1 ⋮ Program Function Key 24	[pf1] ⋮ [pf24]	X ⋮ X	X ⋮ X
Quit	[quit]	X	X
Reset	[reset]	X	X
Response Time Monitor	[rtm]	X	–
Roll Down	[roll down]	–	X
Roll Up	[roll up]	–	X
Rubout	[rubout]	X	X
Rule	[rule]	X	X
SO/SI Display	[so si]	X	X
SO/SI Generate	[so si generate]	–	X
System Request	[sys req]	X	X
Tab Field	[tab field]	X	X
Tab Word	[tab word]	X	X
Test	[test request]	–	X
Unmark	[unmark]	X	X
Upper case	[to upper]	X	–
Upper/Lower Change	[to other]	X	–
Wait for bind	[wait app]	X	X
Wait for System	[wait sys]	X	X
Wait transition	[wait trn]	X	X
Wait while input inh.	[wait inp inh]	X	X
Window Relocation 1 ⋮ Window Relocation 8	[view 1] ⋮ [view 8]	X ⋮ X	X ⋮ X

WAIT Command

[WAIT("[time out][wait condition]")]

Waits until the timeout expires or the wait condition the client specified occurs. For this command, the client has to set at least one option, where:

time out (optional)
: If the client sets a timeout value in the command statements, the following units are available in the wait statement.

  msec
  millisecond
  milliseconds
  sec
  second
  seconds
  minute
  minutes
  hour
  hours

wait condition (optional)

For the wait condition option, the client can select the following options:

while cursor at (cursor row, cursor column)
: While the cursor is at (cursor row, cursor column), it keeps waiting.

while “string”
: While the “string” is somewhere on the screen, it keeps waiting.

while “string” at (cursor row, cursor column)
: While the “string” is at (cursor row, cursor column) on the screen, it keeps waiting.

until cursor at (cursor row, cursor column)
: Until the cursor moves to (cursor row, cursor column), it keeps waiting.

until “string”
: Until the “string” is displayed somewhere on the screen, it keeps waiting.

until “string” at (cursor row, cursor column)
: Until the “string” is displayed at (cursor row, cursor column), it keeps waiting.

Examples:

1. To wait 10 seconds

   ```
   [WAIT("1  seconds")]
   ```

2. To wait while "ABCDEF" is displayed at (2,9) on the screen

   ```
   [WAIT("while ""ABCDEF"" at (2,9)")]
   ```

3. To wait until "ABCDEF" is displayed at (2,9) on the screen, or after 8 seconds

   ```
   [WAIT("8 seconds until ""ABCDEF"" at (2,9)")]
   ```

Set Cursor Position

PC/3270	PC400
X	X

The Set Cursor Position function allows the client application to set the cursor position in the session window.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             PackDDELParam(WW_DDE_POKE,
              hData, aSETCURSOR) );
```

where:

hData

Identifies a handle to a Windows global memory object that contains the cursor positioning information in the following structure:

```
typedef struct tagSETCURSOR
{
  unsigned  char   poke[(sizeof(DDEPOKE)-1)];
   unsigned short uSetCursorType; /  Cursor Set Type
   unsigned short uSetCursor1;       /  Cursor Row or PS Offset
   unsigned short uSetCursor2;       /  Cursor Col
} SETCURSOR;

typedef union tagDDE_SETCURSOR
{
  DDEPOKE       DDEpoke;
  SETCURSOR   DDEsetcursor;
} DDE_SETCURSOR,  lpDDE_SETCURSOR;
```

PC/3270 or PC400 supports two ways to set the cursor position:

PS Offset (uSetCursorType = 0)
Row/Column number (uSetCursorType = 1)

The application specifies which method by setting the uSetCursorType field to the appropriate value, followed by setting the two other fields uSetCursor1 and uSetCursor2 to their appropriate values as follows:

uSetCursorType = 0 offset

- uSetCursor1: 0 ... (PSsize – 1)

uSetCursorType = 1 row/col

- uSetCursor1: 0 ... (PSrows – 1)
- uSetCursor2: 0 ... (PScols – 1)

aSETCURSOR

Identifies cursor position as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 receives the cursor information and moves the cursor to the specified position in the PS. If the cursor is positioned successfully, PC/3270 or PC400 returns a positive ACK message to the client application. Otherwise, a negative ACK message is returned with the following error codes in the low-order byte of the wStatus word.

```
WM_DDE_ACK(wStatus, aSETCURSOR)
```

Return Code	Explanation
1	Cursor set type is not valid. Must be 0 or 1.
2	Cursor PS offset is not valid. Must be 0 ... (PSsize – 1).
3	Cursor row value is not valid. Must be 0 ... (PSrows – 1).
4	Cursor column value is not valid. Must be 0 ... (PScols – 1).
6	The specified format is not valid.
9	A system error occurred.

Set Mouse Intercept Condition

PC/3270	*PC400*
X	X

This function specifies the mouse input to be intercepted. The client sends the following command to set the mouse event to be intercepted.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             PackDDElParam(WM_DDE_POKE,
              hData, aMOUSE) );
```

where:

hData Identifies a handle to a Windows global memory object that specifies the condition of intercepting the mouse input.

If the format is CF_TEXT, the client program sends the condition in the following structure:

```
typedef struct tagSETMOUSE_CF_TEXT
{
   unsigned char poke[(sizeof(DDEPOKE)-1)];
   unsigned char zMouseCondition[1];
} SETMOUSE_CF_TEXT;

typedef union tagDDE_SETMOUSE_CF_TEXT
{
  DDEPOKE                DDEpoke;
  SETMOUSE_CF_TEXT   DDEcond;
} DDE_SETMOUSE_CF_TEXT,  lpDDE_SETMOUSE_CF_TEXT;
```

The following table shows the parameters' values:

Parameter Name	Meaning	Value
Condition	Condition of intercepting the mouse input	A string terminated with '\0', consists of the constants defined as follows in any order: **L** Enable intercepting the left button **l** Disable intercepting the left button **R** Enable intercepting the right button **r** Disable intercepting the right button **M** Enable intercepting the middle button **m** Disable intercepting the middle button **S** Enable intercepting a single click **s** Disable intercepting a single click **D** Enable intercepting a double click **d** Disable intercepting a double click **T** Retrieve the pointed string **t** Do not retrieve the pointed string

If the format is CF_DSPTEXT, the client program sends the condition in the following structure:

```
typedef struct tagSETMOUSE_CF_DSPTEXT
{
   unsigned char poke[(sizeof(DDEPOKE)-1)];
   BOOL              bLeftButton;        / Enable intercepting left button
   BOOL              bRightButton;       / Enable intercepting right button
   BOOL              bMiddleButton;      / Enable intercepting middle button
   BOOL              bSingleClick;       / Enable intercepting single click
   BOOL              bDoubleClick;       / Enable intercepting double click
   BOOL              bRetrieveString; /  Enable intercepting retrieve string
} SETMOUSE_CF_DSPTEXT;

typedef union tagDDE_SETMOUSE_CF_DSPTEXT
{
  DDEPOKE                  DDEpoke;
  SETMOUSE_CF_DSPTEXT   DDEcond;
} DDE_SETMOUSE_CF_DSPTEXT,  lpDDE_SETMOUSE_CF_DSPTEXT;
```

The following table shows the values in the parameters:

Parameter Name	Meaning	Value	
bLeftButton	Enable or disable interception of the left mouse button	**True** **False**	Enable intercepting the left button Disable intercepting the left button
bRightButton	Enable or disable interception of the right mouse button	**True** **False**	Enable intercepting the right button Disable intercepting the right button
bMiddleButton	Enable or disable interception of the middle mouse button	**True** **False**	Enable intercepting the middle button Disable intercepting the middle button
bSingleClick	Enable or disable interception of the single click	**True** **False**	Enable intercepting the single click Disable intercepting the single click
bDoubleClick	Enable or disable interception of the double click	**True** **False**	Enable intercepting the double click Disable intercepting the double click
bRetrieveString	Retrieve or do not retrieve the pointed string	**True** **False**	Retrieve the pointed string Do not retrieve the pointed string

aMOUSE Identifies Mouse atom as the item.

PC/3270 or PC400 Response

When receiving the Set Mouse Intercept Condition request, PC/3270 or PC400 returns an ACK message if it can set the intercept condition to the specified status. Otherwise, a negative ACK message is returned to the client with one of the following return codes in the low-order byte of the wStatus field:

WM_DDE_ACK(wStatus, aMOUSE)

Return Code	Explanation
2	A character in Condition parameter is not valid.
6	The specified format is not valid.
9	A system error occurred.

Set Presentation Space Service Condition

PC/3270	PC400
X	X

The Set Presentation Space Service Condition function sets the condition for using the following functions:

Get Partial Presentation Space
Find Field
Search for String

The client application sets the condition by calling this function such as:

Start PS position
PS length
EOF flag
Search direction
ASCIIZ string

The client must specify the Set Presentation Space Service Condition function before invoking the functions listed above. The conditions set by this function remain in effect until the next Set Presentation Space Service Condition function is called. The client sends the following message to set the condition:

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             PackDDELParam(WM_DDE_POKE,
              (hData, aEPSCOND) );
```

where:

hData

Identifies a handle to a Windows global memory object containing:

```
typedef struct tagPSSERVCOND
{
  unsigned  char   poke[(sizeof(DDEPOKE)-1)];
   unsigned short uPSStart;                    / PS Position
   unsigned short uPSLength;                    / Length of String or PS
   unsigned short uSearchDir;                  / Direction for search
   unsigned short uEOFflag;                     / EOF effective switch
   unsigned char    szTargetString[1]; /  Target String
} PSSERVCOND;

typedef union tagDDE_PSSERVCOND
{
  DDEPOKE        DDEpoke;
  PSSERVCOND   DDEcond;
} DDE_PSSERVCOND,  lpDDE_PSSERVCOND;
```

The following values are valid at the uSearchDir field:

```
PCS_SRCHFRWD                / Search forward.
PCS_SRCHBKWD      1          / Search backward.
```

```
typedef struct tagSETMOUSE_CF_DSPTEXT
{
   unsigned char poke[(sizeof(DDEPOKE)-1)];
   BOOL          bLeftButton;         /  Enable intercepting left button
   BOOL          bRightButton;        /  Enable intercepting right button
   BOOL          bMiddleButton;       /  Enable intercepting middle button
   BOOL          bSingleClick;        /  Enable intercepting single click
   BOOL          bDoubleClick;        /  Enable intercepting double click
   BOOL          bRetrieveString; /  Enable intercepting retrieve string
} SETMOUSE_CF_DSPTEXT;

typedef union tagDDE_SETMOUSE_CF_DSPTEXT
{
  DDEPOKE                 DDEpoke;
  SETMOUSE_CF_DSPTEXT   DDEcond;
} DDE_SETMOUSE_CF_DSPTEXT,  lpDDE_SETMOUSE_CF_DSPTEXT;
```

The following values are valid at the uEOFflag field:

```
PCS_UNEFFECTEOF          /  The string is not truncated at EOF.
PCS_EFFECTEOF     1      /  The string is truncated at EOF.
```

aEPSCOND

Identifies the item for the Set Presentation Space Service Condition function.

PC/3270 or PC400 Response

If PC/3270 or PC400 can perform the Set Presentation Space Service Condition function, then PC/3270 or PC400 returns an ACK message:

WM_DDE_ACK(wStatus, aEPSCOND)

If PC/3270 or PC400 cannot perform the Set Presentation Space Service Condition function, then PC/3270 or PC400 returns a negative ACK message containing the following return codes in the low-order byte of wStatus:

Return Code	Explanation
1	PS position is not valid.
2	Length is not valid.
3	The value of EOF flag is not valid.
4	The value of Search Direction is not valid.
6	The specified format is not valid.
9	A system error occurred.

Set Session Advise Condition

PC/3270	PC400
X	X

This function sets the condition for the DDE_ADVISE of the Start Session Advise function. The client can specify a search string and a region of the screen. When the advise condition is met, the server notifies the client of the condition according to the options specified by the Start Session Advise function.

Note: The client must specify the Set Session Advise Condition function before invoking Start Session Advise. If the advise condition is set after the Start Session Advise function is started, the advise condition will be ignored and the client will receive a negative ACK message. See “Start Session Advise” on page 259 for more information about starting the advise.

The client sends the following message to set the advise condition.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             PackDDElParam(WM_DDE_POKE,
              (hData, aPSCOND) );
```

where:

hData Identifies a handle to a Windows global memory object containing:

```
typedef struct tagSEARCHDATA
{
  unsigned  char   poke[(sizeof(DDEPOKE)-1)];
   unsigned short uPSStart;                          / PS Position of string
   unsigned short uPSLength;                          / Length of String
   BOOL                bCaseSensitive;                 / Case Sensitive TRUE=YES
   unsigned char    SearchString[1]; /  Search String
} SEARCHDATA;

typedef union tagDDE_SEARCHDATA
{
  DDEPOKE        DDEpoke;
  SEARCHDATA   DDEcond;
} DDE_SEARCHDATA,  lpDDE_SEARCHDATA;
```

aPSCOND

Identifies the item for the Set Session Advise Condition function.

PC/3270 or PC400 Response

If PC/3270 or PC400 can perform the Set Session Advise Condition function, PC/3270 or PC400 returns this ACK message:

WM_DDE_ACK(wStatus, aPSCOND)

If PC/3270 or PC400 cannot perform the Set Session Advise Condition function, then PC/3270 or PC400 returns an negative ACK message containing the following return codes in the low-order byte of wStatus:

Return Code	Explanation
1	Advise is already active.
2	Advise condition is already active.
3	PS position is not valid.
4	String length is not valid.
6	The specified format is not valid.
9	A system error occurred.

Set Structured Field Service Condition

PC/3270	PC400
X	X

The Set Structured Field Service Condition function passes the Query Reply data provided by the client application.

Note: The client must call the Set Structured Field Service Condition function before invoking the Start Read SF function or the Write SF function.

The client sends the following message to set the condition.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             PackDDElParam(WM_DDE_POKE,
              (hData, aSFCOND) );
```

where:

hData Identifies a handle to a Windows global memory object containing:

```
typedef struct tagSFSERVCOND
{
  unsigned  char   poke[(sizeof(DDEPOKE)-1)];
   unsigned short uBufferLength;                    / Buffer size of Read_SF
   unsigned short uQRLength;                        / Length of Query Reply dat
   unsigned char    szQueryReply[1]; /  Query Reply data
} SFSERVCOND;

typedef union tagDDE_SFSERVCOND
{
  DDEPOKE         DDEpoke;
  SFSERVCOND   DDEcond;
} DDE_SFSERVCOND,  lpDDE_SFSERVCOND;
```

aSFCOND

Identifies the item for the Set Structured Field Service Condition function.

PC/3270 response

PC/3270 checks the Query Reply ID and Type (not DOID) and the length. If they are valid, then PC/3270 returns an ACK message:

```
WM_DDE_ACK(wStatus, aSFCOND)
```

If PC/3270 cannot perform the Set Structured Field Service Condition function, then PC/3270 returns a negative ACK message containing the following return codes in the low-order byte of wStatus:

Return Code	Explanation
1	PS SF ID is not valid.
2	Length is not valid.
3	One DDM base type is already connected to this session.
4	Structured Field Service Condition is already set.
6	The specified format is not valid.
9	A system error occurred.

Start Close Intercept

PC/3270	*PC400*
X	X

The Start Close Intercept function allows a client application to intercept close requests generated when a user selects the close option from the emulator session window. This function intercepts the close request and discards it until the Stop Close Intercept function is requested. After using this function, the client receives DATA messages notifying it that close requests occurred (CLOSE).

The client sends the following command to begin a session advise.

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
             Pack  DDElParam(WM_DDE_ADVISE,
              (hOptions, aCLOSE) );
```

where:

hOptions Is a handle to a Windows global memory object DDEADVISE structure.

If the value of fDeferUpd is 1, DDE Data messages will be sent to the client application with the hData set to NULL. The client must then issue a DDE REQUEST to request the data item.

If the value of fAckReq is 1, the server does not notify the client of further close requests until the server receives an ACK message from the client in response to any previous notification.

The cfFormat field specifies the format to send the close request. (Must be CF_DSPTEXT.)

aCLOSE Identified close intercept as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 receives the Start Close Intercept and returns an ACK message if it can start the intercept. Otherwise a negative ACK message will be returned to the client with the following return codes in the low-order byte of the wStatus field:

```
WM_DDE_ACK(wStatus, aCLOSE)
```

Return Code	Explanation
1	Close Intercept is already working.
6	The specified format is not valid.
9	A system error occurred.

Once the intercept starts, the client receives DATA messages notifying it that the close request is intercepted:

WM_DDE_DATA(hData, aCLOSE)

where:

hData Identifies a handle to a Windows global memory object containing:

```
typedef struct tagCLOSEREQ
{
  unsigned  char   data[(sizeof(DDEDATA)-1)];
   unsigned short uCloseReqCount; /  Number of the close requests.
} CLOSEREQ;

typedef union tagDDE_CLOSEREQ
{
  DDEDATA      DDEdata;
  CLOSEREQ   DDEclose;
} DDE_CLOSEREQ,  lpDDE_CLOSEREQ;
```

The DATA messages continue until a Stop Close Intercept message is sent to PC/3270 or PC400.

Start Keystroke Intercept

PC/3270	*PC400*
X	X

The Start Keystroke Intercept function allows a client application to filter any keystrokes sent to a session by a terminal operator. After a call to this function, keystrokes are intercepted, and the client receives them (KEYS).

The client sends the following command to begin intercept.

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
             PackDDElParam(WM_DDE_ADVISE,
              (hOptions, aKEYS) );
```

where:

hOptions Is a handle to a Windows global memory object DDEADVISE structure.

If the value of fDeferUpd is 1, DDE Data messages are sent to the client application with the hData set to NULL. The client then issues a DDE REQUEST to request the data item.

If the value of fAckReq is 1, the server does not notify the client of further keystrokes until the server receives an ACK message from the client in response to any previous keystrokes notification.

The cfFormat field specifies the format to send the keystrokes when the keystroke is sent by a terminal operator. (Must be CF_DSPTEXT.)

aKEYS Identified keystrokes as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 receives the Start Keystroke Intercept and returns an ACK message if it can start the intercept. Otherwise a negative ACK message is returned to the client with the following return codes in the low-order byte of the wStatus field:

```
WM_DDE_ACK(wStatus, aKEYS)
```

Return Code	Explanation
1	Keystroke Intercept is already started.
6	The specified format is not valid.
9	A system error occurred.

Once the intercept has started, the client receives DATA messages notifying it that the keystroke is intercepted:

```
WM_DDE_DATA(hData, aKEYS)
```

The DATA messages continue until a Stop Keystroke Intercept message is sent to PC/3270 or PC400. The format of the data item will be the same format as if the client requested the data item via a DDE_REQUEST.

Start Mouse Input Intercept

PC/3270	*PC400*
X	X

The Start Mouse Input Intercept function allows a client application to intercept mouse input when a terminal operator press the mouse button on emulator session window. After calling this function, the client receives DATA messages that include the PS position where mouse input occurred.

The client sends the following command to begin to intercept the mouse input.

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
             PackDDElParam(WM_DDE_ADVISE,
              (hOptions, aMOUSE) );
```

where:

hOptions Is a handle to a Windows global memory object DDEADVISE structure.

If the value of fDeferUpd is 1, DDE Data messages will be sent to the client application with the hData set to NULL. The client must then issue a DDE REQUEST to request the data item.

If the value of fAckReq is 1, the server does not notify the client of further structured field data until the server receives an ACK message from the client in response to any previous notification.

The cfFormat field specifies the format to send the data item has been updated.

aMOUSE Identified MOUSE as the item.

PC/3270 or PC400 response

PC/3270 or PC400 receives the Start Mouse Input Intercept and returns an ACK message if it can start this function. Otherwise a negative ACK message is returned to the client with the following return codes in the low-order byte of the wStatus field:

WM_DDE_ACK(wStatus, aMOUSE)

Return Code	Explanation
1	Mouse Input Intercept has been already started.
6	The specified format is not valid.
9	A system error occurred.

Once the Mouse Input Intercept starts, the client receives DATA messages of the structured field:

WM_DDE_DATA(hData, aMOUSE)

where:

hData

If the format is CF_TEXT, PC/3270 or PC400 returns the mouse input information in the following format:

```
typedef struct tagMOUSE_CF_TEXT
{
   unsigned char data[(sizeof(DDEDATA)-1)];
   unsigned char PSPos[4];            /  PS Offset - Mouse position
   unsigned char Tab1[1];             /  Tab character
   unsigned char PSRowPos[4];       /  ROW number of Mouse position
   unsigned char Tab2[1];             /  Tab character
   unsigned char PSColPos[4];       /  Col number of Mouse position
   unsigned char Tab3[1];             /  Tab character
   unsigned char PSSize[4];          /  Size od Presentation Space
   unsigned char Tab4[1];             /  Tab character
   unsigned char PSRows[4[;          /  Row number of PS
   unsigned char Tab5[1];             /  Tab character
   unsigned char PSCols[4];          /  Column number of PS
   unsigned char Tab6[1];             /  Tab character
   unsigned char Button[1];          /  Type of clicked mouse butt n
   unsigned char Tab7[1];             /  Tab character
   unsigned char Click[1];           /  Type of clicking
   unsigned char Tab8[1];             /  Tab character
   unsigned char zClickString[1];/  Retrived string
} MOUSE_CF_TEXT;

typedef union tagDDE_MOUSE_CF_TEXT
{
  DDEDATA              DDEdata;
  MOUSE_CF_TEXT   DDEmouse;
} DDE_MOUSE_CF_TEXT,  lpDDE_MOUSE_CF_TEXT;
```

The following table shows the values in the parameters:

Parameter Name	Meaning	Value
PSPos	PS offset of the position where mouse was clicked	0 ... (PSSize – 1)
PSRowPos	Row number of the position where mouse was clicked	0 ... (PSRows – 1)
PSColPos	Column number of the position where mouse was clicked	0 ... (PSCols – 1)
PSSize	Presentation space size	
PSRows	Number of presentation space rows	
PSCols	Number of presentation space columns	
ButtonType	Type of clicked mouse button	**L** Left button **M** Middle button **R** Right button
ClickType	Type of clicking	**S** Single click **D** Double click
ClickString	Retrieved string to which the mouse pointed	A character string terminated with a '\0'
Tab1–8	A tab character for delimiter	'\t'

If the format is CF_DSPTEXT, PC/3270 or PC400 returns the mouse input information in the following format:

```
typedef struct tagMOUSE_CF_DSPTEXT
{
  unsigned  char   data[(sizeof(DDEDATA)-1)];
   unsigned short uPSPos;                          / PS Offset    - Mouse position
   unsigned short uPSRowPos;                       / ROW number - Mouse position
   unsigned short uPSColPos;                       / Col number - Mouse position
   unsigned short uPSSize;                         / Size of Presentation Space
   unsigned short uPSRows;                         / Row number of PS
   unsigned short uPSCols;                         / Column number of PS
   unsigned short uButtonType;                     / Type of clicked mouse button
   unsigned short uClickType;                      / Type of clicking
   unsigned char    zClickString[1]; / Retrived 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 mouse was clicked	0 ... (uPSSize – 1)
uPSRowPos	Row number of the position where mouse was clicked	0 ... (uPSRows – 1)
uPSColPos	Column number of the position where mouse was clicked	0 ... (uPSCols – 1)
uPSSize	Size of the presentation space	
uPSRows	Number of rows of the presentation space	
uPSCols	Number of columns of the presentation space	
uButtonType	Type of the clicked mouse button	**0x0001** Left button **0x0002** Middle button **0x0003** Right button
uClickType	Type of clicking	**0x0001** Single click **0x0002** Double click
szClickString	Retrieved string to which the mouse pointed	A character string terminated with a '\0'

The DATA messages continue until a Stop Mouse Input Intercept message is sent to PC/3270 or PC400.

Start Read SF

PC/3270	PC400
X	X

The Start Read SF function allows a client application to read structured field data from the host application. After using this function, the client receives DATA messages notifying it that close requests occurred.

Note: Before using this function, the client must call the Set Structured Field Service Condition function to pass the Query Reply data to the server.

The client sends the following command to begin a Read SF.

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
             PackDDElParam(WM_DDE_ADVISE,
              (hOptions, aSF) );
```

where:

hOptions Is a handle to a Windows global memory object DDEADVISE structure.

If the value of fDeferUpd is 1, DDE Data messages will be sent to the client application with the hData set to NULL. The client must then issue a DDE REQUEST to request the data item.

If the value of fAckReq is 1, the server does not notify the client of further structured field data until the server receives an ACK message from the client in response to any previous notification.

The cfFormat field specifies the format to send the structured field data. (It must be CF_DSPTEXT.)

aSF Identified structured field as the item.

PC/3270 response

PC/3270 receives the Start Read SF and returns an ACK message if it can start the Read SF. Otherwise a negative ACK message will be returned to the client with the following return codes in the low-order byte of the wStatus field:

WM_DDE_ACK(wStatus, aSF)

Return Code	Explanation
1	Read SF is already started.
3	No prior Set Structured Field Service Condition function was called.
6	The specified format is not valid.
9	A system error occurred.

Once the Read SF has started, the client receives DATA messages of the structured field:

WM_DDE_DATA(hData, aSF)

where:

hData Identifies a handle to a Windows global memory object containing:

```
typedef struct 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]; / Retrived string
} MOUSE_CF_DSPTEXT;

typedef union tagDDE_MOUSE_CF_DSPTEXT
{
  DDEDATA                DDEdata;
  MOUSE_CF_DSPTEXT   DDEmouse;
} DDE_MOUSE_CF_DSPTEXT,  lpDDE_MOUSE_CF_DSPTEXT;
typedef struct tagSFDATA
{
  unsigned  char   data[(sizeof(DDEDATA)-1)];
   unsigned short uSFLength;                       / Length of SF data
   unsigned char    szSFData[1];       / SF data
} SFDATA;

typedef union tagDDE_SFDATA
{
  DDEDATA   DDEdata;
  SFDATA     DDEsfdata;
} DDE_SFDATA,  lpDDE_SFDATA;
```

The DATA messages continue until a Stop Read SF message is sent to PC/3270.

Start Session Advise

PC/3270	PC400
X	X

The Start Session Advise function establishes a link between the PC/3270 or PC400 session and the client. This lets the client receive updates of the presentation space (PS), the operator information area (OIA), or the trim rectangle (TRIMRECT) when the data item is updated.

Note: If the client application needs conditional notification when the presentation space is updated, set an advise condition prior to invoking the advise function for the presentation space. See "Set Session Advise Condition" on page 247.

The client sends the following command to begin a session advise.

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
             PackDDElParam(WM_DDE_ADVISE,
              hOptions, aItem) );
```

where:

hOptions Is a handle to a Windows global memory object DDEADVISE structure. This is the structure:

```
typedef struct tagOPTIONS
{
 unsigned reserved:14;          // Reserved
 unsigned fDeferUpd:1;           // Send notification only
 unsigned fAckReq:1;              // Client will ACK all notices
 WORD        cfFormat;                 // Clipboard format to use
}   OPTIONS, FAR  lpOPTIONS;
```

If the value of fDeferUpd is 1, DDE Data messages are sent to the client application with the hData set to NULL. The client must then issue a DDE REQUEST to request the data item.

If the value of fAckReq is 1, the server does not notify the client of further changes to the data item until the server receives an ACK message from the client in response to any previous update notification.

The cfFormat field specifies the format to send the data item when the item has been updated.

aItem Specifies the item of information being requested; in this case, the value can be PS, OIA, or TRIMRECT.

PC/3270 or PC400 Response

PC/3270 or PC400 receives the Start Session Advise and returns an ACK message if it can start the advise. Otherwise, a negative ACK message will be returned to the client with the following return codes in the low-order byte of the wStatus field:

WM_DDE_ACK(wStatus, aItem)

Return Code	Explanation
1	Advise already active for data item.
6	Advise parameter not valid.
9	A system error occurred.

Once the advise has started, the client receives DATA messages notifying it that the data item (PS, OIA, or TRIMRECT) has changed:

WM_DDE_DATA(hData, aItem)

The DATA messages continue until a Stop Session Advise message is sent to PC/3270 or PC400. The format of the data item will be the same format as if the client requested the data item via a DDE_REQUEST.

Stop Close Intercept

PC/3270	*PC400*
X	X

The Stop Close Intercept function ends a client application's ability to intercept close request. The client sends the following command to perform the Stop Close Intercept function.

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
              MAKELPARAM(NULL, aCLOSE) );
```

where:

aCLOSE Identified close intercept as the item.

PC/3270 or PC400 Response

If PC/3270 or PC400 can perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aCLOSE)
```

If PC/3270 or PC400 cannot perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing negative status information and the following return codes in the low-order byte of the wStatus word:

Return Code	Explanation
1	Advise has not started yet.
9	A system error occurred.

Stop Keystroke Intercept

PC/3270	*PC400*
X	X

The Stop Keystroke Intercept function ends a client application's ability to intercept keystrokes. The client sends the following command to perform the Stop Keystroke Intercept function.

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
              MAKELPARAM(NULL, aKEYS) );
```

where:

aKEYS Identified keystrokes as the item.

PC/3270 or PC400 Response

If PC/3270 or PC400 can perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aKEYS)
```

If PC/3270 or PC400 cannot perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing negative status information and the following return codes in the low-order byte of the wStatus word.

Return Code	Explanation
1	Advise has not started yet.
9	A system error occurred.

Stop Mouse Input Intercept

PC/3270	*PC400*
X	X

The Stop Mouse Input Intercept function ends a client application's ability to intercept mouse input.

The client sends the following command to perform the Stop Mouse Input Intercept function.

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
              MAKELPARAM(NULL, aMOUSE) );
```

where:

aMOUSE Identified MOUSE as the item.

PC/3270 or PC400 Response

If PC/3270 or PC400 can perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aMOUSE)
```

If PC/3270 or PC400 cannot perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing negative status information and the following return codes in the low-order byte of the wStatus word.

Return Code	Explanation
1	Advise has not started yet.
9	A system error occurred.

Stop Read SF

PC/3270	*PC400*
X	X

The Stop Read SF function ends a client application's ability to read structured field data.

The client sends the following command to perform the Stop Read SF function.

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
              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 the following return codes in the low-order byte of the wStatus word.

Return Code	Explanation
1	Advise has not started yet.
9	A system error occurred.

Stop Session Advise

PC/3270	*PC400*
X	X

The Stop Session Advise function disconnects a link between PC/3270 or PC400 and the client. The client sends the following command to perform the Stop Session Advise function.

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
              MAKELPARAM(NULL, aItem) );
```

where:

aItem Specifies the item of information being requested; in this case, the value can be PS, OIA, TRIMRECT, or NULL.

If the value of *aItem* is NULL, then the client has requested termination of all active notifications for the conversation.

PC/3270 or PC400 Response

If PC/3270 or PC400 can perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aItem)
```

If PC/3270 or PC400 cannot perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing negative status information and the following return codes in the low-order byte of the wStatus word.

Return Code	Explanation
1	Advise has not started yet.
9	A system error occurred.

Terminate Session Conversation

PC/3270	*PC400*
X	X

The Terminate Session Conversation function disconnects the client from the PC/3270 or PC400 session the client has previously started a conversation with.

The client sends the following command to terminate a session conversation.

```
SendMessage( hServerWnd,
             WM_DDE_TERMINATE,
             hClientWnd,
               );
```

PC/3270 or PC400 Response

PC/3270 or PC400 acknowledges the terminate command with a terminate message:

WM_DDE_TERMINATE

Terminate Structured Field Conversation

PC/3270	*PC400*
X	X

The Terminate Structured Field Conversation function disconnects the client from a structured field conversation.

The client sends the following command to terminate a structured field conversation.

```
SendMessage( hServerWnd,
             WM_DDE_TERMINATE,
             hClientWnd,
               );
```

PC/3270 response

PC/3270 acknowledges the terminate command with a terminate message:

WM_DDE_TERMINATE

Terminate System Conversation

PC/3270	*PC400*
X	X

This disconnects the client from a system conversation.

The client sends the following command to terminate a system conversation.

```
SendMessage( hServerWnd,
             WM_DDE_TERMINATE,
             hClientWnd,
               );
```

PC/3270 or PC400 Response

PC/3270 or PC400 acknowledges the terminate command with this message:

WM_DDE_TERMINATE

When the user closes a PC/3270 or PC400 session, any global memory blocks that were allocated by PC/3270 or PC400 will be freed by Windows. This can cause problems for the client if the client retains any of these global memory objects for long periods of time. If the client application needs to keep the information in a global memory item for a long period of time, it is suggested that the client make a copy of global memory item into a global memory item the client application owns.

Write SF

PC/3270	PC400
X	X

The Write SF function allows a client application to write structured field data to the host application.

Note: The client must call the Set Structured Field Service Condition function before invoking the Write SF function.

The client sends the following message to write structured field data.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             PackDDELParam(WM_DDE_POKE,
              hData, aSF) );
```

where:

hData Identifies a handle to a Windows global memory object containing:

```
typedef struct tagWRITESF
{
  unsigned  char   poke[(sizeof(DDEPOKE)-1)];
   unsigned short uSFLength;          / Length of SF data
   unsigned char    Work[8];           / Work area
   unsigned char    szSFData[1];       / SF data
} WRITESF;

typedef union tagDDE_WRITESF
{
  DDEPOKE   DDEpoke;
  WRITESF   DDEwritesf;
} DDE_WRITESF,  lpDDE_WRITESF;
```

aSF Identified structured field as the item.

PC/3270 response

PC/3270 receives structured field data and sends them to the host application. If the data transmission completes successfully, then PC/3270 returns an ACK message:

```
WM_DDE_ACK(wStatus, aSF)
```

Otherwise PC/3270 returns an negative ACK message containing the following return codes in the low-order byte of wStatus:

Return Code	Explanation
2	Length is not valid.
6	The specified format is not valid.
9	A system error occurred.

DDE Menu Item API in a 32-Bit Environment

PC/3270 or PC400 supports the addition, deletion,and changing of attributes of a dynamic menu item to the session menu bar. A menu will then be created for this menu item with space for up to 16 submenu items.

PC/3270 or PC400 supports two kinds of DDE conversation. One is PC/3270 or PC400, which acts as a DDE menu client application, and the other is PC/3270 or PC400, which acts as a DDE menu server.

DDE Menu Client in a 32-bit Environment

To add,delete,and change menu items, the following DDE conversation must take place between the session and DDE menu server application.

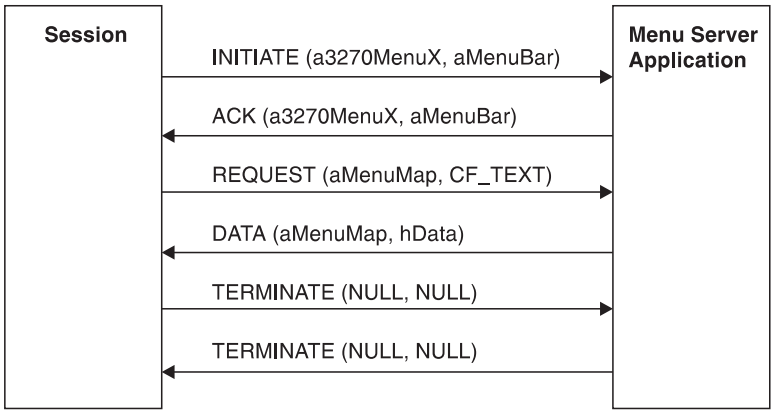

Figure 4. DDE Menu Client Conversation

The following data hierarchy details the menu map PC/3270 or PC400 expects when adding a dynamic menu item and submenu to a session menu bar:

```
POPUP    "MyMenu"
BEGIN
    MENUITEM "Send Files to Host",              SEND
    MENUITEM "Receive Files from Host",     RECEIVE
   MENUITEM  SEPARATOR
    MENUITEM "Convert Files",                   CONVERT
END
```

When the user selects a menu item from the new menu, PC/3270 or PC400 will send a DDE Initiate with 3270MenuN or 5250MenuN as the application and itemN token as the topic. If an ACK is received from the DDE application, PC/3270 or PC400 will inhibit the session from accepting user input. The menu client application can then display a dialog, and so on. When the menu server application has completed processing of the menu item,it will send a DDE Terminate to signal PC/3270 or PC400 the process is complete. PC/3270 or PC400 will then reenable the window for the user.

DDE Menu Server in a 32-bit Environment

To add,delete, and change menu items,the following DDE conversation must take place between the session and a DDE menu client application.

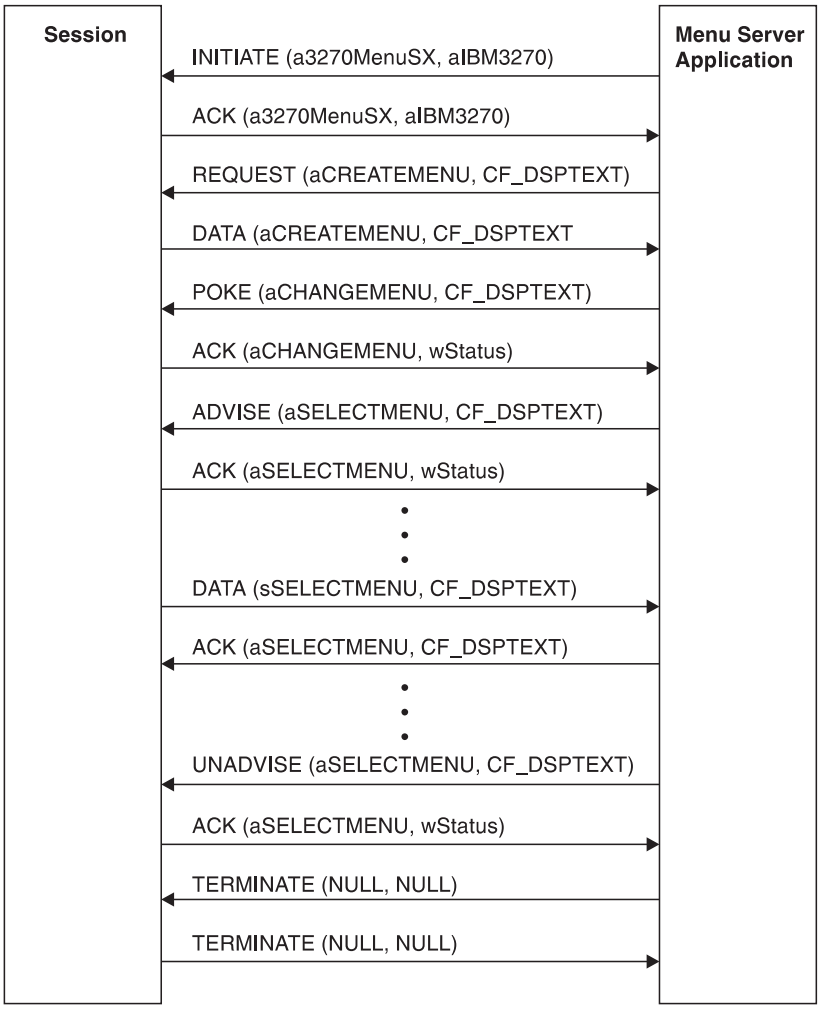

Figure 5. DDE Menu Server Conversation

When the user selects a menu item from the new menu, PC/3270 or PC400 will send a DDE DATA with aSELECTMENU as the item. When PC/3270 or PC400 sends DDE DATA to the client application, PC/3270 or PC400 will inhibit the session from accepting user input. The menu client application can then display a dialog, and so on. When the menu client application has completed processing of the menu item,it will send a DDE ACK to signal PC/3270 or PC400 the process is complete. PC/3270 or PC400 will then reenable the window for the user.

DDE Menu Functions in a 32-bit Environment

The following table lists the DDE Menu Item API functions that are available for use with PC/3270 or PC400 and the page in this section,where each function is more fully documented. PC/3270 and PC400 provide all of the following functions.

Function	Page
Change Menu Item	273
Create Menu Item	280
Initiate Menu Conversation	282
Start Menu Advise	283
Stop Menu Advise	285
Terminate Menu Conversation	286

Change Menu Item

PC/3270	PC400
X	X

The Change Menu Item function appends,deletes,inserts, modifies, and removes menu items. The client sends the following message to the session to change a menu.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             PackDDElParam(WM_DDE_POKE,
             hData,aCHANGEMENU));
```

where:

hData

Identifies a handle to a Windows global memory object that contains the requests for changing a menu. The global memory object contains the following structure:

```
typedef struct tagChangeMenu
{
  unsigned  char   poke[(sizeof(DDEPOKE)-1)];
   HWND              hMenu;                          / Window handle of menu item
   unsigned long    wIDNew;                        / Menu ID of new menu item
   unsigned short wPosition;                     / The position of menu item
   unsigned short wOperation;                   / Specifies the operation
   unsigned short wFlags;                       / Specifies the options
  unsigned  char   szItemName[1];  /  String  of  the  item
} CHANGEMENU;

typedef union tagDDE_CHANGEMENU
{
  DDEPOKE        DDEpoke;
  CHANGEMENU   DDEmenu;
} DDE_CHANGEMENU, lpDDE_CHANGEMENU;
```

The following operations are supported:

```
# MF_APPEND,MF_CHANGE ... MF_BYCOMMANDS are replaced with below commands.
 PCS_INSERT            x       / Inserts a menu item into a menu.
 PCS_CHANGE            x 8     / Modifies a menu item in a menu.
 PCS_APPEND            x 1     / Appends a menu item to the end of a menu
 PCS_DELETE            x 2     / Deletes a menu item from a menu,
                               / destroying the menu item.
 PCS_REMOVE            x1      / Removes a menu item from a menu but
                               / does not destroy the menu item.

 PCS_CHECKED           x 8     / Places a checkmark next to the item.
 PCS_DISABLED          x 2     / Disables the menu item so that it cannot
                               / be selected,but does not gray it.
 PCS_ENABLED           x       / Enables the menu item so that it can be
                               / selected and restores from its grayed
                               / state.
 PCS_GRAYED            x 1     / Disables the menu item so that it cannot
                               / be selected,and grays it.
 PCS_MENUBARBREAK      x 2     / Same as PCS_MENUBREAK except that for
                               / popup menus,separates the new column
                               / from the old column with a vertical line
 PCS_MENUBREAK         x 4     / Plaves the item on a new line for menu
                               / bar items. For popupmenus,places the
                               / item in a new column,with no dividing
                               / line between the columns.
 PCS_SEPARATOR         x 8     / 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          x       / Does not place a checkmark next to the
                               / item (default).

 PCS_BYCOMMAND         x       / 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        x 4     / Specifies that the nPosition parameter
                               / gives the position of the menu item
                               / to be deleted rather than an ID number.
```

If the MF_APPEND is specified in the wOperation field,the following fields must be filled:

hMenu

Identifies the menu to be appended. To append a new item to a pop-up menu, specify the handle that is returned from PC/3270 or PC400 when Create Menu Item function is executed. To append a new item to a top-level menu bar,specify NULL.

wIDNew

Specifies the command ID of the new menu item. If a new item is added to the top-level menu bar,the handle of the menu item returned from PC/3270 or PC400 when Create Menu Item function is executed.

wFlags

The following options can be set:

```
MF_CHECKED          // Places a checkmark next to
                   // the item.
MF_DISABLED         // Disables the menu item so
                   // that it cannot be selected,
                   // but does not gray it.
MF_ENABLED          // Enables the menu item so that
                   // it can be selected and
                   // restores from its grayed
                   //  state.
MF_GRAYED           // Disables the menu item so
                   // that it cannot be selected,
                   // and grays it.
MF_MENUBARBREAK    // Same as MF_MENUBREAK except
                   // that for pop-up menus,
                   // separates the new column from
                   // the old column with a
                   // vertical line.
MF_MENUBREAK        // Places the item on a new line
                   // for menu bar items.
                   // For pop-up menus,places the
                   // item in a new column,with
                   // no dividing line between the
                   //  columns.
MF_SEPARATOR        // Draws a horizontal dividing
                   // line.   Can only be used in a
                   // pop-up menu.   This line cannot
                   // be grayed,disabled,or
                   // highlighted.   The wIDNew and
                   // szItemName fields are
                   //  ignored.
MF_UNCHECKED        // Does not place a checkmark
                   // next to the item (default).
```

szItemName

Specifies the content of the new menu item. Contains a null-terminated character string.

If the MF_CHANGE is specified in the wOperation field, fill these fields:

hMenu

Identifies the menu to be changed. To change an item of a pop-up menu, specify the handle that is returned from PC/3270 or PC400 when Create Menu Item function is executed. To change an item to a top-level menu bar,specify NULL.

nPosition

Specifies the menu item to be changed. The interpretation of the wPosition parameter depends on the setting of the wFlags parameter.

MF_BYPOSITION

Specifies the position of the existing menu item. The first item in the menu is at position zero.

MF_BYCOMMAND

Specifies the command ID of the existing menu item.

wIDNew

Specifies the command ID of the menu item. If an item of the top-level menu bar is changed, the handle of the menu item returned from PC/3270 or PC400 when Create Menu Item function is executed.

wFlags

The following options can be set:

```
MF_BYCOMMAND        // Specifies that the nPosition
                    // parameter gives the menu
                    // item control ID number.
                    // This is the default if
                    // neither MF_BYCOMMAND nor
                    // MF_BYPOSITION is set.
MF_BYPOSITION       // Specifies that the nPosition
                    // parameter gives the position
                    // of the menu item to be
                    // changed rather than an ID
                    //  number.
MF_CHECKED          // Places a checkmark next to
                    // the item.
MF_DISABLED         // Disables the menu item so
                    // that it cannot be selected,
                    // but does not gray it.
MF_ENABLED          // Enables the menu item so
                    // that it can be selected and
                    // restores from its grayed
                    //  state.
MF_GRAYED           // Disables the menu item so
                    // that it cannot be selected,
                    // and grays it.
MF_MENUBARBREAK     // Same as MF_MENUBREAK except
                    // that for pop-up menus,
                    // separates the new column
                    // from the old column with a
                    // vertical line.
MF_MENUBREAK        // Places the item on a new
                    // line for menu bar items.
                    // For pop-up menus,places the
                    // item in a new column,with
                    // no dividing line between
                    // the columns.
MF_SEPARATOR        // Draws a horizontal dividing
                    // line.   Can only be used in
                    // a pop-up menu.   This line
                    // cannot be grayed,disabled,
                    // or highlighted.   The wIDNew
                    // and szItemName fields are
                    //  ignored.
MF_UNCHECKED        // Does not place a checkmark
                    // next to the item (default).
```

szItemName

Specifies the content of the menu item. Contains a null-terminated character string.

If the MF_DELETE is specified in the wOperation field, fill these fields:

hMenu

Identifies the menu to be deleted. To delete an item from a pop-up menu, specify the handle that is returned from PC/3270 or PC400 when Create Menu Item function is executed. To delete an item from a top-level menu bar,specify NULL.

nPosition

Specifies the menu item to be deleted. The interpretation of the nPosition parameter depends on the setting of the wFlags parameter.

MF_BYPOSITION
Specifies the position of the existing menu item. The first item in the menu is at position zero.

MF_BYCOMMAND
Specifies the command ID of the existing menu item.

wFlags

The following options can be set:

```
MF_BYCOMMAND        // Specifies that the nPosition
                    // parameter gives the menu
                    // item control ID number.
                    // This is the default if
                    // neither MF_BYCOMMAND nor
                    // MF_BYPOSITION is set.
MF_BYPOSITION       // Specifies that the nPosition
                    // parameter gives the position
                    // of the menu item to be
                    // deleted rather than an ID
                    //  number.
```

If the MF_INSERT is specified in the wOperation field,the following fields must be filled:

hMenu

Identifies the menu to be inserted. To insert an item to a pop-up menu, specify the handle that is returned from PC/3270 or PC400 when Create Menu Item function is executed. To change an item to a top-level menu bar,specify NULL.

nPosition

Specifies the menu item before the new menu item is to be inserted. The interpretation of the nPosition parameter depends on the setting of the wFlags parameter.

MF_BYPOSITION
Specifies the position of the existing menu item. The first item in the menu is at position zero.

MF_BYCOMMAND
Specifies the command ID of the existing menu item.

wIDNew

Specifies the command ID of the menu item or,if an item of the top-level menu bar is changed, the handle of the menu item returned from PC/3270 or PC400 when Create Menu Item function is executed.

wFlags

The following options can be set:

```
MF_BYCOMMAND        // Specifies that the nPosition
                    // parameter gives the menu
                    // item control ID number. This
                    // is the default if neither
                    // MF_BYCOMMAND nor MF_BYPOSITION
                    // is set.
MF_BYPOSITION       // Specifies that the nPosition
                    // parameter gives the position
                    // of the menu item to be
                    // changed rather than an ID
                    //  number.
MF_CHECKED          // Places a checkmark next to
                    // the item.
MF_DISABLED         // Disables the menu item so
                    // that it cannot be selected,
                    // but does not gray it.
MF_ENABLED          // Enables the menu item so
                    // that it can be selected and
                    // restores from its grayed
                    //  state.
MF_GRAYED           // Disables the menu item so
                    // that it cannot be selected,
                    // and grays it.
MF_MENUBARBREAK     // Same as MF_MENUBREAK except
                    // that for pop-up menus,
                    // separates the new column
                    // from the old column with a
                    // vertical line.
MF_MENUBREAK        // Places the item on a new
                    // line for menu bar items.
                    // For pop-up menus,places the
                    // item in a new column,with
                    // no dividing line between the
                    //  columns.
MF_SEPARATOR        // Draws a horizontal dividing
                    // line.    Can only be used in
                    // a pop-up menu.    This line
                    // cannot be grayed,disabled,
                    // or highlighted. The wIDNew
                    // and szItemName fields are
                    //  ignored.
MF_UNCHECKED        // Does not place a checkmark
                    // next to the item (default).
```

szItemName

Specifies the content of the menu item. Contains a null-terminated character string.

If the MF_REMOVE is specified in the wOperation field,the following fields must be filled:

hMenu

Identifies the menu to be removed. To remove an item from a pop-up menu, specify the handle that is returned from PC/3270 or PC400 when Create Menu Item function is executed. To remove an item from a top-level menu bar,specify NULL.

nPosition

Specifies the menu item to be removed. The interpretation of the nPosition parameter depends upon the setting of the wFlags parameter.

MF_BYPOSITION

Specifies the position of the existing menu item. The first item in the menu is at position zero.

MF_BYCOMMAND

Specifies the command ID of the existing menu item.

wFlags

The following options can be set:

```
MF_BYCOMMAND        // Specifies that the nPosition
                    // parameter gives the menu
                    // item control ID number.
                    // This is the default if
                    // neither MF_BYCOMMAND nor
                    // MF_BYPOSITION is set.
MF_BYPOSITION       // Specifies that the nPosition
                    // parameter gives the
                    // position of the menu item to
                    // be removed rather than an ID
                    //  number.
```

PC/3270 or PC400 Response

PC/3270 or PC400 receives the requests to change a menu and processes them. If the requests cannot be accepted, PC/3270 or PC400 returns a negative ACK message containing the following status codes in the low-order byte of the wStatus word. Otherwise,PC/3270 or PC400 returns a positive ack message signalling that the keystrokes have been sent.

WM_DDE_ACK(wStatus,aCHANGEMENU)

Return code	Explanation
1	The specified parameters are not valid.
6	The specified format is not valid.
9	A system error occurred.

Create Menu Item

PC/3270	*PC400*
X	X

The Create Menu Item function requests PC/3270 or PC400 to add a menu item to the menu bar. A pop-up menu will be created at the same time,but it is initially empty and can be filled with menu items by using this function. The string of the new menu item that will be added to a top-level menu bar,is also specified by using the change menu item function.

The client sends the following message to create a menu item.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELPARAM(cfFormat,aCREATEMENU));
```

where:

cfFormat

Identifies the format for the ID of the new menu item. The valid value is CF_DSPTEXT.

aCREATEMENU

Identifies the create menu item.

PC/3270 or PC400 response

PC/3270 or PC400 returns the handle of the newly created menu item in a dde data message if the PC/3270 or PC400 can create a menu item.

WM_DDE_DATA(hData,aCREATEMENU)

where:

hData

Identifies a handle to a windows global memory object that contains the handle of the menu item. The global memory object contains the following structure: .**

```
typedef struct tagCreateMenu
{
  unsigned  char       data[(sizeof(DDEDATA)-1)];
   HWND                    hMemuItem;        / Handle of the menu item
} CREATEMENU;

typedef union tagDDE_CREATEMENU
{
  DDEDATA        DDEdata;
  CREATEMENU   DDEmenu;
} DDE_CREATEMENU, lpDDE_CREATEMENU;
```

or

WM_DDE_ACK(wStatus,aCREATEMENU)

If PC/3270 or PC400 cannot create a menu item,the following status codes are returned in the low-order byte of the wStatus word:

Return Code	Explanation
6	The specified format is not valid.
9	A system error occurred.

Initiate Menu Conversation

PC/3270	*PC400*
X	X

The Initiate Menu Conversation function connects a client application to an available session of PC/3270 or PC400. Once a menu conversation is established,the session menu is reserved exclusively for the client until the conversation is terminated.

The client application sends the following message to initiate a DDE conversation with a menu:

```
SendMessage( -1,
             WM_DDE_INITIATE,
             hClientWnd,
             MAKELPARAM(aIBM327 32,a327 MenuSN));
```

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.

a3270MenuSN

Identifies the topic atom. The string used to create atom a3270MenuSN is 3270MenuS appended with the session ID A,B,...,Z. In the PC400,the topic atom is a5250MenuSN and the string 5250MenuS appended with the session ID A,B,...,Z. is used to create it.

PC/3270 or PC400 Response

If PC/3270 or PC400 can support a conversation with the client application,PC/3270 or PC400 acknowledges the INITIATE transaction with:

```
WM_DDE_ACK(aIBM327 32,a327 MenuSN)
```

Start Menu Advise

PC/3270	PC400
X	X

The Start Menu Advise function allows a client application to process a user defined routine when the menu item that is added by the client application,is selected. After using this function,the client receives DATA messages indicating which menu item is selected.

The client sends the following command to begin a menu advise.

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
             PackDDElParam(WM_DDE_ADVISE,
             hOptions,aSELECTMENU));
```

where:

hOptions

Is a handle to a Windows global memory object,with the following structure:

```
typedef struct tagOPTIONS
{
 unsigned reserved:14;          // Reserved
 unsigned fDeferUpd:1;          // Send notification only
                                    // (Must be  )
 unsigned fAckReq:1;            // Client will ACK all notices
                                    // (Must be 1)
 WORD       cfFormat;             // Always CF_DSPTEXT
}  OPTIONS,FAR  lpOPTIONS;
```

aSELECTMENU

Identified a menu advise as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 receives the Start Menu Advise and returns an ACK message if it can start the function. Otherwise,a negative ACK message will be returned to the client with the following return codes in the low-order byte of the wStatus field.

Return Code	Explanation
1	Menu Advise has been already started.
6	The specified format is not valid.
9	A system error occurred.

WM_DDE_ACK(wStatus,aSELECTMENU)

Once the menu item (added to the client application) is selected, the client receives DATA messages notifying it which menu item is selected:

WM_DDE_DATA(hData,aSELECTMENU)

where:

hData

Identifies a handle to a Windows global memory object containing:

```
typedef struct tagSELECTMENU
{
  unsigned  char   data[(sizeof(DDEDATA)-1)];
   unsigned short uIDSelected; /  Command ID of the selected menu item
} SELECTMENU;

typedef union tagDDE_SELECTMENU
{
  DDEDATA         DDEdata;
  SELECTMENU   DDEmenu;
} DDE_SELECTMENU, lpDDE_SELECTMENU;
```

The DATA messages continue until a Stop Menu Advise message is sent to PC/3270 or PC400.

Stop Menu Advise

PC/3270	*PC400*
X	X

The Stop Menu Advise function ends a client application's ability to process a user-defined routine when the menu item added by the client application is selected. The client sends the following command to perform the Stop Menu Advise function.

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
             MAKELPARAM(NULL,aSELECTMENU));
```

where:

aSELECTMENU

Identifies a menu advise as the item.

PC/3270 or PC400 Response

If PC/3270 or PC400 can perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus,aCLOSE)
```

If PC/3270 or PC400 cannot perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing negative status information and the following return codes in the low-order byte of the wStatus word:

Return Code	Explanation
1	Advise has not started yet.
9	A system error occurred.

Terminate Menu Conversation

PC/3270	*PC400*
X	X

The Terminate Menu Conversation function disconnects the client from the PC/3270 or PC400 session with which a conversation had been previously started.

The client sends the following command to terminate a session conversation:

```
SendMessage( hServerWnd,
             WM_DDE_TERMINATE,
             hClientWnd,
               );
```

PC/3270 or PC400 Response

PC/3270 or PC400 acknowledges the terminate command with this message:

WM_DDE_TERMINATE

Summary of DDE Functions in a 32-bit Environment

The following table lists the DDE functions that can be used with PC/3270 or PC400. The table lists the name of the DDE function, the command the client sends to PC/3270 or PC400, the values that can be used for the variables in the client command, and the server response.

Table 100 (Page 1 of 5). DDE function summary in a 32-bit environment

function name	client command	server response
Initiate System Conversation(system)	SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELPARAM(aIBM327 32,aSystem));	LOWORD/HIWORD to unpack WM_DDE_ACK(aIBM327 32,aSystem)
Get System Configuration(system)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aSYSCON));	UnPackDDElParam(WM_DDE_DATA,hData,aSYSCON) or UnPackDDElParam(WM_DDE_ACK,wStatus,aSYSCON)
	cfFormat = CF_TEXT	
Get System Formats(system)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aFORMATS));	UnPackDDElParam(WM_DDE_DATA,hData,aFORMATS) or UnPackDDElParam(WM_DDE_ACK,wStatus,aFORMATS)
	cfFormat = CF_TEXT	
Get System Status(system)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aSTATUS));	UnPackDDElParam(WM_DDE_DATA,hData,aSTATUS) or UnPackDDElParam(WM_DDE_ACK,wStatus,aSTATUS)
	cfFormat = CF_TEXT	
Get System SysItems(system)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aSYSITEMS));	UnPackDDElParam(WM_DDE_DATA,hData,aSYSITEMS) or UnPackDDElParam(WM_DDE_ACK,wStatus,aSYSITEMS)
	cfFormat = CF_TEXT	
Get System Topics(system)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aTOPICS));	UnPackDDElParam(WM_DDE_DATA,hData,aTOPICS) or UnPackDDElParam(WM_DDE_ACK,wStatus,aTOPICS)
	cfFormat = CF_TEXT	
Terminate System Conversation(system)	SendMessage(hServerWnd, WM_DDE_TERMINATE, hClientWnd,);	WM_DDE_TERMINATE
Initiate Session Conversation(session)	SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELPARAM(aIBM327 32,aSessionN));	LOWORD/HIWORD to unpack WM_DDE_ACK(aIBM327 32,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_DSPTEXT	
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_DSPTEXT	

DDE Menu Functions in a 32-bit Environment

Table 100 (Page 2 of 5). DDE function summary in a 32-bit environment

function name	client command	server response
Get Mouse Input(session)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aMOUSE)); cfFormat = CF_TEXT | CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aMOUSE) or UnPackDDElParam(WM_DDE_ACK,wStatus,aMOUSE)
Get Number of Close Requests(session)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aCLOSE)); cfFormat = CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aCLOSE) or UnPackDDElParam(WM_DDE_ACK,wStatus,aCLOSE)
Get Operator Information Area(session)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aOIA)); cfFormat = CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aOIA) or UnPackDDElParam(WM_DDE_ACK,wStatus,aOIA)
Get Partial Presentation Space(session)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aEPS)); cfFormat = CF_TEXT | CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aEPS) or UnPackDDElParam(WM_DDE_ACK,wStatus,aEPS)
Get Presentation Space(session)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aPS)); cfFormat = CF_TEXT | CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aPS) or UnPackDDElParam(WM_DDE_ACK,wStatus,aPS)
Get Session Status(session)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aSSTAT)); cfFormat = CF_TEXT	UnPackDDElParam(WM_DDE_DATA,hData,aSSTAT) or UnPackDDElParam(WM_DDE_ACK,wStatus,aSSTAT)
Get Trim Rectangle(session)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aTRIMRECT)); cfFormat = CF_TEXT	UnPackDDElParam(WM_DDE_DATA,hData,aTRIMRECT) or UnPackDDElParam(WM_DDE_ACK,wStatus,aTRIMRECT)
Put Data to Presentation Space(session)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aEPS)); hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aEPS)
Search for String(session)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aSTRING)); cfFormat = CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aSTRING) or UnPackDDElParam(WM_DDE_ACK,wStatus,aSTRING)
Send Keystrokes(session)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aKEYS)); hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aKEYS)
Session Execute Macro(session)	PostMessage(hServerWnd, WM_DDE_EXECUTE, hClientWnd, (LPARAM)hCommands); hCommands = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus, NULL)

Table 100 (Page 3 of 5). DDE function summary in a 32-bit environment

function name	client command	server response
Set Cursor Position(session)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aSETCURSOR));	UnPackDDElParam(WM_DDE_ACK,wStatus,aSETCURSOR)
	hData = Handle to a global memory object	
Set Mouse Intercept Condition(session)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aMOUSE));	UnPackDDElParam(WM_DDE_ACK,wStatus,aMOUSE)
	cfFormat = CF_TEXT | CF_DSPTEXT hData = Handle to a global memory object	
Set Presentation Space Service Condition(session)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aEPSCOND));	UnPackDDElParam(WM_DDE_ACK,wStatus,aEPSCOND)
	hData = Handle to a global memory object	
Set Session Advise Condition(session)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aPSCOND));	UnPackDDElParam(WM_DDE_ACK,wStatus,aPSCOND)
	hData = Handle to a global memory object	
Start Close Intercept(session)	SendMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDElParam(WM_DDE_ADVISE, hOptions,aCLOSE));	UnPackDDElParam(WM_DDE_ACK,wStatus,aCLOSE) or UnPackDDElParam(WM_DDE_DATA,hData,aCLOSE)
	hOptions = Handle to a global memory object	
Start Keystroke Intercept(session)	SendMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDElParam(WM_DDE_ADVISE, hOptions,aKEYS));	UnPackDDElParam(WM_DDE_ACK,wStatus,aKEYS) or UnPackDDElParam(WM_DDE_DATA,hData,aKEYS)
	hOptions = Handle to a global memory object	
Start Mouse Intercept(session)	SendMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDElParam(WM_DDE_ADVISE, hOptions,aMOUSE));	UnPackDDElParam(WM_DDE_ACK,wStatus,aMOUSE) or UnPackDDElParam(WM_DDE_DATA,hData,aMOUSE)
	hOptions = Handle to a global memory object	
Start Session Advise(session)	PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDElParam(WM_DDE_ADVISE, hOptions,aItem));	UnPackDDElParam(WM_DDE_ACK,wStatus,aItem) or UnPackDDElParam(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));	UnPackDDElParam(WM_DDE_ACK,wStatus,aCLOSE)
Stop Keystroke Intercept(session)	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELPARAM(NULL,aKEYS));	UnPackDDElParam(WM_DDE_ACK,wStatus,aKEYS)

DDE Menu Functions in a 32-bit Environment

Table 100 (Page 4 of 5). DDE function summary in a 32-bit environment

function name	client command	server response
Stop Mouse Input Intercept(session)	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELPARAM(NULL,aMOUSE));	UnPackDDElParam(WM_DDE_ACK,wStatus,aMOUSE)
Stop Session Advise(session)	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELPARAM(NULL,aItem));	UnPackDDElParam(WM_DDE_ACK,wStatus,aItem)
	aItem = SysItems | Topics | NULL	
Terminate Session Conversation(session)	SendMessage(hServerWnd, WM_DDE_TERMINATE, hClientWnd,);	WM_DDE_TERMINATE
Initiate Structured Field Conversation(structured field)	SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELPARAM(aIBM327 32,aLUN_xxxx));	LOWORD/HIWORD to unpack WM_DDE_ACK(aIBM327 32,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,);	WM_DDE_TERMINATE
Set Structured Field Service Condition(structured field)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aSFCOND));	UnPackDDElParam(WM_DDE_ACK,wStatus,aSFCOND)
	hData = Handle to a global memory object	
Start Read SF(structured field)	PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDElParam(WM_DDE_ADVISE, hOptions,aSF));	UnPackDDElParam(WM_DDE_ACK,wStatus,aSF) or UnPackDDElParam(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));	UnPackDDElParam(WM_DDE_ACK,wStatus,aSF)
Write SF(structured field)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aSF));	UnPackDDElParam(WM_DDE_ACK,wStatus,aSF)
	hData = Handle to a global memory object	
Initiate Menu Conversation(menu)	SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELPARAM(aIBM327 32,a327 MenuSN));	LOWORD/HIWORD to unpack WM_DDE_ACK(aIBM327 32,a327 MenuSN)
	N = a session letter A through Z	
Change Menu Item(menu)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aCHANGEMENU));	UnPackDDElParam(WM_DDE_ACK,wStatus,aCHANGEMENU)
	hData = Handle to a global memory object	

Table 100 (Page 5 of 5). DDE function summary in a 32-bit environment

function name	client command	server response
Create Menu Item(menu)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aCREATEMENU)); cfFormat = CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aCREATEMENU) or UnPackDDElParam(WM_DDE_ACK,wStatus,aCREATEMENU)
Start Menu Advise(menu)	PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDElParam(WM_DDE_ADVISE, hOption,aSELECTMENU)); hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aSELECTMENU) or UnPackDDElParam(WM_DDE_DATA,hData,aSELECTMENU)
Stop Menu Advise(menu)	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELPARAM(NULL,aSELECTMENU));	UnPackDDElParam(WM_DDE_ACK,wStatus,aCLOSE)
Terminate Menu Conversation(menu)	SendMessage(hServerWnd, WM_DDE_TERMINATE, hClientWnd,);	WM_DDE_TERMINATE

DDE Menu Functions in a 32-bit Environment

Chapter 10. Using DDE Functions with a DDE Client Application

Windows 95 allows users to run multiple Windows application programs and to exchange data between Windows application programs. Dynamic Data Exchange (DDE) allows users this data exchange. Data exchange among Windows application programs can be considered as conversations between *server* and *client* application programs. The client application is an application program that starts DDE, and the server application is an application program that responds to the client application.

The client application needs three names (application program name, topic name, and item name) that are recognized by the server application to start the conversation to exchange data. The client application starts a DDE conversation with the server application by specifying the application program and topic names, and defines the exchange data by specifying the item name.

PC/3270 or PC400 has a function as a DDE server, and can establish DDE conversations with other Windows application programs (Microsoft Visual Basic, Microsoft Excel, Microsoft Word) that have DDE client functions.

Using the PC/3270 and PC400 DDE Interface

To start a DDE conversation and data exchange with PC/3270 or PC400, client application programs need to know the application program name, topic name, and item name that PC/3270 or PC400 can recognize. The exchange data type between an application program and PC/3270 or PC400 is defined with the combination of these names.

Table 101. Naming Scheme for Data Items

Level	Description	Example
Application	A Windows 95 task, a Windows task, or a specific task of the application. In this book, application programs are PC/3270 or PC400.	IBM327032
Topic	A specific part of application programs	SessionA
Item	Type of data passed during DDE conversation	PS (Presentation Space)

Application As a Windows DDE server, PC/3270 or PC400 supports application name *IBM327032* or *IBM525032* for 32-bit applications, *IBM3270* or *IBM5250* for 16-bit applications.

Topic Topic specifies the corresponding topic in the application. The following table shows the topics available to the users:

Table 102 (Page 1 of 2). Topics for PC/3270 or PC400

Topic	Conversation Name	Conversation Type
System	System conversation	Cold link
SessionA, SessionB, ..., SessionZ	Session conversation	Cold link and hot link

© Copyright IBM Corp. 1995, 1996

Table 102 (Page 2 of 2). Topics for PC/3270 or PC400		
Topic	**Conversation Name**	**Conversation Type**
SessA_xxxx, SessB_xxxx, ..., SessZ_xxxx	Session conversation	Hot link

Item The client application programs can exchange data and information with PC/3270 or PC400. Types of data and information are specified by item name.

Explanations for topic conversation procedures and data items to be used follow.

System Conversation

To use the PC/3270 or PC400 system DDE interface, do as follows:

1. Start the system conversation.
2. Request system information.
3. Terminate the system conversation.

Starting the DDE System Conversation with PC/3270 or PC400

To use the DDE interface with PC/3270 or PC400, the client application should first start a DDE conversation with PC/3270 or PC400. To start a DDE conversation, specify *IBM327032* or *IBM525032* as an application name for 32-bit applcations; *IBM3270* or *5250* for 16-bit applications and *System* as a topic name in the DDE function (Initiate) in the client application.

Requesting System Information

After starting the DDE conversation, the client application can request data or information using the DDE function. System information can be requested by specifying the following item names in the DDE function (Request) in the client application:

Item	Return Data	DDE Function
Formats	List of supported Windows clipboard format	Get System Formats
Status	Each session status information	Get System Status
SysCon	Information of emulator support-level and other system-related values	Get System Configuration
SysItems	List of available data items	Get System SysItems
Topics	List of available topics	Get System Topics

Terminating the DDE System Conversation with PC/3270 or PC400

To complete the conversation, the client application needs to terminate the DDE conversation with PC/3270 or PC400. To terminate the conversation, use the DDE function (Terminate) in the client application.

Session Conversation

To use the PC/3270 or PC400 session DDE interface, do as follows:

1. Start the session conversation.
2. Use DDE functions (Request, Poke, Execute).
3. Terminate the session conversation.

Starting the DDE Session Conversation

To use the DDE interface with PC/3270 or PC400 sessions, the client application should start the DDE conversation with PC/3270 or PC400. To start DDE conversation, specify *IBM327032* or *IBM525032* as an application name for 32-bit applications; *IBM3270* or *IBM5250* for 16-bit applications and *SessionA*, *SessionB*, ..., *SessionZ* as topic names in the DDE function (Initiate) in the client application.

Requesting Data

After starting the DDE conversation, the client application can request data using the DDE function. Session information can be requested by specifying the following item names in the DDE function (Request) in the client application:

Item	Return Data	DDE Function
EPS(pos,len,bEOF)	All or a part of session presentation space	Get Partial Presentation Space
FIELD(pos,"type")	Field information	Find Field
OIA	Operator Information Area (OIA) status line information	Get Operator Information Area
PS	Session presentation space	Get Presentation Space
SSTAT	Session status information	Get System Status
STRING(pos,bDir,"string")	String offset start	Search for string
TRIMRECT *1	Session presentation space of trim rectangle	Get Trim Rectangle
Note:		
***1** Parameter can be added.		

Sending Data to the Emulator Window (Poke)

After starting the DDE conversation, the client application can send data to PC/3270 or PC400 sessions using the DDE functions. The following table shows the valid items for the DDE functions:

Item	Explanation	DDE Function
EPS(pos,bEOF)	Sends an ASCII data string to the host presentation space	Put Data to Presentation Space
SETCURSOR	Sets the cursor position	Set Cursor Position

Executing Commands

After starting a DDE conversation, the client application can send commands to the PC/3270 or PC400 session window using the DDE functions. Specify the command in the DDE function (Execute) of the client application. Refer to “Session Execute Macro” on page 396 for details.

Terminating the DDE Session Conversation

The client application should terminate the DDE conversation with PC/3270 or PC400 when completing the task. To terminate the conversation, use the DDE function (Terminate) in the client application.

Session Conversation (Hot Link)

To use the PC/3270 or PC400 session DDE interface, do as follows:

1. Start the session conversation.
2. Start the advice function.
3. Stop the advice function.
4. Terminate the session conversation.

Starting the DDE Session Conversation (Hot Link)

To use the DDE interface with PC/3270 or PC400 sessions, the client application should start the DDE conversation with PC/3270 or PC400. To start the DDE conversation, specify *IBM327032* or *IBM525032* as an application name for 32-bit applications; *IBM3270* or *IBM5250* for 16-bit applications, and *SessionA*, *SessionB*, ..., *SessionZ* as topic names in the DDE function (Initiate) in the client application.

Starting the Hot Link with the Session Window

After starting the DDE conversation, the client application can start the Advise function. Specify the following item names in the DDE function (Advise) in the client application to start the hot link, which enables the automatic data update:

Item	Explanation	DDE Function
CLOSE	Starts to intercept Window Close requests	Start Close Intercept
KEYS	Starts to intercept keystrokes	Start Keystroke Intercept
PS *1 OIA TRIMRECT *1	Start to retrieve data of PS, OIA, or trim rectangle	Start Session Advise
***1:** 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 *1 OIA TRIMRECT *1	Stops to Advise function for the session	Stop Session Advise
***1:** Use the same parameter that is used when Start Session Advise was called.		

Terminating the DDE Session Conversation

The client application should terminate the DDE conversation with PC/3270 or PC400 when completing the task. To terminate the conversation, use the DDE function (Terminate) in the client application.

PC/3270 and PC400 DDE Interface

This section describes the DDE functions that can be used from the other applications, such as Microsoft Excel, Microsoft Word, and Microsoft Visual Basic.

DDE functions for system conversation

Function	Page
Initiate System Conversation	301
Get System Configuration	299
Get System Formats	299
Get System Status	299
Get System SysItems	300
Get System Topics	300
Terminate System Conversation	301

DDE functions for session conversation

Function	Page
Initiate Session Conversation *1	301
Find Field	302
Get Operator Information Area	303
Get Partial Presentation Space	304
Get Presentation Space	305
Get Session Status	305
Get Trim Rectangle	306
Put Data to Presentation Space	307
Search for String	308
Session Execute Macro	308
Set Cursor Position	309
Terminate Session Conversation *2	314

DDE functions for session conversation (hot link)

Function	Page
Initiate Session Conversation (same as *1)	310
Start Close Intercept	310
Start Keystroke Intercept	311
Start Session Advise	311
Stop Close Intercept	313
Stop Keystroke Intercept	313
Stop Session Advise	313
Terminate Session Conversation (same as *2)	314

DDE Functions for System Conversation

The following DDE functions are provided for PC/3270 or PC400 system conversation.

Get System Configuration

The Get System Configuration function returns PC/3270 or PC400 support-level and other system-related values.

DDE parameter Value
Item SysCon

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the topic name (System).

PC/3270 or PC400 Response

The PC/3270 or PC400 system returns the PC/3270 or PC400 system configuration data item.

Returned Information: Refer to "Get System Configuration" on page 383 for details.

The possible reasons that cause PC/3270 or PC400 not to return the system configuration data item are as follows:

An incorrect item name was specified.
A system error has occurred.

Get System Formats

The Get System Formats function returns a list of Windows Clipboard formats that are supported by PC/3270 or PC400.

DDE parameter Value
Item Formats

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the item name (Formats).

PC/3270 or PC400 Response

PC/3270 or PC400 returns a list of supported Windows Clipboard formats.

The possible reasons that cause PC/3270 or PC400 not to return the format data item are as follows:

An incorrect item name was specified.
A system error has occurred.

Get System Status

The Get System Status function returns the status of each configured PC/3270 or PC400 session.

DDE parameter Value
Item SysCon

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the item name (SysCon).

PC/3270 or PC400 Response

PC/3270 or PC400 returns a series of status information to each open session.

Returned Information: Refer to “Get System Status” on page 386 for details.

The possible reasons that cause PC/3270 or PC400 not to return the status data item are as follows:

An incorrect item name was specified.
A system error has occurred.

Get System SysItems

The Get System SysItems function returns a list of data items that can be used with the PC/3270 or PC400 system topic.

DDE parameter Value

Item	SysItems

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the item name (SysItems).

PC/3270 or PC400 Response

PC/3270 or PC400 returns a list of PC/3270 or PC400 system topic data items. The following data items are supported by PC/3270 or PC400:

SysItems
Topics
Status
Formats
SysCon

The possible reasons that cause PC/3270 or PC400 not to return the system data item are as follows:

An incorrect item name was specified.
A system error has occurred.

Get System Topics

The Get System Topics function returns a list of active DDE topics that are supported by PC/3270 or PC400.

DDE parameter Value

Item	Topics

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the item name (Topics).

PC/3270 or PC400 Response

The following topics are supported by PC/3270 or PC400:

System
SessionA, SessionB, ..., SessionZ

The possible reasons that cause PC/3270 or PC400 not to return the system data item are as follows:

An incorrect item name was specified.
A system error has occurred.

Initiate System Conversation

The Initiate System Conversation function starts the system conversation. Only one client application can be connected to one system.

DDE parameter	Value
Topic	System

The client application should start DDE conversation using the DDE function (Initiate) with the PC/3270 or PC400 application name (IBM327032 or IBM525032 for 32-bit applications) or (IBM3270 or IBM5250 for 16-bit applications) and the topic name (System).

Terminate System Conversation

The Terminate System Conversation function terminates the system conversation. Use the DDE function (Terminate) to terminate the DDE conversation from the client application.

DDE Functions for Session Conversation

The following DDE functions are provided for PC/3270 or PC400 session conversation.

Find Field

The Find Field function passes the field information to the client application.

DDE parameter Value
Item FIELD (pos, "type")

Parameter	Value	Explanation
pos	NNNN	PS position
"type"	" " or "T " "P " "N " "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 symbol represents a required blank.

An item in the IBM Personal Communications Version 3.1 format is also supported.

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the foregoing item name.

PC/3270 or PC400 Response

The following table shows the field information that PC/3270 returns:

Field	Returned Information	Explanation
Formatted/Unformatted	Formatted, Unformatted	Whether the presentation space is formatted or unformatted. If Unformatted is specified, no other field information will be returned.
Unprotected/Protected	N	0=Unprotected data field 1=Protected data field
A/N	N	0=Alphanumeric 1=Numeric
I/SPD	N	0=Normal intensity, undetectable 1=Normal intensity, detectable 2=High intensity, detectable 3=Nondisplay, undetectable
MDT	N	0=Field is not changed 1=Field is changed
Field start offset	NNNN	Field starts this field position
Field length	NNNN	Field length

The following table shows the field information that PC400 returns:

Field	Returned Information	Explanation
Formatted/Unformatted	Formatted, Unformatted	Whether the presentation space is formatted or unformatted. If Unformatted is specified, no other field information will be returned.
Field attribute	N	0 = Not field attribute byte 1 = Field attribute byte
Visibility	N	0 = Nondisplay 1 = Display
Unprotected/Protected	N	0 = Unprotected data field 1 = Protected data field
Intensity	N	0 = Normal 1 = High
Field Type	N	0 = Alphanumeric: all characters allowed. 1 = Alphabet only: uppercase and lowercase letters, comma, period, hyphen, blank, and Dup key allowed. 2 = Numeric shift: automatic shift for numerics. 3 = Numeric only: numbers 0–9, comma, period, plus, minus, blank, and Dup key allowed. 5 = Digits only: numbers 0–9 and Dup key allowed. 6 = Magnetic stripe reader data only 7 = Sighed numeric: numbers 0–9, plus, minus, and Dup key allowed.
MDT	N	0=Field is not changed 1=Field is changed
Field start offset	NNNN	Field starts this field position
Field length	NNNN	Field length

The possible reasons that cause PC/3270 or PC400 not to return the field information are as follows:

An incorrect item name was specified.
A system error has occurred.

Get Operator Information Area

The Get Operator Information Area function returns the OIA data information to the client application.

DDE parameter Value
Item OIA

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the item name (OIA).

PC/3270 or PC400 Response

The following table shows the OIA information that PC/3270 or PC400 returns:

Offset	Returned Information	Meaning
0	ONLINE LU-LU SSCP-LU	Online, the screen is unowned LU-LU session owns the screen SSCP-LU session owns the screen

Offset	Returned Information	Meaning
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

The possible reasons that cause PC/3270 or PC400 not to return the OIA information are as follows:

An incorrect item name was specified.
A system error has occurred.

Get Partial Presentation Space

The Get Partial Presentation Space function returns whole or partial presentation space data to the client application.

DDE parameter Value
Item EPS (pos, len, bEOF)

Parameter	Value	Explanation
pos	NNNN	PS position
len	NNNN	PS length
bEOF	1 or 0	EOF switch **1** Yes **0** No

Note: An item in the IBM Personal Communications Version 3.1 format is also supported.

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the foregoing item names.

PC/3270 or PC400 Response

The following table shows the information PC/3270 or PC400 returns:

Field	Returned Information	Explanation
PS start position	NNNN	Specified by pos parameter
PS length	NNNN	Specified by len parameter
PS rows	NNNN	
PS columns	NNNN	
PS		PS data starts from this position

The possible reasons that cause PC/3270 or PC400 not to return the format data items are as follows:

An incorrect item name was specified.
A system error has occurred.

Get Presentation Space

The Get Presentation Space function returns presentation space data to the client application.

DDE parameter Value
Item PS

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the item name (PS).

PC/3270 or PC400 Response

The following table shows the information PC/3270 or PC400 returns:

Field	Returned Information	Explanation
PS size	NNNN	
PS rows	NNNN	
PS columns	NNNN	
PS		PS data starts from this position

The possible reasons that cause PC/3270 or PC400 not to return the format data items are as follows:

An incorrect item name was specified.
A system error has occurred.

Get Session Status

The Get Session Status function returns the connected session status to the client application.

DDE parameter Value
Item SSTAT

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the item name (SSTAT).

PC/3270 or PC400 Response

Refer to “Get Session Status” on page 381 for the returned information.

The possible reasons that cause PC/3270 or PC400 not to return the format data items are as follows:

An incorrect item name was specified.
A system error has occurred.

Get Trim Rectangle

The Get Trim Rectangle function returns the presentation space area of the current (or specified) trim rectangle to the client application.

DDE parameter Value

Item TRIMRECT (row1, col1, row2, col2)
TRIMRECT (pos1, pos2)
TRIMRECT

Parameter	Value	Explanation
row1	NN	Top-left corner row of the trim rectangle
col1	NN	Top-left corner column of the trim rectangle
row2	NN	Bottom-right corner row of the trim rectangle
col2	NN	Bottom-right corner column of the trim rectangle
pos1	NNNN	PS position of the top-left corner of the trim rectangle
pos2	NNNN	PS position of the bottom-right corner of the trim rectangle

Note: An item in the IBM Personal Communications Version 3.1 format is also supported.

The current specified PS trim rectangle is used unless the client application specifies the PS trim rectangle in the parameter.

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the item name (TRIMRECT).

PC/3270 or PC400 Response

The information returned from PC/3270 or PC400 is as follows:

Field	Returned Information	Explanation
PS		PS data starts from this position.

The possible reasons that cause PC/3270 or PC400 not to return the trim rectangle items are as follows:

An incorrect item name was specified.
A system error has occurred.

Initiate Session Conversation

The Initiate Session Conversation function starts a DDE conversation in the available session window. Only one client application can be connected to one session conversation:

DDE parameter Value
Topic SessionA, SessionB, ..., SessionZ

Parameter Value	Explanation
SessionA, SessionB, ..., SessionZ	"SessionA" implies a string combined "Session" and a session ID “A,” “B,” ..., “Z.”

The client application should start the DDE conversation by specifying the DDE function (Initiate) of the client application with the topic name (SessionA, SessionB, ..., SessionZ).

PC/3270 or PC400 Response

If a topic is not specified, PC/3270 or PC400 responds after confirming the following available topics:

System
SessionA, SessionB, ..., SessionZ

Put Data to Presentation Space

The Put Data to Presentation Space function sends an ASCII data string to write on the specified host presentation space.

DDE parameter Value
Item EPS (pos, bEOF)

Parameter	Value	Explanation
pos	NNNN	PS position to start writing the data
bEOF	1 or 0	EOF switch **1** Yes **0** No

Note: An item in the IBM Personal Communications Version 3.1 format is also supported.

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Poke) of the client application with the foregoing item name.

PC/3270 or PC400 Response

The possible reasons that cause PC/3270 or PC400 not to accept the string data are as follows:

An incorrect item name was specified.
The PS position is not valid.
The length is not valid.
The PS input was inhibited.
A system error has occurred.

Search for String

Using the Search for String function, the client application can check whether the specified strings exist within the specified presentation space area.

DDE parameter Value
Item STRING (pos, bDir, "string")

Parameter	Value	Explanation
pos	NNNN	PS start position of the string search
bDir	1 or 0	Search Direction **1** Forward **0** Backward
"string"		Search string Enclose a string including blanks with double quotation marks. To specify a double quotation mark within the string, enclose the double quotation mark with another set of double quotation marks. Example: *This is a double quotation" mark.* is specified as *"This is a double quotation"" mark."*.

Note: An item in the IBM Personal Communications Version 3.1 format is also supported.

The maximum length of the search string is 255.

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the foregoing item names.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the following information:

Field	Returned Information	Explanation
String start offset	NNNN, None	"None" is returned if the string is not found

The possible reasons that cause PC/3270 or PC400 not to return the string start position are as follows:

An incorrect item name was specified.
The PS position is not valid, or the string is too long.
A system error has occurred.

Session Execute Macro

The Session Execute Macro function enables users to send commands and macro strings to PC/3270 or PC400.

Refer to “Session Execute Macro” on page 396 for details of commands and macro strings.

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Execute) of the client application.

PC/3270 or PC400 Response

The possible reasons that cause PC/3270 or PC400 not to return the string start position is as follows:

A system error has occurred.

Set Cursor Position

Using the Set Cursor Position function, the client application can set the cursor position in the session window.

DDE parameter	Value
Item	SETCURSOR
Data (Cursor position)	NNNN or Rn1Rn2

Parameter Value	Explanation
NNNN	PS offset
Rn1Rn2	Row/column **n1** PS position row **n2** PS position column

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Poke) of the client application with the foregoing item names.

PC/3270 or PC400 Response

The possible reasons that cause PC/3270 or PC400 not to move the cursor to the specified PS position are as follows:

An incorrect item name was specified.
The Cursor PS offset is not valid (it must be from 0 to PS size−1).
The Cursor row value is not valid (it must be from 0 to PS row−1).
The Cursor column value is not valid (it must be from 0 to PS column−1).
A system error has occurred.

Terminate Session Conversation

The Terminate Session Conversation function terminates the DDE conversation between the client application and PC/3270 or PC400.

Use the DDE function (Terminate) of the client application to terminate the DDE conversation.

DDE Functions for Session Conversation (Hot Link)

The following DDE functions are provided for PC/3270 or PC400 session conversation with hot link connection.

Initiate Session Conversation

The Initiate Session Conversation function starts a DDE conversation with the available session window.

DDE parameter Value

Topic SessionA, SessionB, ..., SessionZ or SessA_xxxx, SessB_xxxx, ..., SessZ_xxxx

Note: If SessA_xxxx, SessB_xxxx, ..., SessZ_xxxx is used, the client application allows only hot link session conversation.

Parameter Value	Explanation
SessA_xxxx, SessB_xxxx, ..., SessZ_xxxx	String 'SessA_xxxx' indicates session A (SessA_) with any user-defined strings (xxxx). The length of the user-defined strings is not limited.

Specify the PC/3270 or PC400 application name and the foregoing topic name in the DDE function (Initiate) of the client application to start a DDE conversation.

Start Close Intercept

Using the Start Close Intercept function, the client application can intercept the Close request generated by selecting the Close option from the emulator session window. When this service is started, the client application receives the Close request event data.

DDE parameter Value

Item CLOSE

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Advise) of the client application with the foregoing item name.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the following information:

Field	Returned Information	Explanation
Number of PS close request	NNNN	When a Close request is generated, the client application receives "0001".

The possible reasons that cause PC/3270 or PC400 not to start to Close intercept are as follows:

An incorrect item name was specified.

The Close intercept for the session has already started with the same topic name.

A system error has occurred.

Start Keystroke Intercept

Using the Start Keystroke Intercept function, the client application can filter keystrokes that are entered by the terminal operator. When started, the keystrokes are intercepted and received by the client application.

DDE parameter Value
Item KEYS

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Advise) of the client application with the foregoing item name.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the following information:

Field	Returned Information	Explanation
Keys		Refer to Table 99 “SENDKEY Command List” of Chapter 9, “DDE Functions in a 32-Bit Environment” “Session Execute Macro” on page 237.

The possible reasons that cause PC/3270 or PC400 not to start Keystroke Intercept are as follows:

An incorrect item name was specified.

The Keystroke Intercept for the session has already started with the same topic name.

A system error has occurred.

Start Session Advise

The Start Session Advise function establishes the link between the client application and PC/3270 or PC400. As the data item is changed, the client application receives the changed data of the presentation space (PS), Operator Information Area (OIA), or trim rectangle (TRIMRECT).

DDE parameter Value
Item PS (pos, len, bCaseSen, "string")
PS
TRIMRECT (row1, col1, row2, col2)
TRIMRECT (pos1, pos2)
TRIMRECT
OIA

The maximum length of the search string is 255.

Parameter	Value	Explanation
pos	NNNN	PS start position of the string search (PS offset)
len	NNNN	Length of the search string
bCaseSen	1 or 0	Case sensitivity **1** Yes **0** No

Parameter	Value	Explanation
"string"		Search string Enclose a string including blanks with double quotation marks. To specify a double quotation mark within the string, enclose the double quotation mark with another set of double quotation marks. Example: *This is a double quotation" mark.* is specified as *"This is a double quotation"" mark."*.

Note: An item in the IBM Personal Communications Version 3.1 format is also supported.

To receive a conditional advice when the presentation space is updated, the client application needs to set the advise conditions as well as the foregoing parameter values. The foregoing parameter values can be used when the presentation space is specified as the item name.

Parameter	Value	Explanation
row1	NN	Top-left corner row of the trim rectangle
col1	NN	Top-left corner column of the trim rectangle
row2	NN	Bottom-right corner row of the trim rectangle
col2	NN	Bottom-right corner column of the trim rectangle
pos1	NNNN	PS position of the top-left corner of the trim rectangle
pos2	NNNN	PS position of the bottom-right corner of the trim rectangle

Note: An item in the IBM Personal Communications Version 3.1 format is also supported.

The current specified presentation space trim rectangle is used unless the client application specifies the presentation space trim rectangle in the item name parameter. This parameter value can be used when TRIMRECT is specified as the item name.

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Advise) of the client application with the foregoing item name.

PC/3270 or PC400 Response

Refer to “Get Partial Presentation Space” on page 304, “Get Operator Information Area” on page 303, and “Get Trim Rectangle” on page 306.

The possible reasons that cause PC/3270 or PC400 not to start Advise are as follows:

An incorrect item name was specified.
The Advise for the session has already started with the same topic name
A system error has occurred.

Stop Close Intercept

Using the Stop Close Intercept function, the client application stops intercepting the close requests.

DDE parameter Value

Item	CLOSE

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Unadvise) of the client application with the foregoing item name.

PC/3270 or PC400 Response

The possible reasons that cause PC/3270 or PC400 not to stop close intercept are as follows:

The Advise has not been started.
A system error has occurred.

Stop Keystroke Intercept

Using the Stop Close Intercept function, the client application stops intercepting the keystrokes.

DDE parameter Value

Item	KEYS

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Unadvise) of the client application with the foregoing item name.

PC/3270 or PC400 Response

The possible reasons that cause PC/3270 or PC400 not to stop keystroke intercept are as follows:

An incorrect item name was specified.
The Advise has not been started.
A system error has occurred.

Stop Session Advise

The Stop Session Advise function closes the link between the client application and PC/3270 or PC400.

DDE parameter Value

Item	PS (pos, len, bCaseSen, "string") PS TRIMRECT (row1, col1, row2, col2) TRIMRECT (pos1, pos2) TRIMRECT OIA

The maximum length of the search string is 255.

The item name must be the same item name that was used when Start Session Advise was called.

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Unadvise) of the client application with the foregoing item name.

PC/3270 or PC400 Response

The possible reasons that cause PC/3270 or PC400 not to stop Advise are as follows:

An incorrect item name was specified.
The Advise has not been started
A system error has occurred.

Terminate Session Conversation

The Terminate Session Conversation function terminates the DDE conversation between the client application and the PC/3270 or PC400 session.

Use the DDE function (Terminate) of the client application to terminate the DDE conversation.

Visual Basic Sample Program

Following is a sample program with Visual Basic:

Note: This sample program is simplified and differs from the actual sample file provided.

```
'/                                  /
'/                                                  /
'/        System  conversation                    /
'/                                                  /
'/                                  /

'
'
'         Initiate System Conversation
'
'
'
'   Start DDE Conversation with system
'
Sub Command1_Click ()
On Error GoTo ErrHandlerInit
     Dim COLD As Integer
     COLD = 2
     FunctionComp& = True

    DoEvents
     Text1.LinkTopic = "IBM327 32|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. , use "DoEvents"
' function before starting DDE conversation by
' calling "LinkMode" function.
'
'-----------------------------------------------------
'
'
'         Get System Format
'
'
'
'     Request a list of PC/327  or PC4
'    Clipboard  Format
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "Formats"
    Text1.LinkRequest

     If FunctionComp&= False Then
          MsgBox "Error has occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandler:
     FunctionComp& = False
```

```
    Resume  Next
End Sub

'
'
'          Get System Status
'
'
'
'     Requests each PC/327  or PC4
'     Session  Status
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "Status"
    Text1.LinkRequest

     If FunctionComp&= False Then
          MsgBox "Error has occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandler:
     FunctionComp& = False
    Resume  Next
End Sub

'
'
'          Get System Configuration
'
'
'
'     Requests PC/327  or PC4
'     System Configuration Values
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "SysCon"
    Text1.LinkRequest

     If FunctionComp&= False Then
          MsgBox "Error has occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandler:
     FunctionComp& = False
    Resume  Next
End Sub

'
'
'          Get System SysItems
'
'
'
'     Requests a list of Data Items for
'     PC/327  or PC4   System Conversation
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "SysItems"
    Text1.LinkRequest

     If FunctionComp&= False Then
```

```
        MsgBox "Error has occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandler:
     FunctionComp& = False
    Resume  Next
End Sub

'
'
'        Get System Topics
'
'
'
'     Requests a list of PC/327  or PC4
'    Topics
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "Topics"
    Text1.LinkRequest

     If FunctionComp&= False Then
        MsgBox "Error has occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandler:
     FunctionComp& = False
    Resume  Next
End Sub

'
'
'        Terminate System Conversation
'
'
'
'   Terminates DDE Conversation with system
'
Sub Command3_Click ()
On Error GoTo ErrHandler
     Dim NONE As IntegerTerm
     NONE =
     FunctionComp& = True

     Text1.LinkMode = NONE

     If FunctionComp&= False Then
        MsgBox "Error has occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandlerTerm:
     FunctionComp& = False
    Resume  Next
End Sub

'/                                  /
'/                                                  /
'/        Session  conversation                  /
'/                                                  /
'/                                  /

'
'
'        Initiate Session Conversation
'
'
```

```
'
'    Initiate DDE Conversation with system
'
Sub Command1_Click ()
On Error GoTo ErrHandlerInit
     Dim COLD As Integer
     COLD = 2
     FunctionComp& = True

    DoEvents
     Text1.LinkTopic = "IBM327 32|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. , use "DoEvents"
' function before starting DDE conversation by
' calling "LinkMode" function.
'
'-----------------------------------------------------
'
'
'         Find  Field
'
'
'
'     Requests 1   Field Information of PS Position
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "FILED(1  ,""     "")"
    Text1.LinkRequest

     If FunctionComp&= False Then
          MsgBox "Error 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 1   to 1
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "EPS(1  ,1   ,1)"
    Text1.LinkRequest

     If FunctionComp&= False Then
          MsgBox "Error 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 2
'
Sub Command2_Click ()
On Error GoTo ErrHandler
    FunctionComp& = True

    Text1.Text         = "Hello, World!"
    Text1.LinkItem = "EPS(2  ,1)"
    Text1.LinkPoke

    If FunctionComp&= False Then
        MsgBox "Error has occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandler:
    FunctionComp& = False
    Resume  Next
End Sub

'
'
'         Search for String
'
'
'
'     Search forward for string "Hello!" from PS Position 1
'
Sub Command2_Click ()
On Error GoTo ErrHandler
```

```
    FunctionComp& = True

   Text1.LinkItem    =  "STRING(1,1,""Hello!"")"
   Text1.LinkRequest

    If FunctionComp&= False Then
        MsgBox "Error has occurred", 48, "DDE sample"
   End  If
   Exit  Sub

ErrHandler:
    FunctionComp& = False
   Resume  Next
End Sub

'
'
'         Session Execute Macro
'
'
'
'     Maximize the Session
'
Sub Command2_Click ()
On Error GoTo ErrHandler
    FunctionComp& = True

   Text1.LinkExecute  "[WINDOW(MAXIMIZE)]"

    If FunctionComp&= False Then
        MsgBox "Error has occurred", 48, "DDE sample"
   End  If
   Exit  Sub

ErrHandler:
    FunctionComp& = False
   Resume  Next
End Sub

'
'
'         Set Cursor Position
'
'
'
'     Set Cursor Position (Row,Column) = (1,1)
'
Sub Command2_Click ()
On Error GoTo ErrHandler
    FunctionComp& = True

   Text1.Text         =  "R1C1"
    Text1.LinkItem = "SETCURSOR"
   Text1.LinkPoke

    If FunctionComp&= False Then
        MsgBox "Error has occurred", 48, "DDE sample"
   End  If
   Exit  Sub

ErrHandler:
    FunctionComp& = False
   Resume  Next
End Sub

'
'
'         Terminate Session Conversation
'
'
'
'   Terminate DDE Conversation with session
```

```
'
Sub Command3_Click ()
On Error GoTo ErrHandlerTerm
     Dim NONE As Integer
     NONE =
     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 = "IBM327 32|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. , 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 = "IBM327 32|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. , 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 = "IBM327 32|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. , use "DoEvents"
' function before starting DDE conversation by
' calling "LinkMode" function.
'
'----------------------------------------------------
'
'
'          Start Session Advise(TRIMRECT)
'
'
'
'    Receives PS Data in Trim Rectangle when PS Data in Trim Rectangle
'    specified by R1C1:R2 C4  is changed
'
```

```
Sub Command1_Click ()
On Error GoTo ErrHandlerInit
     Dim HOT As Integer
     HOT = 1
     FunctionComp& = True

    DoEvents
     Text1.LinkTopic = "IBM327 32|SessA_TRIMRECT"
    Text1.LinkItem    =  "TRIMRECT(1,1,2 ,4 )"
    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. , 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 = "IBM327 32|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. , use "DoEvents"
' function before starting DDE conversation by
' calling "LinkMode" function.
'
'---------------------------------------------------
'
'
'          Terminate Session Conversation(Hot Link)
'
'
'
'    Terminate DDE Conversation with session (Hot Link)
'
```

```
Sub Command3_Click ()
On Error GoTo ErrHandlerTerm
     Dim NONE As Integer
     NONE =
     FunctionComp& = True

     Text1.LinkMode = NONE

     If FunctionComp&= False Then
          MsgBox "Error has occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandlerTerm:
     FunctionComp& = False
    Resume  Next
End Sub
```

Chapter 11. Server-Requester Programming Interface (SRPI) Support

The Server-Requester Programming Interface (SRPI) is an API that provides access to IBM Enhanced Connectivity Facility (ECF) providing the tools to write SRPI requester programs. SRPI uses a single verb, SEND_REQUEST, to provide a synchronous call-return interface to remote server programs.

PC/3270 SRPI for Windows 95 supports the following programs:

32-bit SRPI Requester Program written in C or C++
The existing 16-bit SRPI Requester Program for PC/3270 for Windows 3.1

How to Use SRPI

You can write the application program using the SRPI in C or C++ for Windows 95. To develop a SRPI application, do as follows:

1. Prepare the source code and add the appropriate SRPI calls.
2. Include the header file UUCCPRB.H in the application program.
3. Compile the source code.
4. Link the resultant .OBJ files with the appropriate object file or libraries.

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

SRPI Compatibility

PC/3270 supports the SRPI function with:

SRPI interface is the same as Personal Communications/3270 Version 3.1.

The SRPI interface has the same function calls as the API calls and parameters of CM/2., except that multiple concurrent SRPI requester applications support the same 3270 logical terminal session.

The return codes for errors or successful operations are the same as with CM/2.

The SRPI interface is usable via a host connect of the emulator in all modes (except asynchronous and Control Unit Terminal connection) when the physical connection to the host is through a token ring or a coaxial cable, or through SNA or non-SNA protocols.

If a call is made to the SRPI interface but there is no response from the host due to a communication failure, an associated error is returned to the caller.

SRPI and EHLLAPI are capable of concurrent operations.

SRPI is supported only for C requester.

Server Alias is not supported.

The 3270 screen update notify is not supported.

© Copyright IBM Corp. 1995, 1996

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.

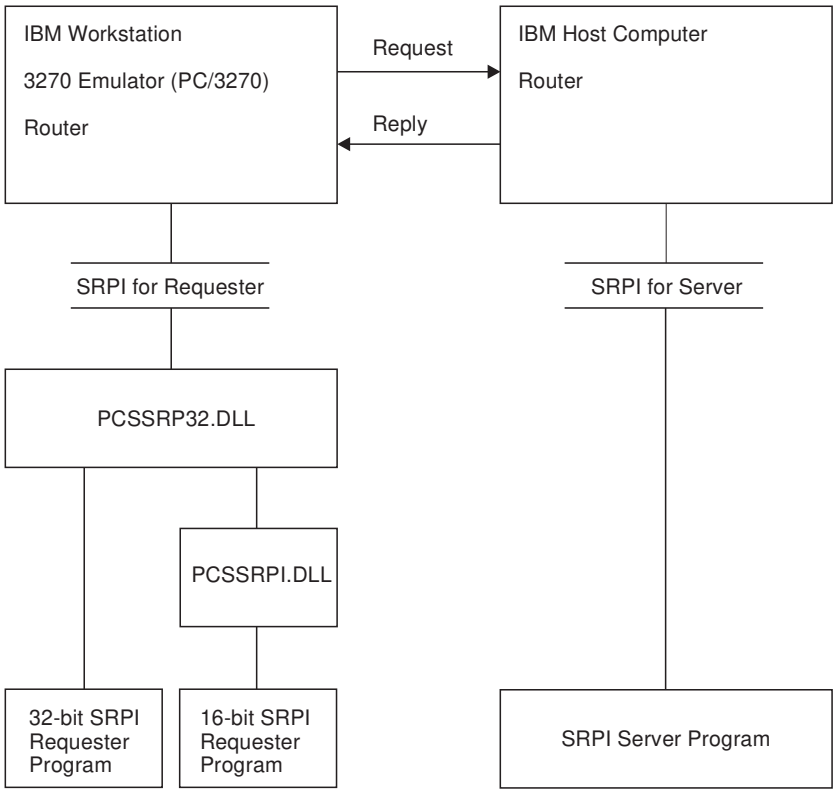

Figure 6. Example of PC/3270 SRPI Requester and Server

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 7 for an illustration of the workstation and host computer relationship.

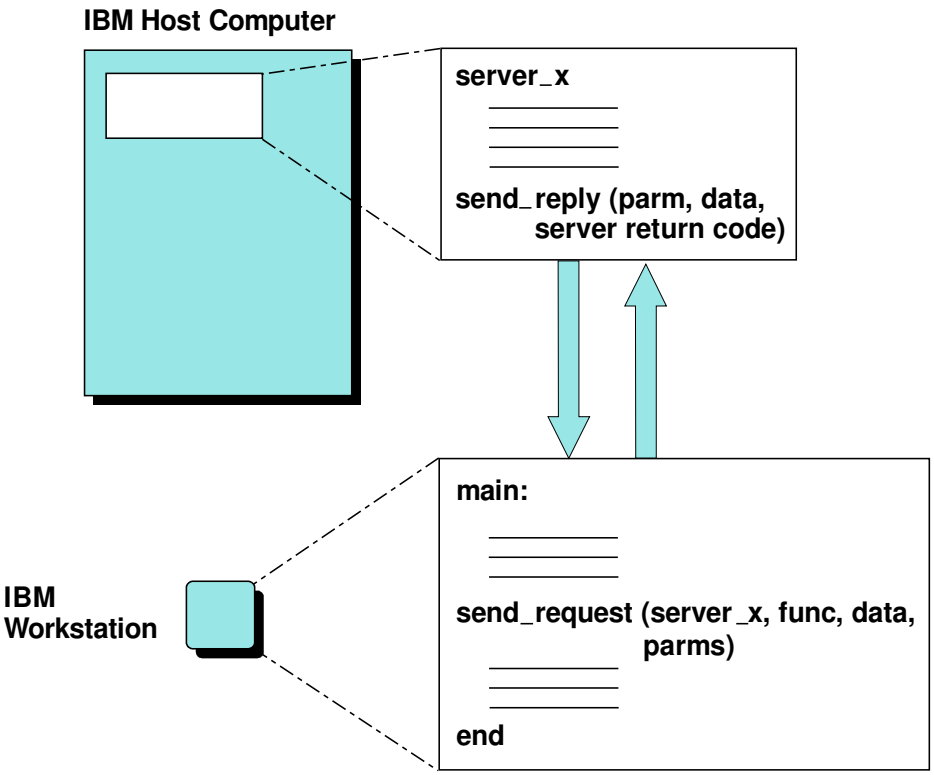

Figure 7. IBM Workstation Requester and IBM Host Computer Server Relationship

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 8 for an illustration of the requester and server flow.

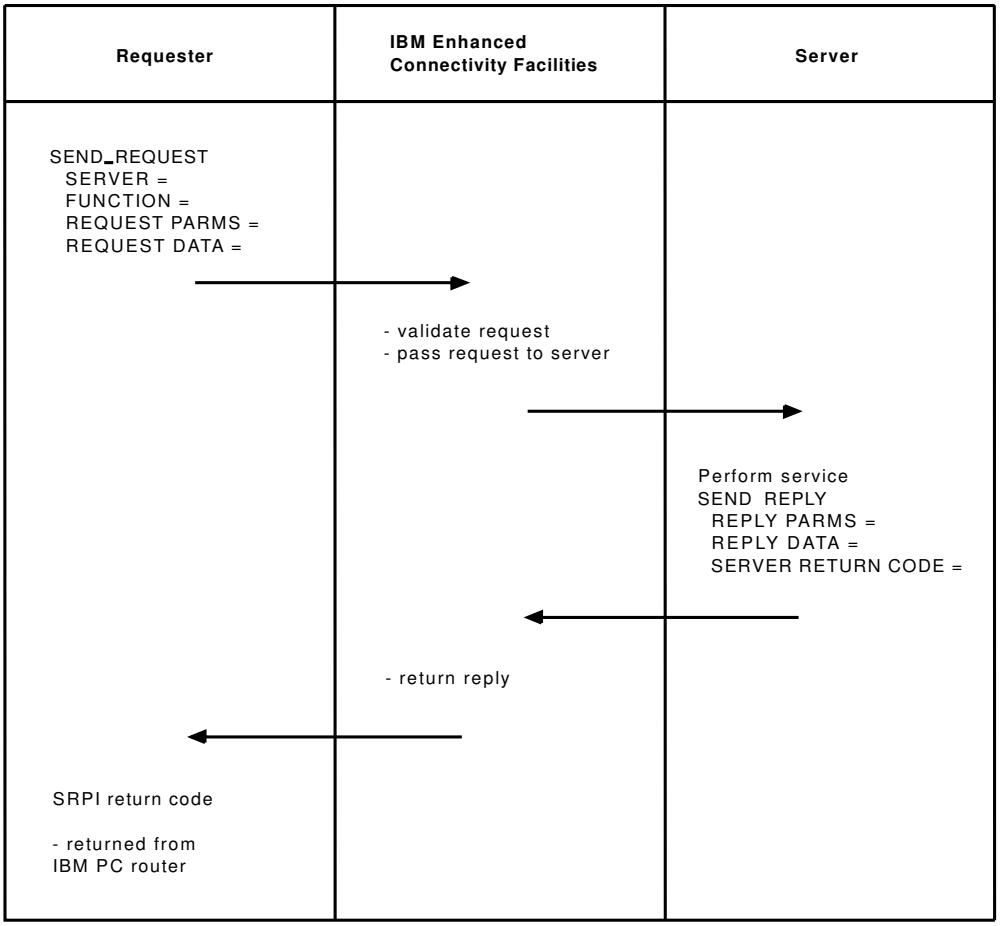

Figure 8. Example of an SRPI Requester and Server Flow

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 103 on page 331 and in Table 104 on page 332.

Supplied Parameters

Table 103 (Page 1 of 2). Parameters Supplied by the SRPI Requester

Name of Parameter	Required/ Optional	Default Value	Description
Server name	Required	Blanks	The name of the host computer server must be 8 bytes long (PC/ASCII), left-justified, and padded with blanks (X'20'); leading blanks, embedded blanks, and names consisting of all blanks are not valid. The valid PC/ASCII characters are *A* through *Z* (uppercase and lowercase), 0 through 9, $, #, and @. The name is converted to EBCDIC before the request is sent to the host computer.
Function ID	Optional	0	A 2-byte unsigned binary number that specifies the server function being requested. Values of 0 through 65535 are valid for specification by an SRPI requester.
Request parameters length	Optional	0	A 2-byte unsigned binary number that specifies the byte length of the request parameters to be passed to the server. Values of 0 through 32763 are valid. A value of 0 indicates that there are no request parameters to be passed.
Request parameters pointer	Optional	0	The 4-byte address of the parameters, if any, to be passed to the server. A nonzero value in the request parameters length indicates that there are parameters to be passed.
Request data length	Optional	0	A 2-byte unsigned binary number that specifies the byte length of the request data to be passed to the server. Values of 0 through 65535 are valid. A value of 0 indicates that there is no request data to be passed.
Request data pointer	Optional	0	The 4-byte address of the data, if any, to be passed to the server. A nonzero value in the request data length indicates that there is data to be passed.
Reply parameters buffer length	Optional	0	A 2-byte unsigned binary number that specifies the length in bytes of the reply parameter buffer supplied by the SRPI requester. Values of 0 through 32763 are valid. A value of 0 indicates that no reply parameters are expected from the server.
Reply parameters buffer pointer	Optional	0	The 4-byte address of the reply parameter buffer. Its presence is indicated by a nonzero value in the reply parameters buffer length.

Table 103 (Page 2 of 2). Parameters Supplied by the SRPI Requester			
Name of Parameter	**Required/ Optional**	**Default Value**	**Description**
Reply data buffer length	Optional	0	A 2-byte unsigned binary number that specifies the length in bytes of the reply data buffer supplied by the SRPI requester. Values of 0 through 65535 are valid. A value of 0 indicates that no reply data is expected from the server.
Reply data buffer pointer	Optional	0	The 4-byte address of the reply data buffer. A nonzero value in the reply data buffer length indicates that there is reply data to be received.

Returned Parameters

Table 104. 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. SeeAppendix 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 Personal Communications Version 4.1 for Windows 95 diskettes and CD-ROM media.

Note: To follow C conventions, the function called **SEND_REQUEST** in other sections is spelled send_request in this section.

C send_request Function

The send_request parameters are grouped into a single C structure of type UERCPRB. The init_send_req_parms function is provided to initialize all send_request parameters in the UERCPRB structure to their defaults. This allows the default values to be set once for parameters not used by a requester. The send_request function is provided to make synchronous calls to the server program.

The init_send_req_parms and send_request functions must be linked with your C application. PCSSRP32.DLL for 32-bit interface and PCSSRPI.DLL for 16-bit interface; both of these object files are provided with PC/3270.

The send_request function copies the contents of the UERCPRB structure into a connectivity programming request block (CPRB) and calls the PCSSRP32.DLL. After the server has completed its processing, the send_request procedure copies the returned parameters from the CPRB into the UERCPRB structure and returns control to the C application.

If the request parameters or data consist of several structures, the application must convert the data or parameters into a single flat structure that consists of a contiguous sequence of bytes that are stored in a buffer. The requesting program must package the request parameters and data in a format recognizable by the server.

UERCPRB is a packed structure. That is, each structure member after the first member is stored at the first available byte.

The memory used for the request parameters can also be used for the reply parameters; the memory used for the request data can also be used for the reply data. The application program must ensure that the reply data and parameters are written into the request data and parameters buffer only when the request data and parameters are no longer needed.

SRPI Record Definition

The UERCPRB record type defines a record passed to the SRPI router using the send_request function. The record is defined in an application program by using the #include preprocessor directive to include the UUCCPRB.H file. For the definitions and value ranges of the supplied and returned parameters, see “Supplied Parameters” on page 331 and “Returned Parameters” on page 332.

SRPI Return Codes

See Appendix D, “SRPI Return Codes” on page 353 for the SRPI return codes.

Appendix A. Query Reply Data Structures Supported by EHLLAPI

This appendix lists and defines the query reply structures supported by the EHLLAPI structured field interface for PC/3270. See the *IBM 3270 Information Display System Data Stream Programmer's Reference* or, in the case of an IBM licensed program, the documentation for the specific licensed program.

Notes:

1. EHLLAPI must scan the query reply buffers to locate the destination/origin ID (DOID) self-defining parameter (SDP) for the structured field support to work and be reliable. The DOID field is then filled in with the assigned ID.
2. The application should build the query reply data structures in the application's private memory.
3. Only cursory checking is performed on the query reply data. Only the ID and the length of the structure are checked for validity.
4. The 2-byte length field at the beginning of each query reply **is not byte reversed**.
5. Only one distributed data management (DDM) base-type connection is allowed per host session. If the DDM connection supports the SDP for the DOID, multiple connections are allowed.
6. If a nonzero return code is received indicating that an application is already connected to the selected session (RC 32 or 39), use that presentation space with caution. Conflicts with SRPI, File Transfer, and other EHLLAPI applications might result.

The DDM Query Reply

Several DDM query reply formats are supported. Here are some of them:

Table 105. DDM Query Reply Base Format

Offset	Length	Content	Meaning
0	1 word	Length	Length of structure
2	1 byte	X'81'	Query reply ID
3	1 byte	X'95'	Query reply type
4–5	2 bytes	FLAGS	Reserved
6–7	2 bytes	LIMIN	Maximum DDM bytes allowed in inbound transmission
8–9	2 bytes	LIMOUT	Maximum DDM bytes allowed in outbound transmission
10	1 byte	NSS	Number of subsets identifier
11	1 byte	DDMSS	DDM subset identifier

© Copyright IBM Corp. 1995, 1996

DDM Application Name Self-Defining Parameter

The DDM Application Name self-defining parameter provides the host application with the name of the application containing control of the DDM Auxiliary Device. The controlling application is identified by the DOID in the Direct Access self-defining parameter.

This 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 106. 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 107. DDM PCLK Auxiliary Device Self-Defining Parameter

Offset	Length	Content	Meaning
0	1 byte	X'04'	Parameter length
1	1 byte	X'03'	PCLK protocol controls
2–3	2 bytes	VERS	Protocol version

VERS. The value given in VERS is used to indicate the versions of PCLK installed in the terminal at the time the query reply is returned. For example, X'0001' indicates PCLK version 1.1.

See the *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

Base DDM Query Reply Formats

The following query reply formats are *examples* of some of the Base + SDP (self-defining parameter) combinations possible. Not all of the combinations are shown.

Table 108. Base DDM Query Reply Format with Name and Direct Access Self-Defining Parameters

Offset	Length	Content	Meaning
0	1 word	Length	Length of structure (includes self-defining parameters)
2	1 byte	X'81'	Query reply ID
3	1 byte	X'95'	Query Reply type
4–5	2 bytes	FLAGS	Reserved
6–7	2 bytes	LIMIN	Maximum DDM bytes allowed in inbound transmission
8–9	2 bytes	LIMOUT	Maximum DDM bytes allowed in outbound transmission
10	1 byte	NSS	Number of subsets supported
11	1 byte	DDMSS	DDM subset identifier
12	1 byte	Length (n+2)	Parameter length
13	1 byte	X'02'	DDM application name
14–(13+n)	n bytes	Name	Name of the remote application program
14+n	1 byte	X'04'	Parameter length
15+n	1 byte	X'01'	Direct access ID
16+n – 17+n	2 bytes	DOID	Destination/origin ID assigned by the subsystem

The self-defining parameters begin at offsets 12 and (14+*n*) where *n* is the length of the application name supplied at offset 14.

See the *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

Table 109. Base DDM Query Reply Format with Direct Access and Name Self-Defining Parameters

Offset	Length	Content	Meaning
0	1 word	Length	Length of structure (includes self-defining parameters)
2	1 byte	X'81'	Query reply ID
3	1 byte	X'95'	Query reply type
4–5	2 bytes	FLAGS	Reserved
6–7	2 bytes	LIMIN	Maximum DDM bytes allowed in inbound transmission
8–9	2 bytes	LIMOUT	Maximum DDM bytes allowed in outbound transmission
10	1 byte	NSS	Number of subsets supported
11	1 byte	DDMSS	DDM subset identifier
12	1 byte	X'04'	Parameter length
13	1 byte	X'01'	Direct access ID
14–15	2 bytes	DOID	Destination/origin ID assigned by the subsystem
16	1 byte	Length (n+2)	Parameter length
17	1 byte	X'02'	DDM application name
16+*n* – 17+*n*	*n* bytes	Name	Name of the remote application program

The self-defining parameters begin at offsets 12 and 16.

See the *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

The IBM Auxiliary Device Query Reply

The Auxiliary Device Reply is used to indicate to the host application the support of an IBM auxiliary device that uses a data stream defined by IBM, see the *IBM 3270 Information Display System Data Stream Programmer's Reference* for more details.

When the function is supported, the query reply is transmitted inbound in reply to a Read Partition structured field specifying Query or Query List (QCODE List = X'9E', Equivalent, or All).

When a workstation supports multiple auxiliary devices, the IBM auxiliary devices query reply must be sent for each device.

Optional Parameters:

All parameters shown in the “base” part of the query reply must be present. Parameters not used are set to X'00'.

At least one self-defining parameter must be present.

Table 110. 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 111 on page 340.

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 111. 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 112. IBM Auxiliary Device PCLK Self-Defining Parameter

Offset	Length	Content	Meaning
0	1 byte	X'04'	Parameter length
1	1 byte	X'03'	PCLK protocol controls
2–3	2 bytes	VERS	Protocol version

VERS. The value given in VERS is used to indicate the versions of PCLK installed in the terminal at the time the query reply is returned. For example, X'0001' indicates PCLK version 1.1.

See the *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

The OEM Auxiliary Device Query Reply

The OEM Auxiliary Device query reply format is as follows:

Table 113. OEM Auxiliary Device Base Format with Direct Access Self-Defining Parameter			
Offset	**Length**	**Content**	**Meaning**
0–1	1 word	Length	Length of structure (includes self-defining parameters)
2	1 byte	X'81'	Query reply ID
3	1 byte	X'8F'	OEM query reply
4–5	2 bytes	FLAGS	Reserved
6–13	4 words	DTYPE	Device type
14–21	4 words	UNAME	User assigned name
22	1 byte	X'04'	Parameter length
23	1 byte	X'01'	Direct access
24–25	1 word	DOID	Destination/origin ID assigned by the subsystem

See the *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

The OEM Auxiliary Device processor supports two self-defining parameters, 01 and 03. These are defined in Table 114.

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 114. 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 115. 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 116. CPR Query Reply Buffer Format			
Offset	**Length**	**Content**	**Meaning**
0	1 word	Length	Length of structure (includes self-defining parameters)
2	1 byte	X'81'	Query reply ID
3	1 byte	X'AB'	Query reply type
4–5	2 bytes	FLAGS	Reserved
6–7	2 bytes	LIMIN	Maximum DDM bytes allowed in inbound transmission
8–9	2 bytes	LIMOUT	Maximum DDM bytes allowed in outbound transmission
10	1 byte	FETAL	Length (in bytes) of the following feature information
11–12	1 word	FEATS	CPR length and feature flags
13–(N*2)+12	0–2 bytes	FEATSs	Additional flags
(N*2)+12	1 byte	X'04'	Length of DOID SDP
(N*2)+13	1 byte	X'01'	Type of D/O ID
(N*2)+14	1 word	DOID	Destination/origin ID assigned by the subsystem

See the *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

The Product-Defined Query Reply

This query reply is used by IBM products using registered subidentifiers within the X'9C' data structure. The Product-Defined Data Stream query reply indicates support of a 3270DS workstation auxiliary device that uses an IBM product-defined data stream. The data stream is *not* defined by a format architecture document having an identifiable control point such as an architecture review board.

When an auxiliary device supports an IBM product-defined data stream, this query reply is transmitted inbound in reply to a Query List (QCODE List = X'9C' or All).

Optional Parameters

All parameters shown in the base part of the query reply and the Direct Access self-defining parameter must be present.

The format of the Product-Defined query reply is as follows:

Table 117. 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 118. Valid REFID and SSID Values for the IBM Product-Defined Query Reply

REFID	SSID	Product and Data Stream Documentation
X'01'		5080 Graphics System: This reference ID indicates the 5080 Graphics System data stream is supported by the auxiliary device. Descriptions of the 5080 Graphics Architecture, structured field, subset ID, DOID, and associated function sets are defined in the *IBM 5080 Graphics System Principles of Operation*.
	X'01' X'02'	5080 HGFD Graphics Subset 5080 RS232 Ports Subset
X'02'		WHIP API (replaced by SRL name when written) This reference ID indicates that the WHIP API data stream is supported by the auxiliary device. A description of the WHIP API architecture is defined in *IBM RT PC Workstation Host Interface Program Version 1.1 User's Guide and Reference Manual*
	X'01'	WHIP Subset 1
X'03' to X'FF'		All other values are reserved.

The IBM Product-Defined processor supports only the Direct Access self-defining parameter. It is defined in Table 119 on page 344.

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 119. 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 120. 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 121 on page 345.

Direct Access Self-Defining Parameter

The presence of the Direct Access ID self defining parameter indicates that the auxiliary device can be accessed directly by using the destination/origin structured field.

Table 121. IBM Product-Defined Direct Access Self-Defining Parameter

Offset	Length	Content	Meaning
0	1 byte	X'04'	Parameter length
1	1 byte	X'01'	Direct access ID
2–3	2 bytes	DOID	Destination/origin ID

DOID. The value in these bytes is used in the ID field of the destination/origin structured field to identify the auxiliary device as the destination or origin of the data that follows.

See the *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

Appendix B. Compatibility with Personal Communications 4.1 for Windows 3.1

This appendix describes the compatibility between EHLLAPI/DDE of PC/3270 and PC400 for Windows 95 and that for Windows 3.1.

The EHLLAPI of PC/3270 and PC400 for Windows 95 is different from that for Windows 3.1 in the following ways:

- The entry point name ***hllapiw*** that is provided by PC/3270 and PC400 for Windows 3.1 is not supported.
- The event object is newly supported for the asynchronous mode of the notification used by Start Close Intercept (41), Start Host Notification (23), Read Structured Fields (126) and Write Structured Fields (127). It can be used by 32-bit applications for Windows 95.

The EHLLAPI of PC/3270 and PC400 for Windows 95 does not support the DOS-mode EHLLAPI.

The DDE of PC/3270 and PC400 for Windows 95 is different from that for Windows 3.1 in the following ways:

- Code Conversion function is not supported, use an appropriate API on Personal Communications for EBCDIC/ASCII conversion.
- The application name ***IBM3270*** or ***IBM5250*** is for 16-bit applications. ***IBM327032*** and ***IBM525032*** are for 32-bit applications.

© Copyright IBM Corp. 1995, 1996

Appendix C. Compatibility with Communications Manager/2

This appendix describes the compatibility between EHLLAPI of PC/3270 and PC400 for Windows 95 and Communications Manager/2 (CM/2).

The following EHLLAPI function is not supported by PC/3270 or PC400:

Storage Manager (17)

The following EHLLAPI functions are different from those with the same names in CM/2. You need to understand the differences when you use these functions:

Set Session Parameter (9)
Copy OIA (13)
Copy String to PS (15)
Copy String to Field (33)
Get Key (51)
Window Status (104)
Query Sessions (10)
Connect for Structured Field (120)
Allocate Communications Buffer (123)
ASCII mnemonics

Set Session Parameter (9)

Set Options

Personal Communications does not provide the following set options provided by Communications Manager:

OLDOIA, NEWOIA
COMPCASE, COMPICASE
OLD5250OIA, NEW5250OIA

Return Parameters

When the **Set Session Parameter** (9) function is terminated, Communications Manager returns a length of the valid data string as the third parameter, the data string length.
However, Personal Communications returns a number of the valid set options as the data string length.

EAB Option

In CM/2, a color remap affects the value of the character color in the EAB attribute copied by **Copy PS** (5) or **Copy PS to String** (8) function when the EAB option is specified in the **Set Session Parameter** (9) function.

In Personal Communications, however, the value of the character color in the EAB attribute depends on the contents of the presentation space regardless of a color remap, and it is not affected by a color remap.

© Copyright IBM Corp. 1995, 1996

Copy OIA (13)

The **Copy OIA** (13) function has the following differences between CM/2 and PC/3270 or PC400. For more information of the group and the column positions, refer to “Copy OIA (13)” on page 66.

Byte Position 21

- Personal Communications returns X'F6'.
- CM/2 returns X'20'.

Byte Positions 61–63

- Personal Communications does not return the printer information.
- CM/2 returns the printer information.

Group 3: Shift State

CM/2 does not return the value of bit 2. Bit 2 is reserved, and bit 0 contains both the Upper Shift and the Caps Lock.

Group 8 Byte 1: Input Inhibited

- Personal Communications does not return bit 6 (Device not working).
- CM/2 can return bit 6.

Group 8 Byte 3: Input Inhibited

- Personal Communications does not return bit 1 (Operator unauthorized) and bit 2 (Operator unauthorized -f).
- CM/2 can return bits 1 and 2.

Group 8 Byte 4: Input Inhibited

- Personal Communications does not return bit 2 (System wait).
- CM/2 can return bit 2.

Group 10: Highlight Group 2

- Personal Communications does not return bit 0 (Selected).
- CM/2 can return bit 0.

Group 11: Color Group 2

- Personal Communications does not return bit 0 (Selected).
- CM/2 can return bit 0.

Group 13: Printer Status

- In Personal Communications, this group is reserved.
- CM/2 can return this group.

Group 14: Graphics

CM/2 does not return bit 0 (Graphic cursor).

Copy String to PS (15)

In CM/2, the EAB option of the **Set Session Parameter** (9) function affects the **Copy String to PS** function. When you specify the EAB option, pass the attribute data that has the same size as the text data to the function with the text data.

In Personal Communications, however, the data to be passed is only text data regardless of EAB option. If you want to use the same interface with CM/2, use the PUTEAB option of **Set Session Parameter** (9).

Storage Manager (17)

Storage Manager (17) function provided by CM/2 is not supported by Personal Communications. Use the APIs provided by Windows 95 to allocate the memory for the applications.

Copy String to Field (33)

In CM/2, when the EAB option of the **Set Session Parameter** (9) function is specified, the attribute data is passed to the function as a part of the data. Therefore, when you specify the EAB option, pass the attribute data that has the same size as the text data to the function with the text data.

In Personal Communications, however, the EAB option does not affect the data contents of the **Copy String to Field** (33) function. The data to be passed is not the attribute data, but only the text data. If you want to use the same interface with CM/2, use the PUTEAB option of **Set Session Parameter** (9).

Get Key (51)

CM/2 returns shift state using @A, @S, or @r, if the shift state of a passed key is not a key or function recognized by the emulator session. Personal Communications does not support these ASCII mnemonics.

Window Status (104)

EHLLAPI function 104 (PM_WINDOW_STATUS) 'query extended status' command (0x03) will return the handle of the emulator presentation space window. This is consistent with the definition of the function and the CM/2 implementation. However, Personal Communications AS/400 and 3270 for Windows EHLLAPI returns the handle of the frame window. EHLLAPI applications written for Personal Communications AS/400 and 3270 for Windows using this function will need to use the parent of the window handle returned.

Query Sessions (10)

In CM/2 and Personal Communications for OS/2, the descriptor for personal computer is returned.

But in Personal Communications for Window, it is not returned.

Connect for Structured Fields (120)

The event object for communication connection status provided by CM/2 is not in Personal Communications.

Allocate Communications Buffer (123)

In CM/2, the maximum value of the requested buffer size is 64 KB minus 8 bytes (X'FFF8').

In Personal Communications, however, it is 64 KB minus 256 bytes (X'FF00').

ASCII Mnemonics

The following ASCII mnemonics are not supported in Personal Communications:

Mnemonics	Meaning
@A@N	Get Cursor
@A@O	Locate Cursor
@A@X	Hexadecimal
@A@Y	Cmd (Function) Key
@A@a	Destructive Backspace
@S@A	Erase EOL
@S@B	Field Advance
@S@C	Field Backspace
@S@D	Valid Character Backspace
@S@P	POR (For sending only)
@S@T	Jump to Task Manager
@/	Overrun of queue (Only in the **Get Key** function)

Appendix D. SRPI Return Codes

This appendix describes error handling in the SRPI environment. Types 0, 1, 2, and 3 return codes and their definitions are listed. Exception class definitions, code values, and object values are listed. Server return codes are also discussed.

Error Handling

An unsuccessful service request in the SRPI environment can result from problems at any of the different layers. SRPI shields applications from transport layer errors as much as possible. Errors within server processing are handled by the applications. The other errors are caused by SRPI and are treated accordingly.

Transport Layer Errors

SRPI tries to recover from transport layer errors. When recovery is not possible, SRPI returns to the requester with a return code indicating transport layer failure. The programmer should handle such failures using the problem determination procedures of the transport mechanism.

Application Errors

SRPI is responsible for routing requests to servers and returning replies to requesters. Requesters and servers are responsible for handling errors (except for abend) that servers encounter. When a server ends abnormally, SRPI returns to the requester with an abend notice in the SRPI return code.

The server return code is set by the server on the IBM host computer running under VM or MVS. The value and meaning of the server return code is dependent on the requester or the server.

SEND_REQUEST Processing Errors

SRPI return codes can encounter a number of errors in processing the **SEND_REQUEST** function. Such errors include:

Incorrect function parameters
Unidentified server
Inability to contact the server

There are also system error codes for internal SRPI errors.

Types of SRPI Return Codes

SRPI return codes include types 0, 1, 2, and 3:

Type 0 Indicates successful completion of the **SEND_REQUEST** function.

Type 1 Indicates errors detected by the SRPI router that prevent a request from being processed.

Type 2 Indicates errors detected by the SRPI router and reported to the remote computer by an acknowledge interchange unit.

Type 3 Indicates errors detected by the remote computer and reported to the SRPI router by an acknowledge interchange unit.

© Copyright IBM Corp. 1995, 1996

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 122.

Table 122 (Page 1 of 3). Type 1 Return Code Definitions and Descriptions

Hexadecimal Return Code	Constant Return Code	Description
X'0100 0402'	UERERRT1START	SRPI is not started because the host ECF program is not started.
X'0100 0404'	UERERRT1LOAD	The SRPI router is not loaded.
X'0100 0408'	UERERRT1BUSY	The SRPI router is busy. This return code is not used by the Personal Communications program.
X'0100 040A'	UERERRT1VER	The version ID in the CPRB passed to the SRPI router is not supported by the resident portion of the SRPI router. The version ID is automatically put into the CPRB by the C interface facility.
X'0100 040C'	UERERRT1EMU	Personal Communications is not loaded.
X'0100 040E'	UERERRT1ROUT	The server name supplied in the CPRB is not defined in the server routing table. Default routing is not configured so SRPI is unable to route the request. Use a valid server name or update the configuration to include the server name.
X'0100 0410'	UERERRT1COMMR	Communications resource not available.
X'0100 0412'	UERERRT1REST	3270 emulation has been restarted since the application last used SRPI. End the application and restart it before using SRPI.
X'0100 0414'	UERERRT1INUSE	The request has been routed to a communication session that is in use by File Transfer.
X'0100 0602'	UERERRT1QPLEN	Request parameters length exceeds the maximum value. The maximum value allowed is 32763.
X'0100 0604'	UERERRT1RPLEN	Reply parameters buffer length exceeds the maximum value. The maximum value allowed is 32763.
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.

Table 122 (Page 2 of 3). Type 1 Return Code Definitions and Descriptions

Hexadecimal Return Code	Constant Return Code	Description
X'0100 060C'	UERERRT1QPAD	One of the following conditions exists: The request parameter address is not valid. The request parameter length extends beyond the end of the request parameter buffer. The request parameter address is 0 with a nonzero request parameter length.
X'0100 060E'	UERERRT1QDAD	One of the following conditions exists: The request data address is not valid. The request data length extends beyond the end of the request data buffer. The request data address is 0 with a nonzero request data length.
X'0100 0610'	UERERRT1RPAD	One of the following conditions exists: The reply parameter buffer address is not valid. The reply parameter buffer length extends beyond the end of the reply parameter buffer. The reply parameter buffer address is 0 with a nonzero reply parameter length.
X'0100 0612'	UERERRT1RDAD	One of the following conditions exists: The reply data buffer address is not valid. The reply data buffer length extends beyond the end of the reply data buffer. The reply data buffer address is 0 with a nonzero reply data length.
X'0100 0616'	UERERRT1TOPV	The TopView environment is not supported. This return code is not used by the Personal Communications program.
X'0100 0622'	UERERRT1INV3270 d	Notification of 3270 screen update indicator is not valid. The Notification of 3270 screen update indicator must be set to X'00' (notify user of 3270 screen update) or X'FF' (suppress user notification of 3270 screen update) in the CPRB.
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.

Table 122 (Page 3 of 3). Type 1 Return Code Definitions and Descriptions		
Hexadecimal Return Code	**Constant Return Code**	**Description**
X'0100 0C00'	UERERRT1CONV	A system error occurred. Conversation with the host ended for one of the following reasons: The host communication session is not active. A link-level communication error occurred. The system was unable to transmit data reliably to or from the host. For example, a sequence error occurred.
X'0100 0C02'	UERERRT1ISE	A system error occurred because of an internal software error in the SRPI router.
X'0100 0C04'	UERERRT1PROT	A system error occurred. This is a protocol violation error or a system software error in the SRPI router or the host.
X'0100 0C06'	UERERRT1SYIN	A system error occurred. The error is caused by system inconsistency. This is a system software error in the SRPI router.

Type 2 Return Code Definitions

Type 2 return codes have the following format: X'02xx yyzz'.

The 3 error-specific bytes consist of the following exception conditions from the acknowledge interchange unit:

xx exception class
yy exception code
zz exception object

Note: No constants are supplied.

Type 3 Return Code Definitions

Type 3 return codes have the following format: X'03xx yyzz'.

The 3 error-specific bytes consist of the following exception conditions from the acknowledge interchange unit:

xx exception class
yy exception code
zz exception object

The return code definitions and descriptions are listed in Table 123.

Table 123. 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 124.

Table 124. Class Definitions for Type 2 and Type 3	
Value	**Definition**
X'00' to X'01'	Reserved
X'02'	Syntax
X'03'	Semantic
X'04'	Process
X'05' to X'FF'	Reserved

Exception Code Values for Type 2 and Type 3

The exception code defines a specific error condition and is required with every error. The exception code values are listed in Table 125.

Table 125. 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 126.

Table 126. Exception Object Values for Type 2 and Type 3

Value	Definition
X'00'	Not specified
X'01'	Prefix
X'07'	Command operand
X'08'	Command subfields
X'1C'	Parameters operand
X'1D'	Data operand
X'13'	Suffix

Note: All exception object values not specified in this table are reserved.

Server Return Codes

A server return code is a doubleword (4-byte) return code supplied by the server program and is returned to the requester program. The contents and meaning of the return status are defined by the requester or the server. For information about server return codes, contact your host personnel or see one of the following manuals:

TSO/E Version 2 Guide to the Server-Requester Programming Interface

IBM Programmer's Guide to the Server-Requester Programming Interface for VM/System Product

Appendix 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 for 16-bit to 32-bit mode.

PC/3270 and PC400 provide a dynamic data exchange (DDE) interface that allows applications to exchange data. The exchange of data between two Windows applications can be thought of as a conversation between a client and a server. The *client* initiates DDE conversations. The *server* in turn responds to the client. PC/3270 and PC400 is a DDE server for the open sessions that PC/3270 and PC400 are managing. For more information about DDE, refer to *Microsoft Windows Software Development Kit Guide to Programming*.

Note: If you use DDE functions with Visual Basic, see Chapter 10, “Using DDE Functions with a DDE Client Application” on page 293.

PC/3270 and PC400 DDE Data Items in a 16-bit Environment

Microsoft Windows DDE uses a three-level naming scheme to identify data items: application, topic, and item. Table 127 on page 361 describes these levels.

Table 127. Naming Scheme for Data Items

Level	Description	Example
Application	A Windows task or a particular task of an application. In this book, the application is PC/3270 or PC400.	IBM3270
Topic	A specific part of an application.	SessionA
Item	A data object that can be passed in a data exchange. An item is an application-defined data item that conforms to one of the Windows clipboard formats or to a private, application-defined, clipboard format. For more information regarding Windows clipboard formats, refer to *Microsoft Windows Software Development Kit Guide to Programming*.	PS (presentation space)

PC/3270 and PC400 support one application name as Windows DDE server.

Table 128. Applications for PC/3270 and PC400

Application
IBM3270 IBM5250

© Copyright IBM Corp. 1995, 1996

The following table lists the topics you can use.

Table 129. Topics for PC/3270 and PC400

Topics
System SessionA, SessionB, ..., SessionZ LUA_xxxx, LUB_xxxx, ..., LUZ_xxxx

In DDE, *atoms* identify application names, topic names, and data items. Atoms represent a character string that is reduced to a unique integer value. The character string is added to an atom table, which can be referred to for the value of the string associated with an atom. Atoms are created with the GlobalAddAtom function call. Refer to *Microsoft Windows Software Development Kit Guide to Programming* for more information about how to create and use atoms.

Using System Topic Data Items

Applications that provide a DDE interface should also provide a special topic SYSTEM. This topic provides a context for items of information that might be of general interest to an application. The SYSTEM topic for PC/3270 or PC400 contains these associated data items:

Item	Function
Formats	Returns the list of clipboard formats (numbers) that PC/3270 or PC400 is capable of rendering.
Status	Returns information about the status of each PC/3270 or PC400 session.
SysCon	Returns the level of PC/3270 or PC400 support and other system related values.
SysItems	Returns the list of data items that are available when connected to the PC/3270 or PC400 system topic.
Topics	Returns the list of PC/3270 or PC400 topics that are available.

Using Session Topic Data Items

For each Session topic, the following data items are supported:

Item	Function
CLOSE	Retrieves the window close requests.
EPS	Retrieves the session presentation space with additional data.
EPSCOND	Retrieves the presentation space service condition.
FIELD	Retrieves the field in the presentation space of the session.
KEYS	Retrieves the keystrokes.
MOUSE	Retrieves the mouse input.
OIA	Retrieves the operator information area status line.
PS	Retrieves the session presentation space.
PSCOND	Retrieves the session advise condition.
SSTAT	Retrieves the session status.
STRING	Retrieves the ASCII string data.
TRIMRECT	Retrieves the session presentation space within the current trim rectangle.

Using LU Topic Data Items (PC/3270 Only)

For each LU topic, the following data items are supported:

Item	Function
SF	Retrieves the destination/origin structured field data.
SFCOND	Retrieves the query reply data.

DDE Functions IN A 16-Bit Environment

The following table lists the DDE functions that are available for use with PC/3270 or PC400 and the page in this section, where each function is more fully documented.

Function	PC/3270 Windows	PC400	Page
Find Field	X	X	366
Get Keystrokes	X	X	368
Get Mouse Input	X	X	370
Get Number of Close Requests	X	X	373
Get Operator Information Area	X	X	374
Get Partial Presentation Space	X	X	375
Get Presentation Space	X	X	378
Get Session Status	X	X	381
Get System Configuration	X	X	383
Get System Formats	X	X	221
Get System Status	X	X	222
Get System SysItems	X	X	223
Get System Topics	X	X	224
Get Trim Rectangle	X	X	225
Initiate Session Conversation	X	X	226
Initiate Structured Field Conversation	X	–	227
Initiate System Conversation	X	X	228
Put Data to Presentation Space	X	X	229
Search for String	X	X	231
Send Keystrokes	X	X	232
Session Execute Macro	X	X	234
Set Cursor Position	X	X	241
Set Mouse Intercept Condition	X	X	242
Set Presentation Space Service Condition	X	X	245
Set Session Advise Condition	X	X	247
Set Structured Field Service Condition	X	–	249
Start Close Intercept	X	X	250
Start Keystroke Intercept	X	X	252
Start Mouse Input Intercept	X	X	253
Start Read SF	X	–	257
Start Session Advise	X	X	259
Stop Close Intercept	X	X	261
Stop Keystroke Intercept	X	X	262
Stop Mouse Input Intercept	X	X	263
Stop Read SF	X	–	264
Stop Session Advise	X	X	265
Terminate Session Conversation	X	X	266
Terminate Structured Field Conversation	X	–	267
Terminate System Conversation	X	X	268
Write SF	X	–	269

Refer to “Summary of DDE Functions in a 16-Bit Envionment” on page 432 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

PC/3270	PC400
X	X

The find field function returns to the client the information of the field specified by the set presentation space service condition function.

Note: The client must set the PS position by using the set presentation service condition function before using this function.

The client sends this following message to receive the field information.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aFIELD) );
```

where:

cfFormat Identifies the format for the field information.
Valid value must be CF_DSPTEXT.

aFIELD Identifies field data item.

The Find Field function supports a new format like Visual Basic. Using the new format, the Find Field function can find a field with specifying its type. The new format is:

FIELD (pos, type)

pos Position where PC/3270 or PC400 starts to search a target field.
type Target field type. The field type are:

Type	Meaning
or T	This field.
P	The previous field, either protected or unprotected.
N	The next field, either protected or unprotected.
NP	The next protected field.
NU	The next unprotected field.
PP	The previous protected field.
PU	The previous unprotected field.

Note: The symbol represents a required blank.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the following information of the field in a DDE data message,

Start PS position
Length
Attribute value

```
WM_DDE_DATA(hData, aFIELD)
```

or responds with an ACK message containing status information.

```
WM_DDE_ACK(wStatus, aFIELD)
```

If PC/3270 or PC400 cannot return the field information, the following status codes will be returned in the low order byte of the wStatus word:

Return Code	Explanation
1	PS position is not valid.
2	PS is unformatted.
6	The specified format is not valid.
9	A system error occurred.

Structure of the Field Information

PC/3270 or PC400 returns the field information in the following structure:

```
typedef struct tagFINDFIELD
{                                   //
 unsigned unused:12;                //     unused
 unsigned fResponse:1;              // TRUE = DDE_REQUEST response
 unsigned fRelease:1;               // TRUE = Client frees this data
 unsigned reserved:1;               //     reserved
 unsigned fAckReq:1;                // TRUE = Client returns DDE_ACK
 int          cfFormat;             // Format of Field data CF_DSPTEXT
 unsigned char cAttribute;    // Attribute character
 unsigned uFieldStart;              // Field start offset
 unsigned uFieldLength;             // Field Length;
}    FINDFIELD, far  lpFINDFIELD;
```

Get Keystrokes

PC/3270	*PC400*
X	X

The Get Keystrokes function returns to the client the keystrokes that are intercepted by the Start Keystroke Intercept function. The client sends the following message to receive the keystroke information.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aKEYS) );
```

where:

cfFormat Identifies the format for the keystroke information. Valid value must be CF_DSPTEXT.

aKEYS Identifies keystroke data item.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the keystrokes in a DDE data message, or responds with one of these ACK messages containing status information:

WM_DDE_DATA(hData, aKEYS)
WM_DDE_ACK(wStatus, aKEYS)

If PC/3270 or PC400 cannot return the keystroke information, the following status codes will be returned in the low-order byte of the wStatus word:

Return Code	Explanation
2	No keystroke was intercepted.
6	The specified format is not valid.
9	A system error occurred.

Structure of the Keystroke Information

PC/3270 or PC400 returns the keystroke information in the following structure:

```
typedef struct tagKEYSTROKE
{                                   //
 unsigned unused:12;                //     unused
 unsigned fResponse:1;              // TRUE = DDE_REQUEST response
 unsigned fRelease:1;               // TRUE = Client frees this data
 unsigned reserved:1;               //     reserved
 unsigned fAckReq:1;                // TRUE = Client returns DDE_ACK
 int            cfFormat;           // Format of keystroke data CF_DSPTEXT
 unsigned uTextType;                // Type of keystrokes
 unsigned char szKeyData[1];// Keystrokes
}    KEYSTROKE, far  lpKEYSTROKE;
```

The format for the keystrokes parameters is the same as for the Session Execute Macro function SENDKEY command.

The following key text types are supported:

```
WC_CHARACTER      // Pure text, no command
WC_TOKEN          1 // including commands
```

Get Mouse Input

PC/3270	*PC400*
X	X

The Get Mouse Input function returns the latest mouse input that are intercepted by Start Mouse Input Intercept function to the client.

Note: The client must call the Start Mouse Input Intercept function before using this function.

The client sends the following command to receive the mouse input information.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aMOUSE) );
```

where:

cfFormat Identifies the format for the presentation space. Valid values are CF_TEXT or CF_DSPTEXT. The structure of the mouse input data, in these two formats, is shown below.

aMOUSE Identifies the mouse input as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the mouse input data in a DDE data message, or responds with one of these ACK messages:

WM_DDE_DATA(hData, aMOUSE)
WM_DDE_ACK(wStatus, aMOUSE)

If PC/3270 or PC400 cannot return the mouse input information, the following status codes are returned in the low-order byte of the wStatus word:

Return Code	Explanation
2	No mouse input information was intercepted.
6	The specified format is not valid.
9	A system error occurred.

Structure of the Mouse Input Information

If the format is CF_TEXT, PC/3270 or PC400 returns the mouse input information in the following format:

```
typedef struct tagMOUSE_CF_TEXT
{
 unsigned          unused:12,          //     Unused
  unsigned         fRespons:1,         // TRUE = DDE_REQUEST response
  unsigned         fRelease:1,         // TRUE = Client frees this data
  unsigned         reserved:1,         //       Reserved
 unsigned          fAckReq:1;           // TRUE = Client returns DDE_ACK
 int               cfFormat;            // CF_TEXT
 unsigned char PSPos[4];              // PS position
 unsigned char Tab1[1];               // TAB character
 unsigned char PSRowPos[4];           // PS row position
 unsigned char Tab2[1];               // TAB character
 unsigned char PSColPos[4];           // PS columns position
 unsigned char Tab3[1];               // TAB character
 unsigned char PSSize[4];             // Size of the PS
 unsigned char Tab4[1];               // TAB character
 unsigned char PSRows[4];             // PS number of rows
 unsigned char Tab5[1];               // TAB character
 unsigned char PSCols[4];             // PS number of columns
 unsigned char Tab6[1];               // TAB character
 unsigned char ButtonType[1];         // Pressed button type
 unsigned char Tab7[1];               // TAB character
 unsigned char ClickType[1];          // Click type
 unsigned char Tab8[1];               // TAB character
 unsigned char ClickString[1];    // Retrieved string
} MOUSE_CF_TEXT, FAR  lpMOUSE_CF_TEXT;
```

The following table shows the values in the parameters:

Parameter Name	Meaning	Value
PSPos	PS offset of the position where mouse was clicked	0 ... (PSSize – 1)
PSRowPos	Row number of the position where mouse was clicked	0 ... (PSRows – 1)
PSColPos	Column number of the position where mouse was clicked	0 ... (PSCols – 1)
PSSize	Size of the presentation space	
PSRows	Number of rows of presentation space	
PSCols	Number of columns of presentation space	
ButtonType	Type of the clicked mouse button	**L** Left button **M** Middle button **R** Right button
ClickType	Type of clicking	**S** Single click **D** Double click
ClickString	Retrieved string to which the mouse pointed	A character string terminated with a '\0'
Tab1–8	A tab character for delimiter	'\t'

If the format is CF_DSPTEXT, PC/3270 or PC400 returns the mouse input information in the following format:

```
typedef struct tagMOUSE_CF_DSPTEXT
{
 unsigned          unused:12,              //     Unused
 unsigned          fRespons:1,             // TRUE = DDE_REQUEST response
 unsigned          fRelease:1,             // TRUE = client frees the storage
 unsigned          reserved:1,             //      Reserved
 unsigned          fAckReq:1;              // TRUE = client returns DDE_ACK
 int               cfFormat;               // CF_DSPTEXT
 unsigned          uPSPos;                 // PS position
 unsigned          uPSRowPos;              // PS row position
 unsigned          uPSColPos;              // PS column position
 unsigned          uPSSize;                // Size of the presentation space
 unsigned          uPSRows;                // PS number of rows
 unsigned          uPSCols;                // PS number of columns
 unsigned          uButtonType;            // Pressed button type
 unsigned          uClickType;             // Click type
 unsigned char szClickString[1];    // Retrieved string
} MOUSE_CF_DSPTEXT, FAR  lpMOUSE_CF_DSPTEXT;
```

The following table shows the values in the parameters:

Parameter Name	Meaning	Value
uPSPos	PS offset of the position where mouse was clicked	0 ... (uPSSize – 1)
uPSRowPos	Row number of the position where mouse was clicked	0 ... (uPSRows – 1)
uPSColPos	Column number of the position where mouse was clicked	0 ... (uPSCols – 1)
uPSSize	Size of the presentation space	
uPSRows	Number of rows of the presentation space	
uPSCols	Number of columns of the presentation space	
uButtonType	Type of the clicked mouse button	**0x0001** Left button **0x0002** Middle button **0x0003** Right button
uClickType	Type of clicking	**0x0001** Single click **0x0002** Double click
szClickString	Retrieved string that the mouse pointed to	A character string terminated with a '\0'

Get Number of Close Requests

PC/3270	PC400
X	X

The Get Number of Close Requests function returns to the client the number of the close requests that are intercepted by the Start Close Intercept function. The client sends the following message to receive the number of the close requests.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aCLOSE) );
```

where:

cfFormat Identifies the format for the close intercept information. Valid value must be CF_DSPTEXT.

aCLOSE Identifies close intercept data item.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the number of the close requests in a DDE data message, or responds with one of these ACK messages:

WM_DDE_DATA(hData, aCLOSE)
WM_DDE_ACK(wStatus, aCLOSE)

If PC/3270 or PC400 cannot return the close intercept information, the following status codes will be returned in the low order byte of the wStatus word:

Return Code	Explanation
6	The specified format is not valid.
9	A system error occurred.

Structure of the Number of the Close Requests Information

PC/3270 or PC400 returns the close intercept information in the following structure:

```
typedef struct tagCLOSEREQ
{                                       //
 unsigned unused:12;                    //     unused
 unsigned fResponse:1;                  // TRUE = DDE_REQUEST response
 unsigned fRelease:1;                   // TRUE = Client frees this data
 unsigned reserved:1;                   //     reserved
 unsigned fAckReq:1;                    // TRUE = Client returns DDE_ACK
 int          cfFormat;                 // Format of close intercept data CF_DSPTEXT
 unsigned uCloseReqCount;               // Number of the close requests.
}    CLOSEREQ, far  lpCLOSEREQ;
```

Get Operator Information Area

PC/3270	*PC400*
X	X

The Get Operator Information Area (OIA) function returns a copy of the OIA to the client. The client sends the following message to request the OIA.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aOIA) );
```

where:

cfFormat Identifies the format for the OIA. For the OIA this format must be CF_DSPTEXT.

aOIA Identifies the operator information area as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the OIA in a DDE data message, or responds with one of these ACK messages:

WM_DDE_DATA(hData, aOIA)
WM_DDE_ACK(wStatus, aOIA)

If PC/3270 or PC400 cannot return the OIA, the following status codes will be returned in the low order byte of the wStatus word:

Return Code	Explanation
6	The specified format is not valid.
9	A system error occurred.

Structure of the Operator Information Area

PC/3270 or PC400 returns the operator information area in the following structure:

```
typedef struct tagOIADATA
{                                       //
 unsigned unused:12;                  //     unused
 unsigned fResponse:1;                // TRUE = DDE_REQUEST response
 unsigned fRelease:1;                 // TRUE = Client frees this data
 unsigned reserved:1;                 //     reserved
 unsigned fAckReq:1;                    // TRUE = Client returns DDE_ACK
 int    cfFormat;                       // Format of OIA data CF_DSPTEXT
 unsigned char OIA[ ];                //
}    OIADATA, far  lpOIADATA;
```

Get Partial Presentation Space

PC/3270	*PC400*
X	X

The Get Partial Presentation Space function returns all or part of the session presentation space to the client.

Note: The client must set the start PS position and the length (or set the EOF flag) by using the Set Presentation Space Service Condition function before using this function.

The client sends the following command to get the presentation space.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aEPS) );
```

where:

cfFormat Identifies the format for the presentation space. Valid values are CF_TEXT or CF_DSPTEXT. The structure of the presentation space, in these two formats, is shown below.

aEPS Identifies presentation space atom as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the presentation space data, or responds with one of these ACK messages containing an error code in the low order byte of the wStatus word:

WM_DDE_DATA(hData, aEPS)
WM_DDE_ACK(wStatus, aEPS)

If PC/3270 or PC400 cannot return the presentation space, the following status codes will be returned in the low order byte of the wStatus word:

Return Code	Explanation
1	No prior Set Presentation Space Service Condition function was called, or an incorrect parameter was set.
6	The specified format is not valid.
9	A system error occurred.

Structure of the Presentation Space

PC/3270 or PC400 returns the part of the presentation space in the format specified in the Get Partial Presentation Space request.

If the format is CF_DSPTEXT, PC/3270 or PC400 returns the presentation space in the following format:

```
typedef struct tagEPS_CF_DSPTEXT
{
 unsigned Unused:12,                  // Unused
 unsigned fResponse:1,                // TRUE = DDE_REQUEST response
 unsigned fRelease:1,                 // TRUE = client frees the storage
 unsigned reserved:1,                 //      Reserved
 unsigned fAckReq:1,                  // TRUE = DDE_ACK requested
 int           cfFormat;              // Format data is rendered in
 unsigned uPSPosition;                // Start PS position
 unsigned uPSLength;                  // Length of the part of the PS
 unsigned uPSRows;                    // PS number of rows
 unsigned uPSCols;                    // PS number of columns
 unsigned uPSOffset;                  // Offset to the presentation space
 unsigned uFieldCount;                // Number of fields
 unsigned uFieldOffset;               // Offset to the field array
 unsigned char PSData[1];             // PS and Field list Array
} EPS_CF_DSPTEXT, FAR  lpEPS_CF_DSPTEXT;

typedef struct tagPSFIELDS
{
 unsigned char cAttribute;          // Attribute Character
 unsigned uFieldStart;              // Field start offset
 unsigned uFieldLength;             // Field Length
} PSFIELDS, FAR  lpPSFIELDS;
```

Note: The following examples show how to obtain long pointers to the PS and the PSFIELDS array.

```
lpps = (lp_EPS_CF_DSPTEXT) lpEPS_CF_DSPTEXT->PSData
       +  lpEPS_CF_DSPTEXT->uPSOffset;
lppsfields = (lpPSFIELDS) lpEPS_CF_DSPTEXT->PSData
                 +  lpEPS_CF_DSPTEXT->uFieldOffset;
```

If the format is CF_TEXT, PC/3270 or PC400 returns the part of the presentation space in the following format:

```
typedef struct tagEPS_CF_TEXT
{
 unsigned          Unused:12;          //      Unused
 unsigned          fResponse:1;        // TRUE = DDE_REQUEST response
  unsigned         fRelease:1;         //  TRUE  =  Client  frees  this  data
  unsigned         reserved:1;         //        Reserved
 unsigned          fAckReq:1;          // TRUE = Client returns DDE_ACK
 int               cfFormat;           // Format of the data
 unsigned char PSPOSITION[4];     // Start PS position
 unsigned char Tab1[1];           // Tab character
 unsigned char PSLENGTH[4];       // Length of the part of the PS
 unsigned char Tab2[1];           // Tab character
 unsigned char PSROWS[4];         // Number of rows in the Partial PS
 unsigned char Tab3[1];           // Tab character
 unsigned char PSCOLS[4];         // Number of columns in the PS
 unsigned char Tab4[1];           // Tab character
 unsigned char PS[1];             // PS
} EPS_CF_TEXT, FAR  lpEPS_CF_TEXT;
```

Following the PS in the buffer is the following additional structure of fields that compose the field list.

```
typedef struct tagFL_CF_TEXT
{
 unsigned char Tab5[1];           // Tab character
 unsigned char PSFldCount[4]; // Number of fields in the PS
 unsigned char Tab6[1];           // Tab character
  PS_FIELD         Field[1];         //  Field  List  Array
} FL_CF_TEXT, FAR  lpFL_CF_TEXT;

typedef struct tagPS_FIELD
{
 unsigned char FieldStart[4];
 unsigned char TabF1[1];
 unsigned char FieldLength[4];
 unsigned char TabF2[1];
 unsigned char Attribute;
 unsigned char TabF3[1[;
} PS_FIELD, FAR  lpPS_FIELD;
```

Note: The following examples show how to obtain long pointers to the PS and the PS_FIELD array.

```
lpps = lpEPS_CF_TEXT->PS;
lpps_field = (lpPS_FIELD) lpEPS_CF_TEXT->PS
             +  atoi(lpEPS_CF_TEXT->PSLENGTH)
              + ((atoi(lpEPS_CF_TEXT->PSROWS) - 1)   2) // CR/LF
              + 1 + 1 + 4 + 1;      // Tabs + size of field count
```

Get Presentation Space

PC/3270	*PC400*
X	X

The Get Presentation Space function returns the session presentation space to the client. The client sends the following command to get the presentation space.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aPS) );
```

where:

cfFormat Identifies the format for the presentation space. Valid values are CF_TEXT or CF_DSPTEXT. The structure of the presentation space, in these two formats, is shown below.

aPS Identifies presentation space atom as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the presentation space and a list of the fields that comprise the presentation space, or responds with one of these ACK messages containing an error code in the low order byte of the wStatus word:

WM_DDE_DATA(hData, aPS)
WM_DDE_ACK(wStatus, aPS)

If PC/3270 or PC400 cannot return the presentation space, the following status codes will be returned in the low order byte of the wStatus word:

Return Code	Explanation
4	The specified format is not valid.
9	A system error occurred.

Structure of the Presentation Space

PC/3270 or PC400 returns the presentation space in the format specified in the Get Presentation Space request.

If the format is CF_DSPTEXT, PC/3270 or PC400 returns the presentation space in the following format:

```
typedef struct tagPS_CF_DSPTEXT
{
 unsigned Unused:12,                  // Unused
 unsigned fResponse:1,                // TRUE = DDE_REQUEST response
 unsigned fRelease:1,                 // TRUE = client frees the storage
 unsigned reserved:1,                 //      Reserved
 unsigned fAckReq:1,                    // TRUE = DDE_ACK requested
 int          cfFormat;                 // Format data is rendered in
 unsigned uPSSize;                      // Size of the presentation space
 unsigned uPSRows;                       // PS number of rows
 unsigned uPSCols;                      // PS number of columns
 unsigned uPSOffset;                   // Offset to the presentation space
 unsigned uFieldCount;                 // Number of fields
 unsigned uFieldOffset;               // Offset to the field array
 unsigned char PSData[1];           // PS and Field list Array
} PS_CF_DSPTEXT, FAR  lpPS_CF_DSPTEXT;

typedef struct tagPSFIELDS
{
 unsigned char cAttribute;         // Attribute Character
 unsigned uFieldStart;               // Field start offset
 unsigned uFieldLength;              // Field Length
} PSFIELDS, FAR  lpPSFIELDS;
```

Note: The following examples show how to obtain long pointers to the PS and the PSFIELDS array.

```
   lpps = (lp_PS_CF_DSPTEXT) lpPS_CF_DSPTEXT->PSData
         +  lpPS_CF_DSPTEXT->uPSOffset;
   lppsfields = (lpPSFIELDS) lpPS_CF_DSPTEXT->PSData
                  +  lpPS_CF_DSPTEXT->uFieldOffset;
```

If the format is CF_TEXT, PC/3270 or PC400 returns the presentation space in the following format:

```
typedef struct tagPS_CF_TEXT
{
 unsigned          Unused:12;            //      Unused
 unsigned          fResponse:1;        // TRUE = DDE_REQUEST response
  unsigned         fRelease:1;         //  TRUE  =  Client  frees  this  data
  unsigned         reserved:1;         //       Reserved
 unsigned          fAckReq:1;            // TRUE = Client returns DDE_ACK
 int               cfFormat;             // Format of the data
 unsigned char PSSIZE[4];             // Size of the PS
 unsigned char Tab1[1];               // Tab character
 unsigned char PSROWS[4];              // Number of rows in the PS
 unsigned char Tab2[1];               // Tab character
 unsigned char PSCOLS[4];              // Number of Cols in the PS
 unsigned char Tab3[1];               // Tab character
 unsigned char PS[1];                 // PS
} PS_CF_TEXT, FAR  lpPS_CF_TEXT;
```

Following the PS in the buffer is the following additional structure of fields that compose the field list.

```
typedef struct tagFL_CF_TEXT
{
 unsigned char Tab4[1];              // Tab character
 unsigned char PSFldCount[4]; // Number of fields in the PS
 unsigned char Tab5[1];              // Tab character
  PS_FIELD          Field[1];          //  Field  List  Array
} FL_CF_TEXT, FAR  lpFL_CF_TEXT;

typedef struct tagPS_FIELD
{
 unsigned char FieldStart[4];
 unsigned char TabF1[1];
 unsigned char FieldLength[4];
 unsigned char TabF2[1];
 unsigned char Attribute;
 unsigned char TabF3[1];
} PS_FIELD, FAR  lpPS_FIELD;
```

Note: The following example shows how to obtain long pointers to the PS and the PS_FIELD array.

```
   lpps = lpPS_CF_TEXT->PS;
   lpps_field = (lpPS_FIELD) lpPS_CF_TEXT->PS
                  +  atoi(lpPS_CF_TEXT->PSSIZE)
                   + ((atoi(lpPS_CF_TEXT->PSROWS) - 1)   2) // CR/LF
                   + 1 + 1 + 4 + 1;      // Tabs + size of field count
```

Get Session Status

PC/3270	PC400
X	X

The Get Session Status function returns the status of the connected session. The client sends the following message to request session status:

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aSSTAT) );
```

where:

cfFormat Identifies the DDE format for the status information. The value used is CF_TEXT.

aSSTAT Identifies session status as the data item requested.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the session status in a DDE data message, or responds with one of these ACK messages containing status information:

WM_DDE_DATA(hData, aSSTAT)
WM_DDE_ACK(wStatus, aSSTAT)

If PC/3270 or PC400 cannot return the session status, the following status codes will be returned in the low order byte of the wStatus word:

Return Code	Explanation
6	The specified format is not valid.
9	A system error occurred.

Format of Status Information

PC/3270 or PC400 returns the session status as text in CF_TEXT format. The following fields are returned with the following possible values:

Fields	Returned values	Description
Status	Closed, Invisible, Maximized, Minimized, Normal	The window is in one of these states.
Usage	DDE, User	The session is connected in either a DDE session or a user session.
ScreenX	NN	Defines the horizontal size of the screen.
ScreenY	NN	Defines the vertical size of the screen.
CursorX	NN	Defines the horizontal position of the cursor. (0 ... ScreenX – 1)

Fields	Returned values	Description
CursorY	NN	Defines the vertical position of the cursor. (0 ... ScreenY – 1)
TrimRect Status	Closed, Moved, Sized	The current status of the trim rectangle.
Trim Rectangle X1	N	The top-left corner X position of the trim rectangle in character coordinates.
Trim Rectangle Y1	N	The top-left corner Y position of the trim rectangle in character coordinates.
Trim Rectangle X2	N	The lower-right corner X position of the trim rectangle in character coordinates.
Trim Rectangle Y2	N	The lower-right corner Y position of the trim rectangle in character coordinates.
Session Presentation Space Status	N	The current status of the presentation space. The following values are possible: 1. 0: The presentation space is unlocked. 2. 4: The presentation space is busy. 3. 5: The presentation space is locked.
Session Window Handle	XXXX	Window handle of the session.

Notes:

1. The status of each field is updated each time the status is requested.
2. A new field might be added in a future version of PC/3270 or PC400.

Get System Configuration

PC/3270	*PC400*
X	X

The Get System Configuration function returns the level of PC/3270 or PC400 support and other system-related values. Most of this information is for use by a service coordinator when a customer calls the IBM Support Center after receiving a system error.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aSYSCON) );
```

where:

cfFormat Identifies the DDE format for the data item requested. The value used is CF_TEXT.

aSYSCON
Identifies system configuration as the data item requested.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the system configuration data item in a DDE DATA message, or responds with one of these ACK messages containing status information:

WM_DDE_DATA(hData, aSYSCON)
WM_DDE_ACK(wStatus, aSYSCON)

If PC/3270 or PC400 cannot return the system configuration, a DDE ACK message will be returned with an error code in the low-order byte of the wStatus word:

WM_DDE_ACK(wStatus, aSYSCON)

Return Code	Explanation
9	A system error occurred.

Format of System Configuration information

PC/3270 or PC400 returns the system configuration as text in CF_TEXT format. The following fields are returned with the following possible values:

Fields	Returned values	Description
Version	N	The version of PC/3270 or PC400
Level	NN	The level of PC/3270 or PC400
Date	MMDDYY	The date of PC/3270 or PC400
System model base number and submodel number	XXXX	The following numbers are examples only: FC00 or FC01 = PC AT FC02 = PC XT Model 286 FC04 = PS/2 Model 50 FC05 = PS/2 Model 60 F800 or F801 = PS/2 Model 80 For more information, see *IBM Personal System/2 and Personal Computer BIOS Interface Technical Reference*.
Monitor Type	MONO, CGA, EGA, VGA, XGA	Type of the monitor
Country Code	NNNN	Country code used with 3270 or 5250 terminal session

Get System Formats

PC/3270	*PC400*
X	X

The Get System Formats function returns the list of Windows clipboard formats supported by PC/3270 or PC400. The client application sends the following message to retrieve the format list supported by PC/3270 or PC400:

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aFORMATS) );
```

where:

cfFormat Identifies the DDE format for the data item requested. The value used is CF_TEXT.

aFORMATS
Identifies formats as the data item requested.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the list of supported Windows clipboard formats in CF_TEXT format in a DDE DATA message.

```
WM_DDE_DATA(hData, aFORMATS)
```

The following Windows Clipboard formats are supported by PC/3270 or PC400:

```
CF_TEXT
CF_DSPTEXT
```

If PC/3270 or PC400 cannot return the formats data item, a DDE ACK message will be returned with an error code in the low-order byte of the wStatus word:

```
WM_DDE_ACK(wStatus, aFORMATS)
```

Return Code	Explanation
9	A system error occurred.

Get System Status

PC/3270	*PC400*
X	X

The Get System Status function returns the status of each 3270 or 5250 session that is available with the current PC/3270 or PC400 configuration. The client application sends the following message to retrieve the status data item:

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aSTATUS) );
```

where:

cfFormat Identifies the DDE format for the data item requested. The value used is CF_TEXT.

aSTATUS
Identifies status as the data item requested.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the status data item in CF_TEXT format in a DDE DATA message:

```
WM_DDE_DATA(hData, aSTATUS)
```

For each opened session, PC/3270 or PC400 returns a line of status information. Each line contains a series of fields with the following range of values:

Fields	Range of values	Description
Session ID	A, B, ..., Z	The short ID of the session.
Host Type	370, 400	The host system currently supported by PC/3270 or PC400.
Emulation Type	3270, 5250	The emulation type supported by PC/3270 or PC400.
Session Status	Closed, Invisible, Normal, Minimized, Maximized	The current status of the session's window.

If PC/3270 or PC400 cannot return the status data item, a DDE ACK message will be returned with an error code in the low-order byte of the wStatus word:

```
WM_DDE_ACK(wStatus, aSTATUS)
```

Return Code	Explanation
9	A system error occurred.

Get System SysItems

PC/3270	*PC400*
X	X

PC/3270 or PC400 supports the DDE system topic so that a client application can connect to the system topic and retrieve information about PC/3270 or PC400 and the status of the sessions that PC/3270 or PC400 is managing.

The Get System SysItems function returns the list of data items available in the PC/3270 or PC400 system topic. The client application sends the following message to get the system topic data items:

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aSYSITEMS) );
```

where:

cfFormat Identifies the DDE format for the data item requested. The value used is CF_TEXT.

aSYSITEMS
Identifies SysItems as the data item requested.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the list of system topic data items in CF_TEXT format in a DDE DATA message.

```
WM_DDE_DATA(hData, aSYSITEMS)
```

The following list of data items are supported by PC/3270 or PC400:

SysItems
Topics
Status
Formats
SysCon

If PC/3270 or PC400 cannot return the system data items, a DDE ACK message will be returned with an error code in the low-order byte of the wStatus word:

```
WM_DDE_ACK(wStatus, aSYSITEMS)
```

Return Code	Explanation
9	A system error occurred.

Get System Topics

PC/3270	*PC400*
X	X

The Get System Topics function returns the list of active DDE topics currently supported by PC/3270 or PC400. The client application sends the following message to the system topic to retrieve the list of topics that are currently active:

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aTOPICS) );
```

where:

cfFormat Identifies the DDE format for the data item requested. The value used is CF_TEXT.

aTOPICS Identifies topics as the data item requested.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the list of DDE topics in CF_TEXT format in a DDE DATA message.

```
WM_DDE_DATA(hData, aTOPICS)
```

The following list of topics are supported by PC/3270 or PC400:

System – System Topic
SessionA – Session A Topic[2]
⋮
SessionZ – Session Z Topic[2]

If PC/3270 or PC400 cannot return the list of topics, a DDE ACK message will be returned with an error code in the low-order byte of the wStatus word:

```
WM_DDE_ACK(wStatus, aTOPICS)
```

Return Code	Explanation
9	A system error occurred.

[2] The actual number of session topics supported is dependent on the number of sessions currently opened. The client program should always query the topics data item of the system topic to obtain the list of sessions currently opened.

Get Trim Rectangle

PC/3270	*PC400*
X	X

The Get Trim Rectangle function returns to the client the area of the presentation space that is within the current trim rectangle. The client sends the following message to receive the trim rectangle.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aTRIMRECT) );
```

where:

cfFormat Identifies the format for the trim rectangle.
Valid value is CF_TEXT.

aTRIMRECT
Identifies trim rectangle as the data item requested.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns trim rectangle in a DDE data message, or responds with one of these ACK messages:

WM_DDE_DATA(hData, aTRIMRECT)
WM_DDE_ACK(wStatus, aTRIMRECT)

If PC/3270 or PC400 cannot return the trim rectangle, the following status codes will be returned in the low-order byte of the wStatus word:

Return Code	Explanation
6	The specified format is not valid.
9	A system error occurred.

Initiate Session Conversation

PC/3270	*PC400*
X	X

The Initiate Session Conversation function connects a client application to an available session of PC/3270 or PC400. Once a session conversation has been established, the session is reserved for exclusive use by the client until the conversation is terminated.

The client application sends the following message to initiate a DDE conversation with a session:

```
SendMessage( -1,
             WM_DDE_INITIATE,
             hClientWnd,
              MAKELONG(aIBM327 32, aSessionN) );
```

where:

aIBM327032
Identifies the application atom. The string used to create atom aIBM3270 is “IBM3270”. In the PC400, the application atom is aIBM5250 and the string “IBM5250” is used to create it.

aSessionN
Identifies the topic atom. The string used to create atom aSessionN is either NULL or “Session” appended with the session ID “A”, “B”, ..., “Z”.

PC/3270 or PC400 Response

If a specific topic is selected and PC/3270 or PC400 can support a conversation with the client application, PC/3270 or PC400 acknowledges the INITIATE transaction with:

```
WM_DDE_ACK(aIBM327 32, aSessionN)
```

If a topic is not selected (aSessionN = NULL), PC/3270 or PC400 responds by acknowledging all topics that are currently available:

```
WM_DDE_ACK(aIBM327 32, aSystem)
WM_DDE_ACK(aIBM327 32, aSessionA)
⋮
WM_DDE_ACK(aIBM327 32, aSessionZ)
```

The client application selects the conversation it wishes to communicate with from the returned list of topics and terminates all other unwanted conversations.

Initiate Structured Field Conversation

PC/3270	*PC400*
X	X

The Initiate Structured Field Conversation function connects a client application and a host application. This allows the applications to send data to each other and to receive data from each other.

The client sends the following command to initiate a structured field conversation:

```
SendMessage( -1,
             WM_DDE_INITIATE,
             hClientWnd,
              MAKELONG(aIBM327 , aLUN_xxxx) );
```

Where:

aIBM3270
Identifies the application atom.

aLUN_xxxx
Identifies the topic atom. The string used to create atom aLUN_xxxx is "LU" appended with the session ID "A", "B", ..., "Z", appended with "_", and appended with the user-defined string of any length.

PC/3270 response

If PC/3270 can support a structured field conversation with the client application, it will return an acknowledgment message with the following parameter:

```
WM_DDE_ACK(aIBM327 , aLUN_xxxx)
```

Initiate System Conversation

PC/3270	*PC400*
X	X

The Initiate System Conversation function connects a client application to the system conversation. Only one client can be connected to the system conversation at a given time. The client sends the following command to initiate a system conversation:

```
SendMessage( -1,
             WM_DDE_INITIATE,
             hClientWnd,
              MAKELONG(aIBM327 32, aSystem) );
```

where:

aIBM327032
Identifies the application atom.

aSystem Identifies the topic atom.

PC/3270 or PC400 Response

If PC/3270 or PC400 can support a system topic conversation with the client application, it returns an acknowledgment message with the following parameters:

```
WM_DDE_ACK(aIBM327 32, aSystem)
```

Put Data to Presentation Space

PC/3270	PC400
X	X

The Put Data to Presentation Space function sends an ASCIIZ data string to be written into the host presentation space at the location specified by the calling parameter. The client sends the following message to the session to send the string.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
              MAKELONG(hData, aEPS) );
```

where:

hData

Identifies a handle to a Windows global memory object that contains the string to be sent to the session. The global memory object contains the following structure:

```
typedef struct tagPutString
{
 unsigned unused:13;                  //    unused
 unsigned fRelease:1;                // Session frees memory
 unsigned freserved:2;               //    reserved
 int          cfFormat;                 // Always CF_DSPTEXT
 unsigned uPSStart;                   // PS Position
 unsigned uEOFflag;                    // EOF effective switch
 char         szStringData[1];         // String Data
} PUTSTRING, FAR  lpPUTSTRING;
```

These values are valid at the uEOFflag field:

```
WC_EFFECTEOF       // The string is truncated at EOF.
WC_UNEFFECTEOF 1 // The string is not truncated at EOF.
```

aEPS Identifies the presentation space atom as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 receives the string data and sends them to the presentation space, and returns a positive ACK message.

If the presentation space does not accept the string data, PC/3270 or PC400 returns a negative ACK message containing the following status codes in the low-order byte of the wStatus word:

```
WM_DDE_ACK(wStatus, aEPS)
```

Return Code	Explanation
1	PS position is not valid.
2	Length is not valid.
3	The value of EOF flag is not valid.
5	Input to the target PS was inhibited.
6	The specified format is not valid.
7	The string was truncated (successful putting).
9	A system error occurred.

Search for String

PC/3270	*PC400*
X	X

This function allows a client application to examine the presentation space for a specified string in a specified area.

Note: The client must set the start PS position, search direction, a string to be searched, and EOF flag by using the Set Presentation Space Service Condition function before using this function.

The client sends the following message to search for the string.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aSTRING) );
```

where:

cfFormat Identifies the format for the search information.
Valid value is CF_DSPTEXT.

aSTRING Identifies the search data item.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the start position of the string in a DDE data message if the string was found in the specified area:

WM_DDE_DATA(hData, aSTRING)
WM_DDE_ACK(wStatus, aSTRING)

If PC/3270 or PC400 cannot return the start position of the string, the following status codes will be returned in the low-order byte of the wStatus word:

Return Code	Explanation
1	PS position is not valid or the string is too long.
2	The string cannot be found.
6	The specified format is not valid.
9	A system error occurred.

Structure of the Search Information

PC/3270 or PC400 returns the Search information in the following structure:

```
typedef struct tagSEARCH
{                                       //
 unsigned unused:12;                   //     unused
 unsigned fResponse:1;                 // TRUE = DDE_REQUEST response
 unsigned fRelease:1;                  // TRUE = Client frees this data
 unsigned reserved:1;                  //     reserved
 unsigned fAckReq:1;                     // TRUE = Client returns DDE_ACK
 int          cfFormat;                  // Format of Search data CF_DSPTEXT
 unsigned uFieldStart;                 // String start offset
}    SEARCH, far  lpSEARCH;
```

Send Keystrokes

PC/3270	PC400
X	X

The Send Keystrokes function sends keystrokes to the connected session. The client sends the following message to the session to send keystrokes.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
              MAKELONG(hData, aKEYS) );
```

where:

hData Identifies a handle to a Windows global memory object that contains the keystrokes to be sent to the session. The global memory object contains the following structure:

```
typedef struct tagKeystrokes
{
 unsigned unused:13;                  //    unused
 unsigned fRelease:1;                 // Session frees memory
 unsigned freserved:2;                //    reserved
 int          cfFormat;               // Always CF_DSPTEXT
 unsigned uTextType;                  // Type of keystrokes
 unsigned uRetryCount;                // Retry count 1 .. 16
 unsigned char szKeyData[1];          // Keystrokes
} KEYSTROKES, FAR  lpKEYSTROKES;
```

The following key text types are supported:

```
WC_PURETEXT       // Pure text, no AID, or included HLLAPI
                  //  commands
WC_HLLAPITEXT 1 // Text, including HLLAPI tokens
```

Note: If the keystrokes are pure text then specifying WC_PURETEXT will transfer the keystrokes to the host in the fastest possible manner. If WC_HLLAPITEXT is specified then the keystroke data can contain HLLAPI commands interspersed with the text.

aKEYS Identifies keystrokes as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 receives the keystrokes and sends them to the presentation space. If the presentation space does not accept the keystrokes, a reset is sent to the presentation space and the keystrokes are sent again. This procedure continues until the presentation space accepts the keystrokes or the retry count is reached. If PC/3270 or PC400 cannot send the keystrokes to the host, PC/3270 or PC400 returns a negative ACK message containing the following status codes in the low-order byte of the wStatus word. Otherwise, PC/3270 or PC400 returns a positive ACK message signalling the keystrokes have been sent.

```
WM_DDE_ACK(wStatus, aKEYS)
```

Return Code	Explanation
1	Retry count is not valid.
2	Type of key strokes is not valid.
6	The specified format is not valid.
9	A system error occurred.

Session Execute Macro

PC/3270	PC400
X	X

You can issue commands and macros with the DDE_EXECUTE function. The DDE_EXECUTE function passes command strings to PC/3270 or PC400. The command strings must conform to DDE specifications. Refer to *Microsoft Windows Software Development Kit Guide to Programming* for more information about the DDE command syntax.

The client sends the following command to issue a DDE_EXECUTE function.

```
PostMessage ( hServerWnd,
              WM_DDE_EXECUTE,
              hClientWnd,
               MAKELONG(NULL, hCommands) );
```

where:

hCommands
Identifies a handle to a Windows global memory object containing PC/3270 or PC400 commands. For a list of commands you can issue, see “Issuing Commands with the Session Execute Macro Function.”

PC/3270 or PC400 Response

If PC/3270 or PC400 can process the command string, PC/3270 or PC400 returns an ACK message containing positive status information to the client. If PC/3270 or PC400 cannot perform the command string, PC/3270 or PC400 returns an ACK message containing this error code in the low-order word of the wStatus word:

Return Code	Explanation
9	A system error occurred.

Issuing Commands with the Session Execute Macro Function

You can issue the following commands with the Session Execute Macro function:

WINDOW
KEYBOARD
SEND
RECEIVE
SENDKEY
WAIT

Use a separate DDE_EXECUTE message for each command.

Notes:

Enclose values that contain nonalphanumeric characters or blanks in double quotation marks ("*value value*").
To include a double quotation mark within a string, type it twice (for example, "This is a double quotation mark:""").
The maximum length for any command is 255 characters.

WINDOW Command

[WINDOW(action, "name")]

Performs window actions, where:

action = HIDE|RESTORE|MAXIMIZE|MINIMIZE|
SHOW|CNGNAME
name = LT name or Switch List Entry name

Note: *name* should be specified when CNGNAME is specified at *action*. If *name* is a NULL string, the default caption will be displayed.

KEYBOARD Command

[KEYBOARD(action)]

Enables or disables the session keyboard, including the mouse, where:

action = LOCK|UNLOCK

SEND Command

[SEND("pcfilename","hostfilename","options")]

Sends files to the host, where:

pcfilename = [path]filename[.ext]
hostfilename =
For VM system:
filename filetype[filemode]
For MVS system:
[']filename[(membername)][']
For CICS system:
filename or _
For OS/4 system:
library name filename member name

options includes any combination of the following file transfer options: MVS, VM, CICS, QUIET, OS/400, and emulation-specific transfer options, separated by spaces.

Refer to *Personal Communications/3270 Version 4.1 for Windows 95 Reference* or *Personal Communications/3270 Version 4.1 for Windows 95 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:
     filename  or  _
   For OS/4  system:
     library name  filename  member name
```

options includes any combination of the following file transfer options: MVS, VM, CICS, QUIET, OS/400, and emulation-specific transfer options, separated by spaces.

Refer to *Personal Communications/3270 Version 4.1 for Windows 95 Reference* or *Personal Communications/3270 Version 4.1 for Windows 95 Reference* for more information about the transfer options.

SENDKEY Command

[SENDKEY(token,token)]

Sends keystrokes to PC/3270 or PC400, where:

token = text string | command | macro macroname

Notes:

Text strings are enclosed in double quotation marks.
Macros are prefixed with “macro”.
The argument string for SENDKEY must be 255 characters or less in size.
The following commands are supported.

Table 130 (Page 1 of 3). SENDKEY Command List

Command Name	Token	PC/3270	PC400
Alternate Cursor	[alt cursor]	X	X
Alternate Viewing Mode	[alt view]	X	X
Attention	[sys attn]	X	X
Backspace	[backspace]	X	X
Back Tab	[backtab]	X	X
Backtab Word	[backtab word]	X	X
Character Advance	[character advance]	–	X
Character Backspace	[backspace valid]	–	X
Clear Screen	[clear]	X	X
Clicker	[click]	X	X
Color Blue	[blue]	X	–
Color Field Inherit	[field color]	X	–
Color Green	[green]	X	–
Color Pink	[pink]	X	–
Color Red	[red]	X	–
Color Turquoise	[turquoise]	X	–
Color White	[white]	X	–
Color Yellow	[yellow]	X	–
Cursor Blink	[cursor blink]	X	X
Cursor Down	[down]	X	X
Cursor Left	[left]	X	X
Cursor Right	[right]	X	X
Cursor Select	[cursor select]	X	X
Cursor Up	[up]	X	X
Delete Character	[delete char]	X	X
Delete Word	[delete word]	X	X
Device Cancel	[device cancel]	X	X
Dup Field	[dup]	X	X
Edit Clear	[edit-clear]	X	X
Edit Copy	[edit-copy]	X	X

Table 130 (Page 2 of 3). SENDKEY Command List

Command Name	Token	PC/3270	PC400
Edit Cut	[edit-cut]	X	X
Edit Paste	[edit-paste]	X	X
Edit Undo	[edit-undo]	X	X
End Field	[end field]	X	X
Enter	[enter]	X	X
Erase EOF	[erase eof]	X	X
Erase Field	[erase field]	X	–
Erase Input	[erase input]	X	X
Fast Cursor Down	[fast down]	X	X
Fast Cursor Left	[fast left]	X	X
Fast Cursor Right	[fast right]	X	X
Fast Cursor Up	[fast up]	X	X
Field Exit	[field exit]	–	X
Field Mark	[field mark]	X	X
Field +	[field +]	–	X
Field –	[field -]	–	X
Graphic Cursor	[+cr]	X	–
Help	[help]	X	X
Highlighting Field Inherit	[field hilight]	X	–
Highlighting Reverse	[reverse]	X	–
Highlighting Underscore	[underscore]	X	–
Home	[home]	X	X
Host Print	[host print]	X	–
Input	[input]	X	X
Input nondisplay	[input nd]	X	X
Insert Toggle	[insert]	X	X
Lower case	[to lower]	X	–
Mark Down	[mark down]	X	X
Mark Left	[mark left]	X	X
Mark Right	[mark right]	X	X
Mark Up	[mark up]	X	X
Move Mark Down	[move down]	X	X
Move Mark Left	[move left]	X	X
Move Mark Right	[move right]	X	X
Move Mark Up	[move up]	X	X
New Line	[newline]	X	X
Next Page	[page down]	–	X
Pause 1 second	[pause]	X	X
Previous Page	[page up]	–	X
Print Screen	[local copy]	X	X
Program Attention Key 1	[pa1]	X	–
Program Attention Key 2	[pa2]	X	–
Program Attention Key 3	[pa3]	X	–

Table 130 (Page 3 of 3). SENDKEY Command List

Command Name	Token	PC/3270	PC400
Program Function Key 1 ⋮ Program Function Key 24	[pf1] ⋮ [pf24]	X ⋮ X	X ⋮ X
Quit	[quit]	X	X
Reset	[reset]	X	X
Response Time Monitor	[rtm]	X	–
Roll Down	[roll down]	–	X
Roll Up	[roll up]	–	X
Rubout	[rubout]	X	X
Rule	[rule]	X	X
SO/SI Display	[so si]	X	X
SO/SI Generate	[so si generate]	–	X
System Request	[sys req]	X	X
Tab Field	[tab field]	X	X
Tab Word	[tab word]	X	X
Test	[test request]	–	X
Unmark	[unmark]	X	X
Upper case	[to upper]	X	–
Upper/Lower Change	[to other]	X	–
Wait for bind	[wait app]	X	X
Wait for System	[wait sys]	X	X
Wait transition	[wait trn]	X	X
Wait while input inh.	[wait inp inh]	X	X
Window Relocation 1 ⋮ Window Relocation 8	[view 1] ⋮ [view 8]	X ⋮ X	X ⋮ X

WAIT Command

[WAIT("[time out][wait condition]")]

Waits until the timeout expires or the wait condition the client specified occurs. For this command, the client has to set at least one option, where:

time out (optional)
: If the client sets a timeout value in the command statements, the following units are available in the wait statement.

  msec
  millisecond
  milliseconds
  sec
  second
  seconds
  minute
  minutes
  hour
  hours

wait condition (optional)

For the wait condition option, the client can select the following options:

while cursor at (cursor row, cursor column)
: While the cursor is at (cursor row, cursor column), it keeps waiting.

while “string”
: While the “string” is somewhere on the screen, it keeps waiting.

while “string” at (cursor row, cursor column)
: While the “string” is at (cursor row, cursor column) on the screen, it keeps waiting.

until cursor at (cursor row, cursor column)
: Until the cursor moves to (cursor row, cursor column), it keeps waiting.

until “string”
: Until the “string” is displayed somewhere on the screen, it keeps waiting.

until “string” at (cursor row, cursor column)
: Until the “string” is displayed at (cursor row, cursor column), it keeps waiting.

Examples:

1. To wait 10 seconds

   ```
   [WAIT("1  seconds")]
   ```

2. To wait while "ABCDEF" is displayed at (2,9) on the screen

   ```
   [WAIT("while ""ABCDEF"" at (2,9)")]
   ```

3. To wait until "ABCDEF" is displayed at (2,9) on the screen, or after 8 seconds

   ```
   [WAIT("8 seconds until ""ABCDEF"" at (2,9)")]
   ```

Set Cursor Position

PC/3270	PC400
X	X

The Set Cursor Position function allows the client application to set the cursor position in the session window.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
              MAKELONG(hData, aSETCURSOR) );
```

where:

hData

Identifies a handle to a Windows global memory object that contains the cursor positioning information in the following structure:

```
typedef struct tagSETCURSOR
{
 unsigned unused:13;                //    unused
 unsigned fRelease:1;               // TRUE = Session frees memory
 unsigned freserved:2;              //    reserved
 int          cfFormat;               // Always CF_DSPTEXT
 unsigned uSetCursorType;           // Cursor Set Type
 unsigned uSetCursor1;              // Cursor Row or PS Offset
 unsigned uSetCursor2;              // Cursor Col
} SETCURSOR, FAR  lpSETCURSOR;
```

PC/3270 or PC400 supports two ways to set the cursor position:

PS Offset (uSetCursorType = 0)
Row/Column number (uSetCursorType = 1)

The application specifies which method by setting the uSetCursorType field to the appropriate value, followed by setting the two other fields uSetCursor1 and uSetCursor2 to their appropriate values as follows:

uSetCursorType = 0 offset

- uSetCursor1: 0 ... (PSsize – 1)

uSetCursorType = 1 row/col

- uSetCursor1: 0 ... (PSrows – 1)
- uSetCursor2: 0 ... (PScols – 1)

aSETCURSOR

Identifies cursor position as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 receives the cursor information and moves the cursor to the specified position in the PS. If the cursor is positioned successfully, PC/3270 or PC400 returns a positive ACK message to the client application. Otherwise, a negative ACK message is returned with the following error codes in the low-order byte of the wStatus word.

```
WM_DDE_ACK(wStatus, aSETCURSOR)
```

Return Code	Explanation
1	Cursor set type is not valid. Must be 0 or 1.
2	Cursor PS offset is not valid. Must be 0 ... (PSsize – 1).
3	Cursor row value is not valid. Must be 0 ... (PSrows – 1).
4	Cursor column value is not valid. Must be 0 ... (PScols – 1).
6	The specified format is not valid.
9	A system error occurred.

Set Mouse Intercept Condition

PC/3270	*PC400*
X	X

This function specifies the mouse input to be intercepted. The client sends the following command to set the mouse event to be intercepted.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
              MAKELONG(hData, aMOUSE) );
```

where:

hData Identifies a handle to a Windows global memory object that specifies the condition of intercepting the mouse input.

If the format is CF_TEXT, the client program sends the condition in the following structure:

```
typedef struct tagSETMOUSE_CF_TEXT
{
 unsigned          unused:12,                  //
 unsigned          fRelease:1,                //
 unsigned          fReserved:3;               //
 int               cfFormat;                   // Always CF_TEXT
 unsigned char Condition[1]                  //
} SETMOUSE_CF_TEXT, FAR  lpSETMOUSE_CF_TEXT;
```

The following table shows the parameters' values:

Parameter Name	Meaning	Value
Condition	Condition of intercepting the mouse input	A string terminated with '\0', consists of the constants defined as follows in any order: **L** Enable intercepting the left button **l** Disable intercepting the left button **R** Enable intercepting the right button **r** Disable intercepting the right button **M** Enable intercepting the middle button **m** Disable intercepting the middle button **S** Enable intercepting a single click **s** Disable intercepting a single click **D** Enable intercepting a double click **d** Disable intercepting a double click **T** Retrieve the pointed string **t** Do not retrieve the pointed string

If the format is CF_DSPTEXT, the client program sends the condition in the following structure:

```
typedef struct tagSETMOUSE_CF_DSPTEXT
{
 unsigned        unused:12,              //
 unsigned        fRelease:1,             //
 unsigned        fReserved:3;            //
 int             cfFormat;               // Always CF_DSPTEXT
 BOOL              bLeftButton;          //
 BOOL              bRightButton;         //
 BOOL              bMiddleButton;        //
 BOOL              bSingleClick;         //
 BOOL              bDoubleClick;         //
 BOOL              bRetrieveString;      //
} SETMOUSE_CF_DSPTEXT, FAR  lpSETMOUSE_CF_DSPTEXT;
```

The following table shows the values in the parameters:

Parameter Name	Meaning	Value	
bLeftButton	Enable or disable interception of the left mouse button	**True** **False**	Enable intercepting the left button Disable intercepting the left button
bRightButton	Enable or disable interception of the right mouse button	**True** **False**	Enable intercepting the right button Disable intercepting the right button
bMiddleButton	Enable or disable interception of the middle mouse button	**True** **False**	Enable intercepting the middle button Disable intercepting the middle button
bSingleClick	Enable or disable interception of the single click	**True** **False**	Enable intercepting the single click Disable intercepting the single click
bDoubleClick	Enable or disable interception of the double click	**True** **False**	Enable intercepting the double click Disable intercepting the double click
bRetrieveString	Retrieve or do not retrieve the pointed string	**True** **False**	Retrieve the pointed string Do not retrieve the pointed string

aMOUSE Identifies Mouse atom as the item.

PC/3270 or PC400 Response

When receiving the Set Mouse Intercept Condition request, PC/3270 or PC400 returns an ACK message if it can set the intercept condition to the specified status. Otherwise, a negative ACK message is returned to the client with one of the following return codes in the low-order byte of the wStatus field:

WM_DDE_ACK(wStatus, aMOUSE)

Return Code	Explanation
2	A character in Condition parameter is not valid.
6	The specified format is not valid.
9	A system error occurred.

Set Presentation Space Service Condition

PC/3270	PC400
X	X

The Set Presentation Space Service Condition function sets the condition for using the following functions:

Get Partial Presentation Space
Find Field
Search for String

The client application sets the condition by calling this function such as:

Start PS position
PS length
EOF flag
Search direction
ASCIIZ string

The client must specify the Set Presentation Space Service Condition function before invoking the functions listed above. The conditions set by this function remain in effect until the next Set Presentation Space Service Condition function is called. The client sends the following message to set the condition:

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
              MAKELONG(hData, aEPSCOND) );
```

where:

hData

Identifies a handle to a Windows global memory object containing:

```
typedef struct tagPSSERVCOND
{
 unsigned     unused:13,                 //
 unsigned     fRelease:1,                //
 unsigned     fReserved:2;               //
 int          cfFormat;                   // Always CF_DSPTEXT
 unsigned     uPSStart;                  // PS Position
 unsigned     uPSLength;                  // Length of PS
 unsigned     uSearchDir;                // Direction for search
 unsigned     uEOFflag;                   // EOF effective switch
 char         szTargetString[1];        // Target String
} PSSERVCOND, FAR  lpPSSERVCOND;
```

The following values are valid at the uSearchDir field:

```
WC_SRCHFRWD      // Search forward.
WC_SRCHBKWD     1 // Search backward.
```

The following values are valid at the uEOFflag field:

```
WC_UNEFFECTEOF   // The string is not truncated at EOF.
WC_EFFECTEOF     1 // The string is truncated at EOF.
```

aEPSCOND

Identifies the item for the Set Presentation Space Service Condition function.

PC/3270 or PC400 Response

If PC/3270 or PC400 can perform the Set Presentation Space Service Condition function, then PC/3270 or PC400 returns an ACK message:

WM_DDE_ACK(wStatus, aEPSCOND)

If PC/3270 or PC400 cannot perform the Set Presentation Space Service Condition function, then PC/3270 or PC400 returns a negative ACK message containing the following return codes in the low-order byte of wStatus:

Return Code	Explanation
1	PS position is not valid.
2	Length is not valid.
3	The value of EOF flag is not valid.
4	The value of Search Direction is not valid.
6	The specified format is not valid.
9	A system error occurred.

Set Session Advise Condition

PC/3270	PC400
X	X

This function sets the condition for the DDE_ADVISE of the Start Session Advise function. The client can specify a search string and a region of the screen. When the advise condition is met, the server notifies the client of the condition according to the options specified by the Start Session Advise function.

Note: The client must specify the Set Session Advise Condition function before invoking Start Session Advise. If the advise condition is set after the Start Session Advise function is started, the advise condition will be ignored and the client will receive a negative ACK message. See “Start Session Advise” on page 421 for more information about starting the advise.

The client sends the following message to set the advise condition.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
              MAKELONG(hData, aPSCOND) );
```

where:

hData Identifies a handle to a Windows global memory object containing:

```
typedef struct tagSEARCHDATA
{
 unsigned     unused:13,                  //
 unsigned     fRelease:1,                 //
 unsigned     fReserved:2;                //
 int          cfFormat;                    // Always CF_DSPTEXT
 WORD           uPSStart;                    // PS Position of string
 WORD           uPSLength                    // Length of String
 BOOL         bCaseSensitive;             // Case Sensitive TRUE=YES
 char         SearchString[1];           //  Search  String
} SEARCHDATA, FAR  lpSEARCHDATA;
```

aPSCOND

Identifies the item for the Set Session Advise Condition function.

PC/3270 or PC400 Response

If PC/3270 or PC400 can perform the Set Session Advise Condition function, PC/3270 or PC400 returns this ACK message:

WM_DDE_ACK(wStatus, aPSCOND)

If PC/3270 or PC400 cannot perform the Set Session Advise Condition function, then PC/3270 or PC400 returns an negative ACK message containing the following return codes in the low-order byte of wStatus:

Return Code	Explanation
1	Advise is already active.
2	Advise condition is already active.
3	PS position is not valid.
4	String length is not valid.
6	The specified format is not valid.
9	A system error occurred.

Set Structured Field Service Condition

PC/3270	PC400
X	X

The Set Structured Field Service Condition function passes the Query Reply data provided by the client application.

Note: The client must call the Set Structured Field Service Condition function before invoking the Start Read SF function or the Write SF function.

The client sends the following message to set the condition.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
              MAKELONG(hData, aSFCOND) );
```

where:

hData Identifies a handle to a Windows global memory object containing:

```
typedef struct tagSFSERVCOND
{
 unsigned     unused:12,              //
 unsigned     fRelease:1,            //
 unsigned     fReserved:3;           //
 int          cfFormat;               // Always CF_DSPTEXT
 WORD           uBufferSize;            // Buffer size of Read SF
 WORD           uQRLength;              // Length of Query Reply data
 char         szQueryReply[1];       // Query Reply data
} SFSERVCOND, FAR  lpSFSERVCOND;
```

aSFCOND

Identifies the item for the Set Structured Field Service Condition function.

PC/3270 response

PC/3270 checks the Query Reply ID and Type (not DOID) and the length. If they are valid, then PC/3270 returns an ACK message:

```
WM_DDE_ACK(wStatus, aSFCOND)
```

If PC/3270 cannot perform the Set Structured Field Service Condition function, then PC/3270 returns a negative ACK message containing the following return codes in the low-order byte of wStatus:

Return Code	Explanation
1	PS SF ID is not valid.
2	Length is not valid.
3	One DDM base type is already connected to this session.
4	Structured Field Service Condition is already set.
6	The specified format is not valid.
9	A system error occurred.

Start Close Intercept

PC/3270	PC400
X	X

The Start Close Intercept function allows a client application to intercept close requests generated when a user selects the close option from the emulator session window. This function intercepts the close request and discards it until the Stop Close Intercept function is requested. After using this function, the client receives DATA messages notifying it that close requests occurred (CLOSE).

The client sends the following command to begin a session advise.

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
              MAKELONG(hOptions, aCLOSE) );
```

where:

hOptions Is a handle to a Windows global memory object, with the following structure:

```
typedef struct tagOPTIONS
{
 unsigned reserved:14;              //     reserved
 unsigned fDeferUpd:1;               // Send notification only
 unsigned fAckReq:1;                  // Client will ACK all notices
 WORD         cfFormat;                    // Clipboard format to use
}    OPTIONS, FAR  lpOPTIONS;
```

If the value of fDeferUpd is 1, DDE Data messages will be sent to the client application with the hData set to NULL. The client must then issue a DDE REQUEST to request the data item.

If the value of fAckReq is 1, the server does not notify the client of further close requests until the server receives an ACK message from the client in response to any previous notification.

The cfFormat field specifies the format to send the close request. (Must be CF_DSPTEXT.)

aCLOSE Identified close intercept as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 receives the Start Close Intercept and returns an ACK message if it can start the intercept. Otherwise a negative ACK message will be returned to the client with the following return codes in the low-order byte of the wStatus field:

```
WM_DDE_ACK(wStatus, aCLOSE)
```

Return Code	Explanation
1	Close Intercept is already working.
6	The specified format is not valid.
9	A system error occurred.

Once the intercept starts, the client receives DATA messages notifying it that the close request is intercepted:

WM_DDE_DATA(hData, aCLOSE)

where:

hData Identifies a handle to a Windows global memory object containing:

```
typedef struct tagCLOSEREQ
}
 unsigned     unused:12,            //     unused
 unsigned     fResponse:1,          // TRUE = DD_REQUEST response
 unsigned     fRelease:1,           // TRUE = Client releases memory
 unsigned     reserved:1,           //     reserved
 unsigned     fAckReq:1,            // TRUE = DDE_ACK is required
  int         cfFormat;             //  Always  CF_DSPTEXT
 WORD            uCloseReqCount;    // Counter of the Close Requests
} CLOSEREQ, FAR  lpCLOSEREQ;
```

The DATA messages continue until a Stop Close Intercept message is sent to PC/3270 or PC400.

Start Keystroke Intercept

PC/3270	PC400
X	X

The Start Keystroke Intercept function allows a client application to filter any keystrokes sent to a session by a terminal operator. After a call to this function, keystrokes are intercepted, and the client receives them (KEYS).

The client sends the following command to begin intercept.

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
              MAKELONG(hOptions, aKEYS) );
```

where:

hOptions Is a handle to a Windows global memory object, with the following structure:

```
typedef struct tagOPTIONS
}
 unsigned reserved:14;              // Reserved
 unsigned fDeferUpd:1;               // Send notification only
 unsigned fAckReq:1;                  // Client will ACK all notices
 WORD         cfFormat;                    // Clipboard format to use
}    OPTIONS, FAR  lpOPTIONS;
```

If the value of fDeferUpd is 1, DDE Data messages are sent to the client application with the hData set to NULL. The client then issues a DDE REQUEST to request the data item.

If the value of fAckReq is 1, the server does not notify the client of further keystrokes until the server receives an ACK message from the client in response to any previous keystrokes notification.

The cfFormat field specifies the format to send the keystrokes when the keystroke is sent by a terminal operator. (Must be CF_DSPTEXT.)

aKEYS Identified keystrokes as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 receives the Start Keystroke Intercept and returns an ACK message if it can start the intercept. Otherwise a negative ACK message is returned to the client with the following return codes in the low-order byte of the wStatus field:

```
WM_DDE_ACK(wStatus, aKEYS)
```

Return Code	Explanation
1	Keystroke Intercept is already started.
6	The specified format is not valid.
9	A system error occurred.

Once the intercept has started, the client receives DATA messages notifying it that the keystroke is intercepted:

```
WM_DDE_DATA(hData, aKEYS)
```

The DATA messages continue until a Stop Keystroke Intercept message is sent to PC/3270 or PC400. The format of the data item will be the same format as if the client requested the data item via a DDE_REQUEST.

Start Mouse Input Intercept

PC/3270	*PC400*
X	X

The Start Mouse Input Intercept function allows a client application to intercept mouse input when a terminal operator press the mouse button on emulator session window. After calling this function, the client receives DATA messages that include the PS position where mouse input occurred.

The client sends the following command to begin to intercept the mouse input.

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
              MAKELONG(hOptions, aMOUSE) );
```

where:

hOptions Is a handle to a Windows global memory object, with the following structure:

```
typedef struct tagOPTIONS
{
 unsigned reserved:14;          // Reserved
 unsigned fDeferUpd:1;           // Send notification only
 unsigned fAckReq:1;             // Client will ACK all notices
 WORD        cfFormat;               // Clipboard format to use
}   OPTIONS, FAR  lpOPTIONS;
```

If the value of fDeferUpd is 1, DDE Data messages will be sent to the client application with the hData set to NULL. The client must then issue a DDE REQUEST to request the data item.

If the value of fAckReq is 1, the server does not notify the client of further structured field data until the server receives an ACK message from the client in response to any previous notification.

The cfFormat field specifies the format to send the data item has been updated.

aMOUSE Identified MOUSE as the item.

PC/3270 or PC400 response

PC/3270 or PC400 receives the Start Mouse Input Intercept and returns an ACK message if it can start this function. Otherwise a negative ACK message is returned to the client with the following return codes in the low-order byte of the wStatus field:

WM_DDE_ACK(wStatus, aMOUSE)

Return Code	Explanation
1	Mouse Input Intercept has been already started.
6	The specified format is not valid.
9	A system error occurred.

Once the Mouse Input Intercept starts, the client receives DATA messages of the structured field:

WM_DDE_DATA(hData, aMOUSE)

where:

hData

If the format is CF_TEXT, PC/3270 or PC400 returns the mouse input information in the following format:

```
typedef struct tagMOUSE_CF_TEXT
{
 unsigned          unused:12,          //     Unused
  unsigned         fRespons:1,        // TRUE = DDE_REQUEST response
  unsigned         fRelease:1,        // TRUE = Client frees this data
  unsigned         reserved:1,        //       Reserved
 unsigned          fAckReq:1;           // TRUE = Client returns DDE_ACK
 int               cfFormat;            // CF_TEXT
 unsigned char PSPos[4];              // PS position
 unsigned char Tab1[1];               // TAB character
 unsigned char PSRowPos[4];           // PS row position
 unsigned char Tab2[1];               // TAB character
 unsigned char PSColPos[4];           // PS columns position
 unsigned char Tab3[1];               // TAB character
 unsigned char PSSize[4];             // Size of the PS
 unsigned char Tab4[1];               // TAB character
 unsigned char PSRows[4];             // PS number of rows
 unsigned char Tab5[1];               // TAB character
 unsigned char PSCols[4];             // PS number of columns
 unsigned char Tab6[1];               // TAB character
 unsigned char ButtonType[1];         // Pressed button type
 unsigned char Tab7[1];               // TAB character
 unsigned char ClickType[1];          // Click type
 unsigned char Tab8[1];               // TAB character
 unsigned char ClickString[1];        // Retrieved string
} MOUSE_CF_TEXT, FAR  lpMOUSE_CF_TEXT;
```

The following table shows the values in the parameters:

Parameter Name	Meaning	Value
PSPos	PS offset of the position where mouse was clicked	0 ... (PSSize – 1)
PSRowPos	Row number of the position where mouse was clicked	0 ... (PSRows – 1)
PSColPos	Column number of the position where mouse was clicked	0 ... (PSCols – 1)
PSSize	Presentation space size	
PSRows	Number of presentation space rows	
PSCols	Number of presentation space columns	
ButtonType	Type of clicked mouse button	**L** Left button **M** Middle button **R** Right button
ClickType	Type of clicking	**S** Single click **D** Double click
ClickString	Retrieved string to which the mouse pointed	A character string terminated with a '\0'
Tab1–8	A tab character for delimiter	'\t'

If the format is CF_DSPTEXT, PC/3270 or PC400 returns the mouse input information in the following format:

```
typedef struct tagMOUSE_CF_DSPTEXT
{
 unsigned          unused:12,            //     Unused
 unsigned          fRespons:1,           // TRUE = DDE_REQUEST response
 unsigned          fRelease:1,           // TRUE = client frees the storage
 unsigned          reserved:1,           //     Reserved
 unsigned          fAckReq:1;            // TRUE = client returns DDE_ACK
 int               cfFormat;             // CF_DSPTEXT
 unsigned          uPSPos;               // PS position
 unsigned          uPSRowPos;            // PS row position
 unsigned          uPSColPos;            // PS column position
 unsigned          uPSSize;              // Size of the presentation space
 unsigned          uPSRows;              // PS number of rows
 unsigned          uPSCols;              // PS number of columns
 unsigned          uButtonType;          // Pressed button type
 unsigned          uClickType;           // Click type
 unsigned char szClickString[1];   // Retrieved string
} MOUSE_CF_DSPTEXT, FAR  lpMOUSE_CF_DSPTEXT;
```

The following table shows the values in the parameters:

Parameter Name	Meaning	Value
uPSPos	PS offset of the position where mouse was clicked	0 ... (uPSSize – 1)
uPSRowPos	Row number of the position where mouse was clicked	0 ... (uPSRows – 1)
uPSColPos	Column number of the position where mouse was clicked	0 ... (uPSCols – 1)
uPSSize	Size of the presentation space	
uPSRows	Number of rows of the presentation space	
uPSCols	Number of columns of the presentation space	
uButtonType	Type of the clicked mouse button	**0x0001** Left button **0x0002** Middle button **0x0003** Right button
uClickType	Type of clicking	**0x0001** Single click **0x0002** Double click
szClickString	Retrieved string to which the mouse pointed	A character string terminated with a '\0'

The DATA messages continue until a Stop Mouse Input Intercept message is sent to PC/3270 or PC400.

Start Read SF

PC/3270	*PC400*
X	X

The Start Read SF function allows a client application to read structured field data from the host application. After using this function, the client receives DATA messages notifying it that close requests occurred.

Note: Before using this function, the client must call the Set Structured Field Service Condition function to pass the Query Reply data to the server.

The client sends the following command to begin a Read SF.

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
              MAKELONG(hOptions, aSF) );
```

where:

hOptions Is a handle to a Windows global memory object, with the following structure:

```
typedef struct tagOPTIONS
{
 unsigned reserved:14;          // Reserved
 unsigned fDeferUpd:1;           // Send notification only
 unsigned fAckReq:1;              // Client will ACK all notices
 WORD        cfFormat;                 // Clipboard format to use
}   OPTIONS, FAR  lpOPTIONS;
```

If the value of fDeferUpd is 1, DDE Data messages will be sent to the client application with the hData set to NULL. The client must then issue a DDE REQUEST to request the data item.

If the value of fAckReq is 1, the server does not notify the client of further structured field data until the server receives an ACK message from the client in response to any previous notification.

The cfFormat field specifies the format to send the structured field data. (It must be CF_DSPTEXT.)

aSF Identified structured field as the item.

PC/3270 response

PC/3270 receives the Start Read SF and returns an ACK message if it can start the Read SF. Otherwise a negative ACK message will be returned to the client with the following return codes in the low-order byte of the wStatus field:

WM_DDE_ACK(wStatus, aSF)

Return Code	Explanation
1	Read SF is already started.
3	No prior Set Structured Field Service Condition function was called.
6	The specified format is not valid.
9	A system error occurred.

Once the Read SF has started, the client receives DATA messages of the structured field:

WM_DDE_DATA(hData, aSF)

where:

hData Identifies a handle to a Windows global memory object containing:

```
typedef struct tagSFDATA
{
 unsigned     unused:12,                //
 unsigned     fRelease:1,              //
 unsigned     fReserved:3;             //
 int          cfFormat;                 // Always CF_DSPTEXT
 WORD           uSFLength;                  // Length of SF data
 char         szSFData[1];              // SF data
} SFDATA, FAR  lpSFDATA;
```

The DATA messages continue until a Stop Read SF message is sent to PC/3270.

Start Session Advise

PC/3270	PC400
X	X

The Start Session Advise function establishes a link between the PC/3270 or PC400 session and the client. This lets the client receive updates of the presentation space (PS), the operator information area (OIA), or the trim rectangle (TRIMRECT) when the data item is updated.

Note: If the client application needs conditional notification when the presentation space is updated, set an advise condition prior to invoking the advise function for the presentation space. See "Set Session Advise Condition" on page 409.

The client sends the following command to begin a session advise.

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
              MAKELONG(hOptions, aItem) );
```

where:

hOptions Is a handle to a Windows global memory object. This is the structure:

```
typedef struct tagOPTIONS
{
 unsigned reserved:14;              // Reserved
 unsigned fDeferUpd:1;               // Send notification only
 unsigned fAckReq:1;                  // Client will ACK all notices
 WORD         cfFormat;                    // Clipboard format to use
}    OPTIONS, FAR  lpOPTIONS;
```

If the value of fDeferUpd is 1, DDE Data messages are sent to the client application with the hData set to NULL. The client must then issue a DDE REQUEST to request the data item.

If the value of fAckReq is 1, the server does not notify the client of further changes to the data item until the server receives an ACK message from the client in response to any previous update notification.

The cfFormat field specifies the format to send the data item when the item has been updated.

aItem Specifies the item of information being requested; in this case, the value can be PS, OIA, or TRIMRECT.

PC/3270 or PC400 Response

PC/3270 or PC400 receives the Start Session Advise and returns an ACK message if it can start the advise. Otherwise, a negative ACK message will be returned to the client with the following return codes in the low-order byte of the wStatus field:

WM_DDE_ACK(wStatus, aItem)

Return Code	Explanation
1	Advise already active for data item.
6	Advise parameter not valid.
9	A system error occurred.

Once the advise has started, the client receives DATA messages notifying it that the data item (PS, OIA, or TRIMRECT) has changed:

WM_DDE_DATA(hData, aItem)

The DATA messages continue until a Stop Session Advise message is sent to PC/3270 or PC400. The format of the data item will be the same format as if the client requested the data item via a DDE_REQUEST.

Stop Close Intercept

PC/3270	*PC400*
X	X

The Stop Close Intercept function ends a client application's ability to intercept close request. The client sends the following command to perform the Stop Close Intercept function.

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
              MAKELONG(NULL, aCLOSE) );
```

where:

aCLOSE Identified close intercept as the item.

PC/3270 or PC400 Response

If PC/3270 or PC400 can perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aCLOSE)
```

If PC/3270 or PC400 cannot perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing negative status information and the following return codes in the low-order byte of the wStatus word:

Return Code	Explanation
1	Advise has not started yet.
9	A system error occurred.

Stop Keystroke Intercept

PC/3270	*PC400*
X	X

The Stop Keystroke Intercept function ends a client application's ability to intercept keystrokes. The client sends the following command to perform the Stop Keystroke Intercept function.

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
              MAKELONG(NULL, aKEYS) );
```

where:

aKEYS Identified keystrokes as the item.

PC/3270 or PC400 Response

If PC/3270 or PC400 can perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aKEYS)
```

If PC/3270 or PC400 cannot perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing negative status information and the following return codes in the low-order byte of the wStatus word.

Return Code	Explanation
1	Advise has not started yet.
9	A system error occurred.

Stop Mouse Input Intercept

PC/3270	*PC400*
X	X

The Stop Mouse Input Intercept function ends a client application's ability to intercept mouse input.

The client sends the following command to perform the Stop Mouse Input Intercept function.

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
              MAKELONG(NULL, aMOUSE) );
```

where:

aMOUSE Identified MOUSE as the item.

PC/3270 or PC400 Response

If PC/3270 or PC400 can perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aMOUSE)
```

If PC/3270 or PC400 cannot perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing negative status information and the following return codes in the low-order byte of the wStatus word.

Return Code	Explanation
1	Advise has not started yet.
9	A system error occurred.

Stop Read SF

PC/3270	*PC400*
X	X

The Stop Read SF function ends a client application's ability to read structured field data.

The client sends the following command to perform the Stop Read SF function.

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
              MAKELONG(NULL, aSF) );
```

where:

aSF Identified structured field as the item.

PC/3270 response

If PC/3270 can perform the DDE_UNADVISE, PC/3270 returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aSF)
```

If PC/3270 cannot perform the DDE_UNADVISE, PC/3270 returns an ACK message containing negative status information and the following return codes in the low-order byte of the wStatus word.

Return Code	Explanation
1	Advise has not started yet.
9	A system error occurred.

Stop Session Advise

PC/3270	*PC400*
X	X

The Stop Session Advise function disconnects a link between PC/3270 or PC400 and the client. The client sends the following command to perform the Stop Session Advise function.

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
              MAKELONG(NULL, aItem) );
```

where:

aItem Specifies the item of information being requested; in this case, the value can be PS, OIA, TRIMRECT, or NULL.

If the value of *aItem* is NULL, then the client has requested termination of all active notifications for the conversation.

PC/3270 or PC400 Response

If PC/3270 or PC400 can perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aItem)
```

If PC/3270 or PC400 cannot perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing negative status information and the following return codes in the low-order byte of the wStatus word.

Return Code	Explanation
1	Advise has not started yet.
9	A system error occurred.

Terminate Session Conversation

PC/3270	*PC400*
X	X

The Terminate Session Conversation function disconnects the client from the PC/3270 or PC400 session the client has previously started a conversation with.

The client sends the following command to terminate a session conversation.

```
SendMessage( hServerWnd,
             WM_DDE_TERMINATE,
             hClientWnd,
              MAKELONG(NULL, NULL) );
```

PC/3270 or PC400 Response

PC/3270 or PC400 acknowledges the terminate command with a terminate message:

WM_DDE_TERMINATE

Terminate Structured Field Conversation

PC/3270	*PC400*
X	X

The Terminate Structured Field Conversation function disconnects the client from a structured field conversation.

The client sends the following command to terminate a structured field conversation.

```
SendMessage( hServerWnd,
             WM_DDE_TERMINATE,
             hClientWnd,
              MAKELONG(NULL, NULL) );
```

PC/3270 response

PC/3270 acknowledges the terminate command with a terminate message:

WM_DDE_TERMINATE

Terminate System Conversation

PC/3270	*PC400*
X	X

This disconnects the client from a system conversation.

The client sends the following command to terminate a system conversation.

```
SendMessage( hServerWnd,
             WM_DDE_TERMINATE,
             hClientWnd,
              MAKELONG(NULL, NULL) );
```

PC/3270 or PC400 Response

PC/3270 or PC400 acknowledges the terminate command with this message:

WM_DDE_TERMINATE

When the user closes a PC/3270 or PC400 session, any global memory blocks that were allocated by PC/3270 or PC400 will be freed by Windows. This can cause problems for the client if the client retains any of these global memory objects for long periods of time. If the client application needs to keep the information in a global memory item for a long period of time, it is suggested that the client make a copy of global memory item into a global memory item the client application owns.

Write SF

PC/3270	*PC400*
X	X

The Write SF function allows a client application to write structured field data to the host application.

Note: The client must call the Set Structured Field Service Condition function before invoking the Write SF function.

The client sends the following message to write structured field data.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
              MAKELONG(hData, aSF) );
```

where:

hData Identifies a handle to a Windows global memory object containing:

```
typedef struct tagWRITESF
{
 unsigned     unused:12,                //
 unsigned     fRelease:1,              //
 unsigned     fReserved:3;              //
 int          cfFormat;                 // Always CF_DSPTEXT
 WORD           uSFLength;                    // Length of SF data
 char         Work[8];                  // Work area
 char         szSFData[1];              // SF data
} WRITESF, FAR  lpWRITESF;
```

aSF Identified structured field as the item.

PC/3270 response

PC/3270 receives structured field data and sends them to the host application. If the data transmission completes successfully, then PC/3270 returns an ACK message:

```
WM_DDE_ACK(wStatus, aSF)
```

Otherwise PC/3270 returns an negative ACK message containing the following return codes in the low-order byte of wStatus:

Return Code	Explanation
2	Length is not valid.
6	The specified format is not valid.
9	A system error occurred.

Summary of DDE Functions in a 16-Bit Envionment

The following table lists the DDE functions that can be used with PC/3270 or PC400. The table lists the name of the DDE function, the command the client sends to PC/3270 or PC400, and the values that can be used for the variables in the client command.

Table 131 (Page 1 of 4). Summary of DDE Functions in a 16-Bit Environment

Function name	Client command
Find Field	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aFIELD));
	cfFormat = CF_DSPTEXT
Get Keystrokes	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aKEYS));
	cfFormat = CF_DSPTEXT
Get Mouse Input	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aMOUSE));
	cfFormat = CF_TEXT | CF_DSPTEXT
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

Table 131 (Page 2 of 4). Summary of DDE Functions in a 16-Bit Environment

Function name	Client command
Get System Status	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aSTATUS));
	cfFormat = CF_TEXT
Get System SysItems	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aSYSITEMS));
	cfFormat = CF_TEXT
Get System Topics	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aTOPICS));
	cfFormat = CF_TEXT
Get Trim Rectangle	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aTRIMRECT));
	cfFormat = CF_TEXT
Initiate Session Conversation	SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELONG(aIBM327 32, aSessionN));
	N = a session letter A through Z.
Initiate Structured Field Conversation	SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELONG(aIBM327 32, aLUN_xxxx)
	N = a session letter A through Z. xxxx = a user defined string.
Initiate System Conversation	SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELONG(aIBM327 32, 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

DDE Functions in a 16-bit Environment

Table 131 (Page 3 of 4). Summary of DDE Functions in a 16-Bit Environment

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 Mouse Input Intercept	PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, MAKELONG(hOptions, aMOUSE));
	hOptions = Handle to a global memory object
Start Read SF	PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, MAKELONG(hOptions, aSF));
	hOptions = Handle to a global memory object
Start Session Advise	PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, MAKELONG(hOptions, aItem));
	hOptions = Handle to a global memory object aItem = OIA | PS | TRIMRECT
Stop Close Intercept	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELONG(NULL, aCLOSE));
Stop Keystroke Intercept	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELONG(NULL, aKEYS));

Table 131 (Page 4 of 4). Summary of DDE Functions in a 16-Bit Environment

Function name	Client command
Stop Mouse Input Intercept	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELONG(NULL, aMOUSE));
Stop Read SF	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELONG(NULL, aSF));
Stop Session Advise	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELONG(NULL, aItem));
	aItem = OIA | PS | TRIMRECT | NULL
Terminate Session Conversation	SendMessage(hServerWnd, WM_DDE_TERMINATE, hClientWnd, MAKELONG(NULL, NULL));
Terminate Structured Field Conversation	SendMessage(hServerWnd, WM_DDE_TERMINATE, hClientWnd, MAKELONG(NULL, NULL));
Terminate System Conversation	SendMessage(hServerWnd, WM_DDE_TERMINATE, hClientWnd, MAKELONG(NULL, NULL));
Write SF	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, MAKELONG(hData, aSF));
	hData = Handle to a global memory object

DDE Menu Item API in a 16-bitEnvironment

PC/3270 or PC400 supports the addition, deletion, and changing of attributes of a dynamic menu item to the session menu bar. A menu will then be created for this menu item with space for up to 16 submenu items.

PC/3270 or PC400 supports two kinds of DDE conversation. One is PC/3270 or PC400, which acts as a DDE menu client application, and the other is PC/3270 or PC400, which acts as a DDE menu server.

DDE Menu Client in a 16-bit Environment

To add, delete, and change menu items, the following DDE conversation must take place between the session and DDE menu server application.

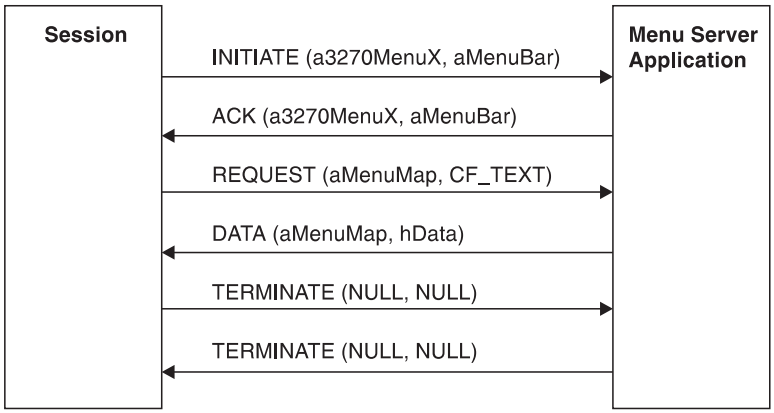

Figure 9. DDE Menu Client Conversation

The following data hierarchy details the menu map PC/3270 or PC400 expects when adding a dynamic menu item and submenu to a session menu bar:

```
POPUP    "MyMenu"
BEGIN
    MENUITEM "Send Files to Host",              SEND
    MENUITEM "Receive Files from Host",       RECEIVE
   MENUITEM  SEPARATOR
    MENUITEM "Convert Files",                    CONVERT
END
```

When the user selects a menu item from the new menu, PC/3270 or PC400 will send a DDE Initiate with 3270MenuN or 5250MenuN as the application and itemN token as the topic. If an ACK is received from the DDE application, PC/3270 or PC400 will inhibit the session from accepting user input. The menu client application can then display a dialog, and so on. When the menu server application has completed processing of the menu item, it will send a DDE Terminate to signal PC/3270 or PC400 the process is complete. PC/3270 or PC400 will then reenable the window for the user.

DDE Menu Server, 32-bit

To add, delete, and change menu items, the following DDE conversation must take place between the session and a DDE menu client application.

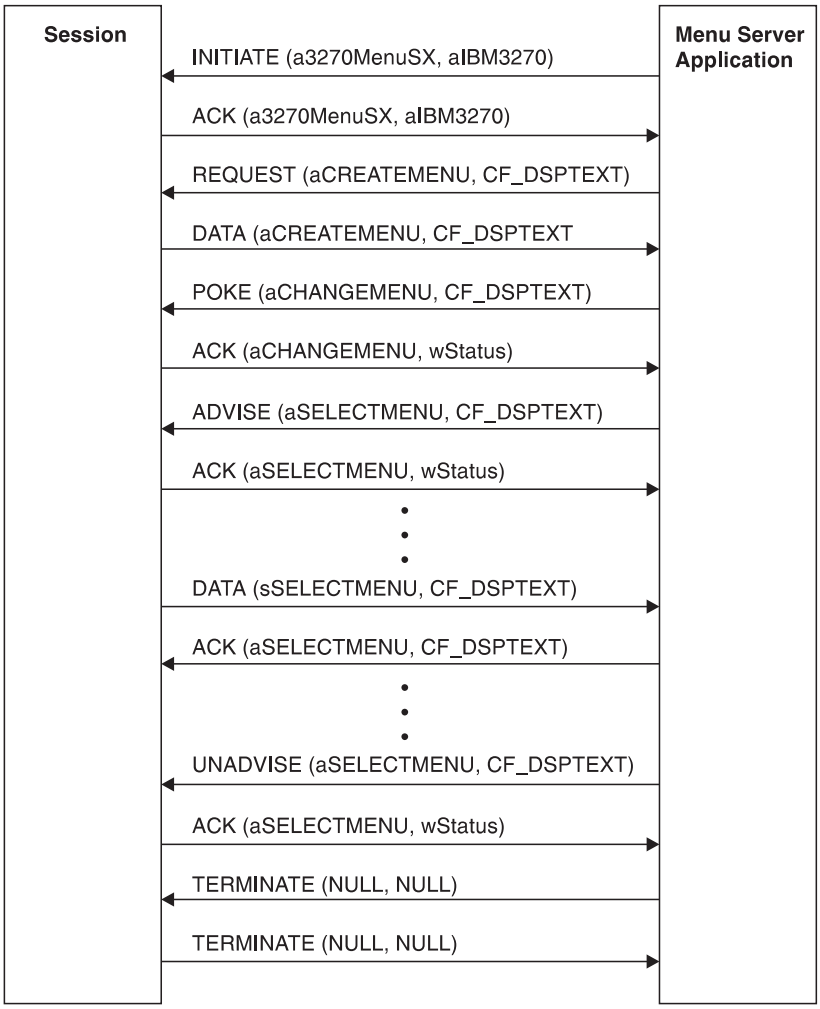

Figure 10. DDE Menu Server Conversation

When the user selects a menu item from the new menu, PC/3270 or PC400 will send a DDE DATA with aSELECTMENU as the item. When PC/3270 or PC400 sends DDE DATA to the client application, PC/3270 or PC400 will inhibit the session from accepting user input. The menu client application can then display a dialog, and so on. When the menu client application has completed processing of the menu item, it will send a DDE ACK to signal PC/3270 or PC400 the process is complete. PC/3270 or PC400 will then reenable the window for the user.

DDE Menu Functions in a 16-bit Environment

The following table lists the DDE Menu Item API functions that are available for use with PC/3270 or PC400 and the page in this section, where each function is more fully documented. PC/3270 and PC400 provide all of the following functions.

Function	Page
Change Menu Item	439
Create Menu Item	445
Initiate Menu Conversation	447
Start Menu Advise	448
Stop Menu Advise	450
Terminate Menu Conversation	451

Change Menu Item

PC/3270	PC400
X	X

The Change Menu Item function appends, deletes, inserts, modifies, and removes menu items. The client sends the following message to the session to change a menu.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
              MAKELONG(hData, aCHANGEMENU) );
```

where:

hData

Identifies a handle to a Windows global memory object that contains the requests for changing a menu. The global memory object contains the following structure:

```
typedef struct tagChangeMenu
{
 unsigned unused:13;         //    unused
 unsigned fRelease:1;       // Session frees memory
 unsigned freserved:2;      //    reserved
  int          cfFormat;          //  Always  CF_DSPTEXT
 HANDLE      hMenu;               // Handle of the menu item
  WORD        wPosition;        // The position of the menu
                                 //  item
 WORD        wIDNew;              // The menu ID of the new
                                 // menu item
 WORD        wOperation;       // Specifies the operation
 WORD        wFlags;             // Specifies the options
 unsigned char szItemName[1]; // String of the item
} CHANGEMENU, FAR  lpCHANGEMENU;
```

The following operations are supported:

```
MF_APPEND    // Appends a menu item to the end of a menu.
MF_CHANGE    // Modifies a menu item in a menu.
MF_DELETE    // Deletes a menu item from a menu, destroying
                // the menu item.
MF_INSERT    // Inserts a menu item into a menu.
MF_REMOVE    // Removes a menu item from a menu but does not
                // destroy the menu item.
```

If the MF_APPEND is specified in the wOperation field, the following fields must be filled:

hMenu

Identifies the menu to be appended. To append a new item to a pop-up menu, specify the handle that is returned from PC/3270 or PC400 when Create Menu Item function is executed. To append a new item to a top-level menu bar, specify NULL.

wIDNew

Specifies the command ID of the new menu item. If a new item is added to the top-level menu bar, the handle of the menu item returned from PC/3270 or PC400 when Create Menu Item function is executed.

wFlags

The following options can be set:

```
MF_CHECKED          // Places a checkmark next to
                    // the item.
MF_DISABLED         // Disables the menu item so
                    // that it cannot be selected,
                    // but does not gray it.
MF_ENABLED          // Enables the menu item so that
                    // it can be selected and
                    // restores from its grayed
                    //  state.
MF_GRAYED           // Disables the menu item so
                    // that it cannot be selected,
                    // and grays it.
MF_MENUBARBREAK     // Same as MF_MENUBREAK except
                    // that for pop-up menus,
                    // separates the new column from
                    // the old column with a
                    // vertical line.
MF_MENUBREAK        // Places the item on a new line
                    // for menu bar items.
                    // For pop-up menus, places the
                    // item in a new column, with
                    // no dividing line between the
                    //  columns.
MF_SEPARATOR        // Draws a horizontal dividing
                    // line.  Can only be used in a
                    // pop-up menu.  This line cannot
                    // be grayed, disabled, or
                    // highlighted.  The wIDNew and
                    // szItemName fields are
                    //  ignored.
MF_UNCHECKED        // Does not place a checkmark
                    // next to the item (default).
```

szItemName

Specifies the content of the new menu item. Contains a null-terminated character string.

If the MF_CHANGE is specified in the wOperation field, fill these fields:

hMenu

Identifies the menu to be changed. To change an item of a pop-up menu, specify the handle that is returned from PC/3270 or PC400 when Create Menu Item function is executed. To change an item to a top-level menu bar, specify NULL.

nPosition

Specifies the menu item to be changed. The interpretation of the wPosition parameter depends on the setting of the wFlags parameter.

MF_BYPOSITION

Specifies the position of the existing menu item. The first item in the menu is at position zero.

MF_BYCOMMAND

Specifies the command ID of the existing menu item.

wIDNew

Specifies the command ID of the menu item. If an item of the top-level menu bar is changed, the handle of the menu item returned from PC/3270 or PC400 when Create Menu Item function is executed.

wFlags

The following options can be set:

```
MF_BYCOMMAND        // Specifies that the nPosition
                    // parameter gives the menu
                    // item control ID number.
                    // This is the default if
                    // neither MF_BYCOMMAND nor
                    // MF_BYPOSITION is set.
MF_BYPOSITION       // Specifies that the nPosition
                    // parameter gives the position
                    // of the menu item to be
                    // changed rather than an ID
                    //  number.
MF_CHECKED          // Places a checkmark next to
                    // the item.
MF_DISABLED         // Disables the menu item so
                    // that it cannot be selected,
                    // but does not gray it.
MF_ENABLED          // Enables the menu item so
                    // that it can be selected and
                    // restores from its grayed
                    //  state.
MF_GRAYED           // Disables the menu item so
                    // that it cannot be selected,
                    // and grays it.
MF_MENUBARBREAK   // Same as MF_MENUBREAK except
                    // that for pop-up menus,
                    // separates the new column
                    // from the old column with a
                    // vertical line.
MF_MENUBREAK        // Places the item on a new
                    // line for menu bar items.
                    // For pop-up menus, places the
                    // item in a new column, with
                    // no dividing line between
                    // the columns.
MF_SEPARATOR        // Draws a horizontal dividing
                    // line.   Can only be used in
                    // a pop-up menu.   This line
                    // cannot be grayed, disabled,
                    // or highlighted.   The wIDNew
                    // and szItemName fields are
                    //  ignored.
MF_UNCHECKED        // Does not place a checkmark
                    // next to the item (default).
```

szItemName

Specifies the content of the menu item. Contains a null-terminated character string.

If the MF_DELETE is specified in the wOperation field, fill these fields:

hMenu

Identifies the menu to be deleted. To delete an item from a pop-up menu, specify the handle that is returned from PC/3270 or PC400 when Create Menu Item function is executed. To delete an item from a top-level menu bar, specify NULL.

nPosition

Specifies the menu item to be deleted. The interpretation of the nPosition parameter depends on the setting of the wFlags parameter.

MF_BYPOSITION
Specifies the position of the existing menu item. The first item in the menu is at position zero.

MF_BYCOMMAND
Specifies the command ID of the existing menu item.

wFlags

The following options can be set:

```
MF_BYCOMMAND        // Specifies that the nPosition
                    // parameter gives the menu
                    // item control ID number.
                    // This is the default if
                    // neither MF_BYCOMMAND nor
                    // MF_BYPOSITION is set.
MF_BYPOSITION       // Specifies that the nPosition
                    // parameter gives the position
                    // of the menu item to be
                    // deleted rather than an ID
                    //  number.
```

If the MF_INSERT is specified in the wOperation field, the following fields must be filled:

hMenu

Identifies the menu to be inserted. To insert an item to a pop-up menu, specify the handle that is returned from PC/3270 or PC400 when Create Menu Item function is executed. To change an item to a top-level menu bar, specify NULL.

nPosition

Specifies the menu item before the new menu item is to be inserted. The interpretation of the nPosition parameter depends on the setting of the wFlags parameter.

MF_BYPOSITION
Specifies the position of the existing menu item. The first item in the menu is at position zero.

MF_BYCOMMAND
Specifies the command ID of the existing menu item.

wIDNew

Specifies the command ID of the menu item or, if an item of the top-level menu bar is changed, the handle of the menu item returned from PC/3270 or PC400 when Create Menu Item function is executed.

wFlags

The following options can be set:

```
MF_BYCOMMAND        // Specifies that the nPosition
                    // parameter gives the menu
                    // item control ID number. This
                    // is the default if neither
                    // MF_BYCOMMAND nor MF_BYPOSITION
                    // is set.
MF_BYPOSITION       // Specifies that the nPosition
                    // parameter gives the position
                    // of the menu item to be
                    // changed rather than an ID
                    //  number.
MF_CHECKED          // Places a checkmark next to
                    // the item.
MF_DISABLED         // Disables the menu item so
                    // that it cannot be selected,
                    // but does not gray it.
MF_ENABLED          // Enables the menu item so
                    // that it can be selected and
                    // restores from its grayed
                    //  state.
MF_GRAYED           // Disables the menu item so
                    // that it cannot be selected,
                    // and grays it.
MF_MENUBARBREAK   // Same as MF_MENUBREAK except
                    // that for pop-up menus,
                    // separates the new column
                    // from the old column with a
                    // vertical line.
MF_MENUBREAK        // Places the item on a new
                    // line for menu bar items.
                    // For pop-up menus, places the
                    // item in a new column, with
                    // no dividing line between the
                    //  columns.
MF_SEPARATOR        // Draws a horizontal dividing
                    // line.   Can only be used in
                    // a pop-up menu.   This line
                    // cannot be grayed, disabled,
                    // or highlighted. The wIDNew
                    // and szItemName fields are
                    //  ignored.
MF_UNCHECKED        // Does not place a checkmark
                    // next to the item (default).
```

szItemName

Specifies the content of the menu item. Contains a null-terminated character string.

If the MF_REMOVE is specified in the wOperation field, the following fields must be filled:

hMenu

Identifies the menu to be removed. To remove an item from a pop-up menu, specify the handle that is returned from PC/3270 or PC400 when Create Menu Item function is executed. To remove an item from a top-level menu bar, specify NULL.

nPosition

Specifies the menu item to be removed. The interpretation of the nPosition parameter depends upon the setting of the wFlags parameter.

MF_BYPOSITION

Specifies the position of the existing menu item. The first item in the menu is at position zero.

MF_BYCOMMAND

Specifies the command ID of the existing menu item.

wFlags

The following options can be set:

```
MF_BYCOMMAND        // Specifies that the nPosition
                    // parameter gives the menu
                    // item control ID number.
                    // This is the default if
                    // neither MF_BYCOMMAND nor
                    // MF_BYPOSITION is set.
MF_BYPOSITION       // Specifies that the nPosition
                    // parameter gives the
                    // position of the menu item to
                    // be removed rather than an ID
                    //  number.
```

PC/3270 or PC400 Response

PC/3270 or PC400 receives the requests to change a menu and processes them. If the requests cannot be accepted, PC/3270 or PC400 returns a negative ACK message containing the following status codes in the low-order byte of the wStatus word. Otherwise, PC/3270 or PC400 returns a positive ack message signalling that the keystrokes have been sent.

WM_DDE_ACK(wStatus, aCHANGEMENU)

Return code	Explanation
1	The specified parameters are not valid.
6	The specified format is not valid.
9	A system error occurred.

Create Menu Item

PC/3270	*PC400*
X	X

The Create Menu Item function requests PC/3270 or PC400 to add a menu item to the menu bar. A pop-up menu will be created at the same time, but it is initially empty and can be filled with menu items by using this function. The string of the new menu item that will be added to a top-level menu bar, is also specified by using the change menu item function.

The client sends the following message to create a menu item.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aCREATEMENU) );
```

where:

cfFormat

Identifies the format for the ID of the new menu item. The valid value is CF_DSPTEXT.

aCREATEMENU

Identifies the create menu item.

PC/3270 or PC400 response

PC/3270 or PC400 returns the handle of the newly created menu item in a dde data message if the PC/3270 or PC400 can create a menu item.

WM_DDE_DATA(hData, aCREATEMENU)

where:

hData

Identifies a handle to a windows global memory object that contains the handle of the menu item. The global memory object contains the following structure:

```
typedef struct tagcreatemenu
}
 unsigned     unused:12,            //     unused
 unsigned     fresponse:1,         // true = dd_request response
 unsigned     frelease:1,          // true = client releases memory
 unsigned     reserved:1,           //     reserved
 unsigned     fackreq:1,            // true = dde_ack is required
 int          cfformat;                      // always cf_dsptext
 handle       hmemuitem;                     // handle of the menu item
} CREATEMENU, FAR  lpCREATEMENU;
```

or

WM_DDE_ACK(wStatus, aCREATEMENU)

If PC/3270 or PC400 cannot create a menu item, the following status codes are returned in the low-order byte of the wStatus word:

Return Code	Explanation
6	The specified format is not valid.
9	A system error occurred.

Initiate Menu Conversation

PC/3270	*PC400*
X	X

The Initiate Menu Conversation function connects a client application to an available session of PC/3270 or PC400. Once a menu conversation is established, the session menu is reserved exclusively for the client until the conversation is terminated.

The client application sends the following message to initiate a DDE conversation with a menu:

```
SendMessage( -1,
             WM_DDE_INITIATE,
             hClientWnd,
              MAKELONG(aIBM327 32, a327 MenuSN) );
```

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.

a3270MenuSN

Identifies the topic atom. The string used to create atom a3270MenuSN is 3270MenuS appended with the session ID A, B, ..., Z. In the PC400, the topic atom is a5250MenuSN and the string 5250MenuS appended with the session ID A, B, ..., Z. is used to create it.

PC/3270 or PC400 Response

If PC/3270 or PC400 can support a conversation with the client application, PC/3270 or PC400 acknowledges the INITIATE transaction with:

```
WM_DDE_ACK(aIBM327 32, a327 MenuSN)
```

Start Menu Advise

PC/3270	PC400
X	X

The Start Menu Advise function allows a client application to process a user defined routine when the menu item that is added by the client application, is selected. After using this function, the client receives DATA messages indicating which menu item is selected.

The client sends the following command to begin a menu advise.

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
              MAKELONG(hOptions, aSELECTMENU) );
```

where:

hOptions

Is a handle to a Windows global memory object, with the following structure:

```
typedef struct tagOPTIONS
{
 unsigned reserved:14;          // Reserved
 unsigned fDeferUpd:1;          // Send notification only
                                   // (Must be  )
 unsigned fAckReq:1;            // Client will ACK all notices
                                   // (Must be 1)
 WORD        cfFormat;             // Always CF_DSPTEXT
}    OPTIONS, FAR  lpOPTIONS;
```

aSELECTMENU

Identified a menu advise as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 receives the Start Menu Advise and returns an ACK message if it can start the function. Otherwise, a negative ACK message will be returned to the client with the following return codes in the low-order byte of the wStatus field.

Return Code	Explanation
1	Menu Advise has been already started.
6	The specified format is not valid.
9	A system error occurred.

```
WM_DDE_ACK(wStatus, aSELECTMENU)
```

Once the menu item (added to the client application) is selected, the client receives DATA messages notifying it which menu item is selected:

```
WM_DDE_DATA(hData, aSELECTMENU)
```

where:

hData

Identifies a handle to a Windows global memory object containing:

```
typedef struct tagSELECTMENU
{
 unsigned    Unused:12,          //    unused
 unsigned    fResponse:1,       // TRUE = DD_REQUEST response
 unsigned    fRelease:1,        // TRUE = Client releases memory
 unsigned    reserved:1,        //    reserved
 unsigned    fAckReq:1,          // TRUE = DDE_ACK is required
 int         cfFormat;                  // Always CF_DSPTEXT
 WORD        uIDSelected;                // Command ID of the
                                         // selected menu item
} SELECTMENU, FAR  lpSELECTMENU;
```

The DATA messages continue until a Stop Menu Advise message is sent to PC/3270 or PC400.

Stop Menu Advise

PC/3270	*PC400*
X	X

The Stop Menu Advise function ends a client application's ability to process a user-defined routine when the menu item added by the client application is selected. The client sends the following command to perform the Stop Menu Advise function.

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
              MAKELONG(NULL, aSELECTMENU) );
```

where:

aSELECTMENU

Identifies a menu advise as the item.

PC/3270 or PC400 Response

If PC/3270 or PC400 can perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aCLOSE)
```

If PC/3270 or PC400 cannot perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing negative status information and the following return codes in the low-order byte of the wStatus word:

Return Code	Explanation
1	Advise has not started yet.
9	A system error occurred.

Terminate Menu Conversation

PC/3270	*PC400*
X	X

The Terminate Menu Conversation function disconnects the client from the PC/3270 or PC400 session with which a conversation had been previously started.

The client sends the following command to terminate a session conversation:

```
SendMessage( hServerWnd,
             WM_DDE_TERMINATE,
             hClientWnd,
              MAKELONG(NULL, NULL) );
```

PC/3270 or PC400 Response

PC/3270 or PC400 acknowledges the terminate command with this message:

WM_DDE_TERMINATE

DDE Menu Functions in a 16-bitEnvironment

Index

Numerics

01, Connect Presentation Space 56
02, Disconnect Presentation Space 94
03, Send Key 102, 151, 175, 182
04, Wait 182
05, Copy Presentation Space 75
06, Search Presentation Space 147, 182
07, Query Cursor Location 122
08, Copy Presentation Space to String 80
09, Set Session Parameters 160
10, Query Sessions 131
11, Reserve 143
12, Release 142, 143
13, Copy OIA 66
14, Query Field Attribute 123
15, Copy String to Presentation Space 88
18, Pause 112, 180
20, Query System 133
21, Reset System 143, 144, 160
22, Query Session Status 60, 129
23, Start Host Notification 112, 125, 171
24, Query Host Update 112, 113, 125, 180
25, Stop Host Notification 180
30, Search Field 145, 182
31, Find Field Position 62, 97
32, Find Field Length 62, 95
33, Copy String to Field 85
34, Copy Field to String 61
40, Set Cursor 159
41, Start Close Intercept 168
42, Query Close Intercept 119
43, Stop Close Intercept 179
50, Start Keystroke Intercept 175
51, Get Key 100, 118, 175
52, Post Intercept Status 118, 175
53, Stop Keystroke Intercept 181
90, Send File 149
91, Receive File 140
99, Convert Position or Convert RowCol 58
61, Lock PMSVC API 106
101, Connect Window Services 54
102, Disconnect Window Service 93
103, Query Window Coordinates 127
104, Window Status 114
105, Change Switch List LT Name 50
106, Change PS Window Name 48
110, Start Playing Macro 178
3270 terminal emulation 331

A

Allocate Communications Buffer (123) 46
applications
  error codes 353
  using SRPI 332
ASCII characters 35
ASCII mnemonics 35
ATTRB 162
attribute bytes 61, 75, 80, 85, 89, 162
AUTORESET 163

B

beep 118
BLANK 164
buffer size 333

C

C for OS/2
  memory preallocation for returned data strings 21
C for Windows 95
  compiling 41
  function call syntax 40
  linking 41
  parameter name declaration 40
C language
  init_send_req_parms 334
  interface 333
  record definition 334
  requesters 333
  return codes
  send_request 334
  syntax 333
C++ for Windows 95
  compiling 41
  function call syntax 40
  linking 41
  parameter name declaration 40
call (input) parameters
  general 43
  passing 21
call statement
  See function calls
call/return 328
calls, prerequisite 43
Change Menu Item 273, 439
Change Switch List LT Name (105)
Change Window Name (106)
character, escape 36, 101, 152, 163
characters, ASCII 35

© Copyright IBM Corp. 1995, 1996

command highlighting xv
common error handling
general 24
common error-handling
return code summary 33
communication services functions
Receive File (91) 140
Send File (90) 149
Communications Manager
applications 332
Compatibility with Personal Communications V4.1 with Windows 3.1
compatible
hardware xvi
host xvii
related publications xvii
software xvi
compile option highlighting xv
compiling
general 24
IBM C for OS/2 41
IBM C++ for OS/2 41
composite screen 1
Connect for Structured Fields (120)
Connect Presentation Space (1) 56, 151
functions where not required 57
interaction with disconnect 28
Connect Window Services (101)
Connect Window Services (101) 54
Convert Position or Convert RowCol (99)
return code 24
COPY FIELD TO STRING (34) i, 15, 61
copy functions
Copy Field to String (34) 61
Copy OIA (13) 66
Copy Presentation Space (5) 75
Copy Presentation Space to String (8) 80
Copy String to Field (33) 85
Copy String to Presentation Space (15) 88
Copy OIA (13) 12, 66
Copy Presentation Space (5) 75
Copy Presentation Space to String (8) 12, 80
Copy String to Field (33) 15, 85
Copy String to Presentation Space (15) 88
CPRB (Connectivity Programming Request Block)
mapping
C language
storage 333
create menu item 280, 445
cursor movement 13
customization 27

D

data string
calling 22
data string *(continued)*
returned 23
specification 13
DDE data items
LU Topic 199
Session Topic 199
System Topic 198
DDE data items, 16-bit
LU Topic 363
Session Topic 363
System Topic 362
DDE Functions in a 16-Bit Environment 361
Find Field 366
function list 364
Get Keystrokes 368
Get Mouse Input 370
Get Number of Close Requests 373
Get Operator Information Area 374
Get Partial Presentation Space 375
Get Presentation Space 378
Get Session Status 381
Get System Configuration 383
Get System Formats 385
Get System Status 386
Get System SysItems 387
Get System Topics 388
Get Trim Rectangle 389
Initiate Session Conversation 390
Initiate Structured Field Conversation 391
Initiate System Conversation 392
naming conventions for parameters 365
Put Data to Presentation Space 393
Search for String 394
Send Keystrokes 395
Session Execute Macro 396
Set Cursor Position 403
Set Mouse Intercept Condition 404
Set Presentation Space Service Condition 407
Set Session Advise Condition 409
Set Structured Field Service Condition 411
Start Close Intercept 412
Start Keystroke Intercept 414
Start Mouse Input Intercept 415
Start Read SF 419
Start Session Advise 421
Stop Close Intercept 423
Stop Keystroke Intercept 424
Stop Mouse Input Intercept 425
Stop Read SF 426
Stop Session Advise 427
summary of DDE functions in a 16-bit environment 432
Terminate Session Conversation 428
Terminate Structured Field Conversation 429
Terminate System Conversation 430
Write SF 431

DDE functions in a 32-bit environment 197
DDE data items, 32-bit 197
LU Topic 199
Session Topic 199
System Topic 198
Find Field 202
function list 200
Get Keystrokes 204
Get Mouse Input 206
Get Number of Close Requests 209
Get Operator Information Area 210
Get Partial Presentation Space 211
Get Presentation Space 214
Get Session Status 217
Get System Configuration 219
Get System Formats 221
Get System Status 222
Get System SysItems 223
Get System Topics 224
Get Trim Rectangle 225
Initiate Session Conversation 226
Initiate Structured Field Conversation 227
Initiate System Conversation 228
naming conventions for parameters 201
Put Data to Presentation Space 229
Search for String 231
Send Keystrokes 232
Session Execute Macro 234
Set Cursor Position 241
Set Mouse Intercept Condition 242
Set Presentation Space Service Condition 245
Set Session Advise Condition 247
Set Structured Field Service Condition 249
Start Close Intercept 250
Start Keystroke Intercept 252
Start Mouse Input Intercept 253
Start Read SF 257
Start Session Advise 259
Stop Close Intercept 261
Stop Keystroke Intercept 262
Stop Mouse Input Intercept 263
Stop Read SF 264
Stop Session Advise 265
Terminate Session Conversation 266
Terminate Structured Field Conversation 267
Terminate System Conversation 268
Write SF 269
DDE menu functions in a 16-bit environment 436
Change Menu Item 439
create menu item 445
Initiate Menu Conversation 447
list 438
Start Menu Advise 448
Stop Menu Advise 450
Terminate Menu Conversation 451
DDE menu functions in a 32-bit environment 270
Change Menu Item 273
create menu item 280
Initiate Menu Conversation 282
list 272
Start Menu Advise 283
Stop Menu Advise 285
Terminate Menu Conversation 286
debugging 25
default
values 332
default directory
Receive File 141
device services functions
Get Key (51) 100
Post Intercept Status (52) 118
Release (12) 142
Reserve (11) 143
Start Keystroke Intercept (50) 175
Stop Keystroke Intercept (53) 181
directory, default
Receive File 141
Send File 150
Disconnect from Structured Fields (121) 91
Disconnect Presentation Space (2) 94
interaction with connect 28
Disconnect Window Service (102) 93
DLL
introduction 3

E

EAB 164
EHLLAPI
function names xv
functions 43
interface description 2
purpose 1
running your program 25
summary 44
EHLLAPI/DDE 1
advantages of using 1
defined 1
overview 1, 3
parts of 3
EHLLAPI/DDE.DLL 3
emulator, terminal 2
EOT 14, 161
error code, extended 134
error handling 353
general 33
source code 23
ESC 163
escape character 36, 101, 152, 163
exception code values 358

exception object values 358
exiting application 24
extended error code 134

F

field-formatted PS 62, 145
field-related functions
  Copy Field to String (34) 61
  Copy String to Field (33) 85
  Find Field Length (32) 95
  Find Field Position (31) 97
  Search Field (30) 145
fields, host
  input protected 151
  numeric only 151
file transfer functions
  Receive File (91) 140
  Send File (90) 149
Find Field 202, 366
Find Field Length (32) 62, 95
Find Field Position (31) 62, 97
flow, requester and server 330
FPAUSE 162
Free Communications Buffer (124) 99
function calls 3
  call (input) parameters 43
  defined 2
  notes on using the function 43
  page layout conventions 43
  prerequisite calls 43
  return (output) parameters 43
  source code pattern 22
  use of 43
function names xv
function number
  calling, general 21
  introduction 2
  returned, general 23
function number parameter xv

G

Get Key (51) 35, 100, 118, 175
Get Keystrokes 204, 368
Get Mouse Input 206, 370
Get Number of Close Requests 209, 373
Get Operator Information Area 210, 374
Get Partial Presentation Space 211, 375
Get Presentation Space 214, 378
Get Request Completion (125) 103
Get Session Status 217, 381
Get System Configuration 219, 383
Get System Formats 221, 385
Get System Status 222, 386
Get System SysItems 223, 387
Get System Topics 224, 388
Get Trim Rectangle 225, 389

H

hardware requirements xvi
highlighting xv
  ASCII mnemonics xv
  commands
    compile xv
    link xv
  displayed text xv
  entry-point names xv
  function names xv
  keywords xv
  operator input xv
  options
    compile xv
    link xv
    session xv
  programmer input xv
  sample code xv
  variable names xv
hllapi 22
host
  computer router 330
  computer server 330
host fields
  input protected 151
  numeric only 151
host requirements xvii
host session 27

I

IBM C++ for OS/2
  memory preallocation for returned data strings 21
IBM Support Center 133
init_send_req_parms
  C language 334
initialization
  general 21
Initiate Menu Conversation 282, 447
Initiate Session Conversation 226, 390
Initiate Structured Field Conversation 227, 391
Initiate System Conversation 228, 392
input protected fields 151
installation 27
interface description, general 2
invoking SEND_REQUEST 333
IPAUSE 162

K

keyboard mnemonics
  See also “mnemonics.” 34
keyboard, session 34
keystroke interception, Get Key (51) 100
keyword highlighting xv

L

language interface
  C language 333
  language support, programming 5
length
  calling 22
  returned 24
licensing agreement xi
link option highlighting xv
linking
  general 24
  IBM C for OS/2 41
  IBM C++ for OS/2 41
LOCK PRESENTATION SPACE API (60) 109
LOCK Window Services API (61) 106
LWAIT 163, 182

M

memory preallocation 21, 24
memory requirements xvi
mnemonics 89
  for Send Key 34
mnemonics, ASCII 35
mnemonics, shift key 35

N

NOATTRB 162
NOBLANK 164
NOEAB 164
NOQUIET 162
NORESET 163
Notification of 3270 screen update indicator
  C language
NOXLATE 164
NULLATTRB 162
numeric only fields 151
NWAIT 163, 182

O

OIA 66, 182
Operator Information Area
  See “OIA.” 66
operator services functions
  Pause (18) 112
  Query Host Update (24) 125
operator services functions *(continued)*
  Query Session Status (22) 129
  Query Sessions (10) 131
  Query System (20) 133
  Reset System (21) 144
  Send Key (3) 151
  Set Session Parameters (9) 160
  Start Host Notification (23) 171
  Stop Host Notification (25) 180
  Wait (4) 182
options 161—167
options, session, highlighting xv

P

parameters
  assigning values 23
  call 43
  data string, call 22
  data string, return 23
  function number xv
  function number, call 21
  function number, return 23
  general 2
  in function call statement 22
  length, call 22
  length, return 24
  passing values 23
  PS position, call 22
  receiving values 23
  return code, return 24
  returned 332
  SEND_REQUEST 331
  supplied 331
  variable name rules 23
path, default
  Receive File 141
  Send File 150
Pause (18) 12, 112, 180
PCSAPI 189
  how to use 189
  pcsConnectSession 190
  pcsDisconnectSession 191
  pcsQueryEmulatorStatus 192
  pcsQueryWorkstationProfile 193
  pcsStartSession 194
  pcsStopSession 195
pcsDisconnectSession 191
pcsQueryEmulatorStatus 192
pcsQueryWorkstationProfile 193
pcsStartSession 194
pcsStopSession 195
performance considerations 333
Post Intercept Status (52) 17, 118, 175
preallocation of memory 21, 24

prerequisite calls
See also individual functions by name
general 43
REXX EHLLAPI
presentation services functions
Connect Presentation Space (1) 56
Copy Field to String (34) 61
Copy OIA (13) 66
Copy Presentation Space (5) 75
Copy Presentation Space to String (8) 80
Copy String to Field (33) 85
Copy String to Presentation Space (15) 88
Disconnect Presentation Space (2) 94
Find Field Length (32) 95
Find Field Position (31) 97
Get Request Completion (125) 103
LOCK PRESENTATION SPACE API (60) 109
Query Cursor Location (7) 122
Query Field Attribute (14) 123
Search Field (30) 145
Search Presentation Space (6) 147
Set Cursor (40) 159
presentation space
character table 67
cursor movement 13
determining size of existing 27
field-formatted 61, 62, 85, 95, 97, 145
host-connected 28
maximum number of 28
how specified 28
identifier 27, 28, 29
blank specifier 29
function 27
how processed 28
letter specifier 29
null specifier 29
processing for functions not requiring connect 29
processing for functions requiring connect 29
size 22, 24, 27
supported types 27
presentation space position (calling)
general 22
programmed operator 1, 11
defined 1
tasks 1
programming interface, server-requester 328
programming language support 5
PSID
See presentation space, identifier
Put Data to Presentation Space 229, 393

Q

Query Close Intercept (42) 119
Query Communications Buffer Size (122) 120
Query Cursor Location (7) 122
Query Field Attribute (14) 123
Query Host Update (24) 112, 113, 125, 171, 180
Query Reply Data Structures
Query Session Status (22) 60, 129
Query Sessions (10) 131
Query System (20) 133
Query Window Coordinates (103) 127
QUIET 162

R

Read Structured Fields (126) 135
Receive File (91) 15, 140, 163
default path for target file 141
RECEIVE.EXE location 140
record definition
C language 334
relationship requester server 329
Release (12) 15, 142, 143
request record initialization
C language
requester
C language 333
server flow, and 330
server relationship 329
Reserve (11) 15, 142, 143
Reset System (21) 21, 24, 142, 143, 144, 160
return (output) parameters
general 43
processing 23
return codes 353
general 24
summary 33
router
dynamic link
SRPI 330, 331

S

sample code highlighting xv
Search Field (30) 145, 182
Search for String 231, 394
search functions
Search Field (30) 145
Search Presentation Space (6) 147
Search Presentation Space (6) 12, 147, 182
Send File (90) 15, 149, 163
default path for target file 150
Send File 150
SEND.EXE location 149
Send Key (3) 35, 102, 151, 175, 182
Send Keystrokes 232, 395
SEND_REQUEST
invoking 333
parameters
returned 332

SEND_REQUEST *(continued)*
parameters *(continued)*
supplied 331
processing errors 353
routing 330
send_request function
C language 334
definition
sending keystrokes
mnemonics 34
Send Key (3) 151
server
name 332
return codes 359
server-requester programming interface 328
service 133
Session Execute Macro 234, 396
session keyboard 34
session options highlighting xv
session, host 27
Set Cursor (40) 159
Set Cursor Position 241, 403
Set Mouse Intercept Condition 242, 404
Set Presentation Space Service Condition 245, 407
Set Session Advise Condition 247, 409
Set Session Parameters (9) 151
general 160
List of affected functions 160
Valid Input 161—167
Set Structured Field Service Condition 249, 411
shift key mnemonics 35
size of presentation space 22, 24, 27
declaring 27
determining 27
maximum number of 27
valid names 27
software requirements xvi
high-level languages xvi
source code syntax
IBM C for OS/2 40
IBM C++ for OS/2 40
specifying strings 13, 89
SRCHALL 161
SRCHBKWD 162
SRCHFROM 161
SRCHFRWD 162
Start Close Intercept 250, 412
Start Close Intercept (41) 168
Start Host Notification (23) 112, 125, 162, 171, 180
Start Keystroke Intercept 252, 414
Start Keystroke Intercept (50) 175
Start Menu Advise 283, 448
Start Mouse Input Intercept 253, 415
Start Playing Macro (110) 178
Start Read SF 257, 419
Start Session Advise 259, 421
statement, call
See function calls
Stop Close Intercept 261, 423
Stop Close Intercept (43) 179
Stop Host Notification (25) 180
Stop Keystroke Intercept 262, 424
Stop Keystroke Intercept (53) 181
Stop Keystroke Intercept (53) you can call the 17
Stop Menu Advise 285, 450
Stop Mouse Input Intercept 263, 425
Stop Read SF 264, 426
Stop Session Advise 265, 427
STREOT 14, 161
string interception, Get Key (51) 100
string specification 13, 161
STRLEN 14, 161
syntax
See also source code syntax
C language 333

T

terminal emulator 2
Terminate Menu Conversation 286, 451
Terminate Session Conversation 266, 428
Terminate Structured Field Conversation 267, 429
Terminate System Conversation 268, 430
TIMEOUT 162
trace 19
trademarks xii
transport layer errors 353
TWAIT 163, 182

U

UERCPRB
C language 334

V

variable names, highlighting xv

W

Wait (4) 12, 151, 182
window services functions
Change Switch List LT Name (105) 50
Change Window Name (106) 48
LOCK Window Services API (61) 106
Window Status (104) 114
Write SF 269, 431
Write Structured Fields (127) 183

X
XLATE 164

Communicating Your Comments to IBM

Personal Communications AS/400 and 3270
for Windows 95
Emulator Programming
Version 4.1

Publication No. SC31-8207-00

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

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

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

If you prefer to send comments by mail, use the RCF at the back of this book.

If you prefer to send comments by FAX, use this number:

United States and Canada: **1-800-227-5088**

If you prefer to send comments electronically, use this network ID:

- IBM Mail Exchange: **USIB2HPD at IBMMAIL**
- IBMLink: **CIBMORCF at RALVM13**
- Internet: **USIB2HPD@VNET.IBM.COM**

Make sure to include the following in your note:

Title and publication number of this book
Page number or topic to which your comment applies.

Help us help you!

Personal Communications AS/400 and 3270
for Windows 95
Emulator Programming
Version 4.1

Publication No. SC31-8207-00

We hope you find this publication useful, readable and technically accurate, but only you can tell us! Your comments and suggestions will help us improve our technical publications. Please take a few minutes to let us know what you think by completing this form.

Overall, how satisfied are you with the information in this book?	Satisfied	Dissatisfied

How satisfied are you that the information in this book is:	Satisfied	Dissatisfied
Accurate		
Complete		
Easy to find		
Easy to understand		
Well organized		
Applicable to your task		

Specific Comments or Problems:

Please tell us how we can improve this book:

Thank you for your response. When you send information to IBM, you grant IBM the right to use or distribute the information without incurring any obligation to you. You of course retain the right to use the information in any way you choose.

Your Internet Address: _____

Name

Address

Company or Organization

Phone No.

Help us help you!
SC31-8207-00

Cut or Fold
Along Line

IBM®

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

Information Development
Department CGMD
International Business Machines Corporation
PO BOX 12195
RESEARCH TRIANGLE PARK NC 27709-9990

Fold and Tape **Please do not staple** Fold and Tape

SC31-8207-00

Cut or Fold
Along Line

IBM®

Part Number: 64H8207

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

64H827

SC31-827-