LANDP® Family



# Programming Reference

*Version 6.0*

SC34-6045-00

LANDP® Family



# Programming Reference

*Version 6.0*

SC34-6045-00

**Note**

Before using this information and the product it supports, be sure to read the general information under Appendix C, “Notices” on page 673.

**First Edition (February 2002)**

This book is based on the previous edition, *LANDP Family Programming Reference Version 5.0*, GC34-5531-00, which remains applicable and current for users of LANDP® Version 5.0.

This edition applies to LANDP Family Version 6 (Program number 5724–B16) and to all subsequent releases and modifications, until otherwise indicated in new editions. Make sure you are using the correct edition for the level of product.

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

At the back of this publication is a page titled “Sending your comments to IBM”. If you want to make comments, but the methods described are not available to you, please address them to:

IBM United Kingdom Laboratories, User Technologies,
Mail Point 095, Hursley Park, Winchester, Hampshire, England, SO21 2JN.

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

**© Copyright International Business Machines Corporation 1992, 2002. All rights reserved.**
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

**Figures** . . . . . xxi

**Tables of Function Codes** . . . . . xxiii

**About this book** . . . . . xxv
Who should read this book . . . . . xxv
What you need to know . . . . . xxv
How this book is organized . . . . . xxv
Conventions used in this book . . . . . xxvi
Operating environments . . . . . xxvi
Function, return, and event codes . . . . . xxvii
Bit positions . . . . . xxvii
DB2 Universal Database® . . . . . xxvii
Windows . . . . . xxvii
Linux . . . . . xxviii
Related Information . . . . . xxviii
Web site . . . . . xxviii

**Summary of Changes** . . . . . xxix

**Part 1. Supervisor** . . . . . 1

**Chapter 1. Supervisor local functions** . . . . . 3
Using the supervisor local functions . . . . . 4
Linux semaphore-handling . . . . . 5
Request reference . . . . . 6
Ask for asynchronous events (AA function) . . . . . 6
Associate semaphore (AS function) . . . . . 7
Dissociate semaphore (DS function) . . . . . 9
Simulate hot-key (EE function) . . . . . 10
Disconnect an application program (EJ function) . . . . . 11
Unload LANDP (ES function) . . . . . 12
Inquire information (II function) . . . . . 14
Initialize (IN function) . . . . . 16
Query event (QE function) . . . . . 17
System status information (SI function) . . . . . 18
Start posting events (SP function) . . . . . 20
Activate and deactivate timers (T0 through T8 functions) . . . . . 21
Stop posting events (TP function) . . . . . 23
Wait for asynchronous events (WM function) . . . . . 24
Asynchronous request—asking for TT request (Z4 function) . . . . . 29
Asynchronous event notification or cancellation (Z5 function) . . . . . 30
Extended asynchronous event notification (ZN function) . . . . . 31

**Part 2. Wide area communication servers** . . . . . 33

© Copyright IBM Corp. 1992, 2002 

**Chapter 2. SNA communication server** . . . . . 35
SNA server characteristics . . . . . 37
BIND parameters . . . . . 38
BIND protocols . . . . . 38
SNA connection process . . . . . 42
SSCP–LU session . . . . . 48
Cryptography support in the SNA server . . . . . 48
SNA support for more than 30 user sessions per workstation . . . . . 49
Customization . . . . . 49
Installation requirements . . . . . 50
Function requests . . . . . 50
Events . . . . . 50
Compatibility . . . . . 50
Special considerations for LANDP for DOS . . . . . 50
CN and RL functions . . . . . 50
ACTPU received . . . . . 50
ACTLU received . . . . . 51
Using the SNA server . . . . . 51
Request reference . . . . . 53
Close (CL function) . . . . . 53
Connect (CN function) . . . . . 54
Define connection (DC function) . . . . . 55
Get status (GS function) . . . . . 58
Open (OP function) . . . . . 58
Query connection (QC function) . . . . . 60
Read host (RH function) . . . . . 60
Release (RL function) . . . . . 64
Send host (SH function) . . . . . 64
Server information (ZI function) . . . . . 66
DOS compression server . . . . . 68
Request-Unit compression . . . . . 68
Writing your own compression server . . . . . 69
Request reference . . . . . 70
Example program that uses the SNA server . . . . . 71
Migration considerations . . . . . 73

**Chapter 3. The cryptographic interface** . . . . . 75
Cipher blocks . . . . . 76
Cipher block chaining . . . . . 76
Using encryption and decryption . . . . . 77
The LANDP cryptographic interface . . . . . 78
Key records . . . . . 78
Using the cryptographic interface . . . . . 79
Request reference . . . . . 80
Encrypt (X'0030' function) . . . . . 80
Decrypt (X'0031' function) . . . . . 81
Random number generation (X'0040' function) . . . . . 82
Session key add (X'0050' function) . . . . . 83
Session key update (X'0051' function) . . . . . 84
Key record delete (X'0052' function) . . . . . 84

Example of SNA session-level encryption using the cryptographic interface . . . . 85
SNA session-level encryption process . . . . . 86

**Chapter 4. Native X.25 communication server** . . . . . 91
Server characteristics . . . . . 91
Caller session initiation . . . . . 92
Called session initiation . . . . . 92
Operation . . . . . 93
Using the native X.25 server . . . . . 94
Request reference . . . . . 95
Close (CL function) . . . . . 95
Connect (CN function) . . . . . 95
Define connection (DC function) . . . . . 96
Get status (GS function). . . . . 98
Open (OP function) . . . . . 99
Query connection (QC function). . . . . 100
Read host (RH function) . . . . . 101
Release (RL function) . . . . . 103
Send host (SH function) . . . . . 104

**Chapter 5. Program-to-program communication server** . . . . . 105
PPC conversation allocation . . . . . 105
Contention-winner application programs . . . . . 105
Contention-loser application programs. . . . . 106
Conversation state changes . . . . . 106
Application program in SEND state. . . . . 106
Application program in RECEIVE state . . . . . 106
PPC conversation deallocation . . . . . 107
PPC responses management . . . . . 107
Server exit for user data processing . . . . . 107
Conversation types . . . . . 108
Using the PPC server . . . . . 108
PPC server flag description . . . . . 110
Request reference . . . . . 111
Close conversation (CL function) . . . . . 111
Get attributes (GA function) . . . . . 112
Get status (GS function) . . . . . 113
Open conversation (OP function) . . . . . 113
Receive data (RD function) . . . . . 115
Send data (SD function) . . . . . 116

**Chapter 6. MQSeries Link server** . . . . . 119
Using the MQSeries Link server. . . . . 120
Integration With MQSeries . . . . . 122
Request reference . . . . . 123
Begin Transaction (BT). . . . . 123
Checkpoint (CP) . . . . . 124
Close session (CS) . . . . . 124
Close queue (CQ) . . . . . 125
End Transaction (ET) . . . . . 126

MQGET (GQ function) . . . . . 127
MQPUT (PQ function) . . . . . 131
MQPUT1 (P1 function) . . . . . 134
Rollback (RB function) . . . . . 135

**Chapter 7. TCP/IP Wide Area Communications server** . . . . . 137

**Part 3. I/O device servers** . . . . . 139

**Chapter 8. Financial printer server** . . . . . 141
Using the financial printer server . . . . . 143
Financial printer server flag descriptions . . . . . 145
Request reference . . . . . 146
Arm operator panel for user input (AR function) . . . . . 146
Clear operator panel display (CD function) . . . . . 148
Check printing status (CH function). . . . . 148
Close printer (CL function) . . . . . 149
Format parameter load (DF function) . . . . . 149
Download user characters (DU function) . . . . . 155
Get error counters (EC function) . . . . . 156
Set lights (LL function) . . . . . 158
Open printer (OP function) . . . . . 158
Read code page (RC function) . . . . . 159
Read data (RD function) . . . . . 160
Set redirection mode (RM function). . . . . 161
Set code page (SC function) . . . . . 162
Write to printer (WR function) . . . . . 162
Migration considerations . . . . . 164

**Chapter 9. IBM 4748 printer server** . . . . . 167
Using the A/B keys for printer sharing . . . . . 169
Considerations for IBM DOS H7.0 and IBM OS/2 Warp H3.0 . . . . . 169
User-defined character (UDC) support. . . . . 169
Using the 4748 printer server . . . . . 170
IBM 4748 printer server flag descriptions . . . . . 172
Request reference . . . . . 173
Check printing status (CH function). . . . . 173
Close printer (CL function) . . . . . 173
Format parameter load (DF function) . . . . . 173
Get error counters (EC function) . . . . . 181
Load user-defined character (LD function) . . . . . 182
Set lights (LL function) . . . . . 184
Open printer (OP function) . . . . . 184
Read data (RD function) . . . . . 185
Write to printer (WR function) . . . . . 185

**Chapter 10. IBM 4770 printer server**. . . . . 189
Using the 4770 printer server . . . . . 190
4770 printer server flag descriptions . . . . . 191

Request reference . . . . . 192
Check printing status (CH function). . . . . 192
Close printer (CL function) . . . . . 192
Format parameter load (DF function) . . . . . 192
Get error counters (EC function) . . . . . 194
Open printer (OP function) . . . . . 195
Write to printer (WR function) . . . . . 196

**Chapter 11. Printer manager server** . . . . . 199
Modes of operation . . . . . 200
Using the printer manager server . . . . . 200
Request reference . . . . . 201
Acquire a parallel printer (AC function) . . . . . 201
Release a parallel printer port (RL function) . . . . . 202

**Chapter 12. Magnetic stripe reader/encoder server** . . . . . 203
Using the MSR/E server . . . . . 205
Request reference . . . . . 206
Arm the IBM 47xx MSR/E device (AR function). . . . . 206
Arm the IBM 47xx MSR/E device (AT function) . . . . . 207
Check the write status (CH function) . . . . . 208
Close (CL function) . . . . . 209
Load or retrieve device parameters (DV function) . . . . . 209
Get error counters and read/encode capability (EC function) . . . . . 211
Terminate a pending function (KL function) . . . . . 213
Open (OP function) . . . . . 213
Read from the IBM 47xx MSR/E (RD function) . . . . . 214
Write to IBM 47xx MSR/E device (WR function) . . . . . 214
Write to IBM 47xx MSR/E device (WT function) . . . . . 215
Event notification. . . . . 215
Input data formats . . . . . 216
Non-4704 read compatibility mode . . . . . 216
4704 read compatibility mode . . . . . 217
Read format if AT function is used (LANDP for OS/2 only) . . . . . 218
Output data formats. . . . . 219
Output data format used in WT function (LANDP for OS/2) . . . . . 219
Migration considerations . . . . . 220

**Chapter 13. Personal identification number pad server** . . . . . 223
Cryptographic function details . . . . . 224
Using the PIN pad server . . . . . 226
PIN pad server flag summary . . . . . 227
Request reference . . . . . 228
Arm the PIN pad or MSR for data input (AR function) . . . . . 228
Arm the MSR device (AT function) . . . . . 234
Close PIN pad or MSR (CL function) . . . . . 236
Load or retrieve MSR device parameters (DV function) . . . . . 236
Get error counters and MSR read/encode capability (EC function) . . . . . 239
Generate message authentication code (GA function) . . . . . 240
Load initial chaining value (IV function) . . . . . 241

Terminate a pending function (KL function) . . . 242
Load key (LK function) . . . 242
Load master key (LM function) . . . 243
Load PIN verification parameters (LP function) . . . 244
Open PIN pad (OP function) . . . 245
Read from the IBM 47xx PIN pad (RD function) . . . 246
Read serial number (RN function) . . . 247
Verify message authentication code (VA function) . . . 248
Write display (WD function) . . . 249
Event notification . . . 249
Migration considerations . . . 249

**Part 4. Data management servers** . . . 251

**Chapter 14. Shared-file server** . . . 255
Shared-file server structure . . . 256
Shared-file description . . . 256
Program control block . . . 257
Initialization of the shared-file server structure . . . 258
Shared-file distributor server . . . 258
Shared-file replicator server . . . 259
Log files . . . 259
Variable length records . . . 260
Shared-file server operating modes . . . 260
File locking in on-line mode . . . 261
Using the shared-file server . . . 262
Request reference . . . 264
Begin transaction (BT function) . . . 264
Close batch (CB function) . . . 264
Close on-line (CO function) . . . 265
Checkpoint (CP function) . . . 265
Close session (CS function) . . . 266
Delete record (DL function) . . . 266
End transaction (ET function) . . . 267
Request exclusive use (EX function) . . . 268
Fetch next record (FN function) . . . 268
Fetch previous record (FP function) . . . 269
Fetch unique record (FU function) . . . 271
Grant (GF function) . . . 272
Get next record (GN function) . . . 273
Get previous record (GP function) . . . 274
Get unique record (GU function) . . . 275
Habilitate (enable) logging (HL function) . . . 276
Hold next record (HN function) . . . 276
Hold previous record (HP function) . . . 277
Hold unique record (HU function) . . . 278
Initialize DBD (ID function) . . . 280
Inhibit logging (IL function) . . . 281
Initialize pointers (IP function) . . . 281
Insert record (IS function) . . . 282

Keep next record (KN function) . . . . . 283
Keep previous record (KP function) . . . . . 284
Keep unique record (KU function) . . . . . 284
Open batch (OB function) . . . . . 286
Open on-line (OO function) . . . . . 286
Open session (OS function) . . . . . 287
Query PCB (QP function) . . . . . 287
Rollback (RB function) . . . . . 288
Revoke (RF function) . . . . . 289
Read header (RH function) . . . . . 290
Replace record (RP function). . . . . 292
Statistic request (SR function) . . . . . 293
Test status (TS function) . . . . . 294
Erase shared-file data (ZD function) . . . . . 295
Examples of shared-file transactions . . . . . 296
Online transaction to search for information . . . . . 296
Online transaction to update one shared file. . . . . 296
Offline transaction . . . . . 296
Typical use of the shared-file server in on-line mode . . . . . 297
Major changes at file level . . . . . 298

**Chapter 15. Query server** . . . . . 301
Modes of operation . . . . . 302
Not-logged-on mode . . . . . 302
Shared-file mode. . . . . 302
Query mode . . . . . 302
Return code and explanation handling . . . . . 306
I/O structure blocks . . . . . 306
DBD definition . . . . . 307
PCB definition . . . . . 308
Short descriptor . . . . . 308
Long descriptor . . . . . 308
Pre-Fetch block . . . . . 309
Using the query server. . . . . 309
Using the query server in not-logged-on mode . . . . . 312
Request reference in not-logged-on mode or in any mode . . . . . 312
Close session (CS function) . . . . . 312
Grant (GF function) . . . . . 312
Habilitate (enable) logging (HL function) . . . . . 313
Inhibit logging (IL function) . . . . . 314
Open on-line (OO function) . . . . . 314
Open query mode (OQ function) . . . . . 315
Open session (OS function) . . . . . 315
Revoke (RF function) . . . . . 316
Test status (TS function) . . . . . 316
Using the query server in query mode. . . . . 317
Notes on the query server in query mode . . . . . 317
Supported data types . . . . . 318
Request reference for query server in query mode . . . . . 319
Close query mode (CQ function) . . . . . 319

Commit work (CW function) . . . 319
Describe query (DQ function) . . . 320
Delete row (DR function) . . . 321
Describe table (DT function) . . . 321
End query (EQ function) . . . 322
Fetch row (FR function) . . . 323
Insert row (IR function) . . . 323
Replace row (RR function) . . . 324
Rollback work (RW function) . . . 325
Set parameters (SP function) . . . 325
Structured query (SQ function) . . . 326
Using the query server in shared-file mode . . . 328
Concepts . . . 328
Shared-File mode in LANDP for OS/2 . . . 328
Data definition and data manipulation . . . 329
Compatibility with the shared-file server . . . 329
Request reference in shared-file mode . . . 331
Begin transaction (BT function) . . . 331
Close on-line (CO function) . . . 331
Checkpoint (CP function) . . . 332
Define DBD (DD function) . . . 332
Define index (DI function) . . . 333
Delete record (DL function) . . . 334
Define PCB (DP function) . . . 334
Erase DBD (ED function) . . . 335
Erase index (EI function) . . . 336
Erase PCB (EP function) . . . 336
End transaction (ET function) . . . 337
Request exclusive use (EX function) . . . 337
Get next record (GN function) . . . 338
Get previous record (GP function) . . . 339
Get unique record (GU function) . . . 340
Hold next record (HN function) . . . 341
Hold previous record (HP function) . . . 342
Hold unique record (HU function) . . . 342
Initialize DBD (ID function) . . . 343
Initialize pointers (IP function) . . . 344
Insert record (IS function) . . . 344
Keep next record (KN function) . . . 345
Keep previous record (KP function) . . . 346
Keep unique record (KU function) . . . 347
Rollback (RB function) . . . 348
Replace record (RP function) . . . 348
Statistics on DBD (SD function) . . . 349
Statistic request (SR function) . . . 350
Erase shared-file data (ZD function) . . . 351
Migrating data . . . 351

**Chapter 16. ODBC query server** . . . 353
Implications of using ODBC as an access path to the RDBMS . . . 354

Modes of operation . . . . . 354
Not-logged-on mode . . . . . 354
Query mode . . . . . 354
Return code and explanation handling . . . . . 356
I/O structure blocks . . . . . 357
Short descriptor . . . . . 357
Long descriptor . . . . . 357
Pre-Fetch block . . . . . 358
Using the ODBC query server . . . . . 358
Using the ODBC query server in not-logged-on mode . . . . . 360
Request reference in not-logged-on mode . . . . . 360
Change database (CD function) . . . . . 360
Close session (CS function) . . . . . 361
Grant (GF function) . . . . . 361
Habilitate (enable) logging (HL function) . . . . . 362
Inhibit logging (IL function) . . . . . 362
Open query mode (OQ function) . . . . . 362
Open session (OS function) . . . . . 363
Revoke (RF function) . . . . . 364
Test status (TS function) . . . . . 364
Using the ODBC query server in query mode . . . . . 365
Supported data types . . . . . 366
Request reference for ODBC query server in query mode . . . . . 368
Close query mode (CQ function) . . . . . 368
Commit work (CW function) . . . . . 368
Describe query (DQ function) . . . . . 369
Delete row (DR function) . . . . . 370
Describe table (DT function) . . . . . 370
End query (EQ function) . . . . . 371
Fetch row (FR function) . . . . . 372
Insert row (IR function) . . . . . 373
Obtain RDBMS information (RI function) . . . . . 374
Replace row (RR function) . . . . . 374
Rollback work (RW function) . . . . . 375
Set parameters (SP function) . . . . . 375
Structured query (SQ function) . . . . . 376

**Chapter 17. Electronic journal server** . . . . . 379
Electronic journal environments . . . . . 380
Physical electronic journals . . . . . 380
Logical electronic journals . . . . . 381
Data integrity of electronic journal files . . . . . 383
Search operations . . . . . 385
Relationship to the record format definitions . . . . . 385
Search definition . . . . . 385
Using the electronic journal server . . . . . 390
Obtaining database error messages . . . . . 392
Request reference . . . . . 392
Allocate physical electronic journal (AL function) . . . . . 392
Add record (AR function) . . . . . 392

Deallocate physical electronic journal (DA function) . . . 394
Delete record (DL function) . . . 394
Inquire logical (IL function) . . . 395
Inquire physical (IP function) . . . 396
Release electronic journal (RL function) . . . 398
Retrieve record (RR function) . . . 398
Reset electronic journal file (RS function). . . . 400
Select current electronic journal (SL function) . . . 401
Test initialization (TI function). . . . 402
Terminate session (TS function) . . . 402
Update record (UP function) . . . 403

**Chapter 18. Store-for-forwarding server** . . . 405
Storing records . . . 405
Processing stored records . . . 406
Data integrity of store-for-forwarding files . . . 406
Obtaining database error messages . . . 408
Search operations . . . 408
Using the store-for-forwarding server . . . 408
Request reference . . . 409
Add record (AR function) . . . 409
Set communication status (CS function) . . . 410
Delete record (DL function) . . . 411
Disable storing (DS function) . . . 412
Enable storing (EN function) . . . 412
Inquire status (IS function) . . . 413
Query file (QD function) . . . 414
Retrieve record (RR function) . . . 414
Reset store-for-forwarding file (RS function) . . . 417
Initiate store (SI function) . . . 417
Begin transmission (TB function) . . . 418
Stop transmission (TE function) . . . 419
Test initialization (TI function). . . . 420
Terminate session (TS function) . . . 420
Update record (UP function) . . . 420
Example of handling off-line situations. . . . 421
Forwarding using the LANDP forwarding server . . . 421
Example of local data collection for later forwarding . . . 422
Example of initiating local record storing . . . 422
Example of initiating locally collected data forwarding. . . . 422
Your own application program for forwarding . . . 423

**Part 5. Application integration servers** . . . 425

**Chapter 19. CICS interface server** . . . 427
Running with CICS . . . 427
CICS external call interface . . . 427
Synchronous and asynchronous server use . . . 428
Using the CICS interface server . . . 428

Request reference . . . . . 429
Call CICS (CC function) . . . . . 429
Read CICS (RC function) . . . . . 433

**Chapter 20. Dynamic data exchange access server** . . . . . 435
Using the DDE access server . . . . . 436
DDE access server high-level functions . . . . . 437
Request reference for high-level functions . . . . . 437
Get data (GD function) . . . . . 437
Insert data (ID function) . . . . . 438
Lock topic (LO function) . . . . . 439
Load program (LP function) . . . . . 439
Query applications (QA function) . . . . . 440
Run command (RC function) . . . . . 440
Unlock topic (UL function) . . . . . 441
DDE access server low-level functions . . . . . 442
Overview of low-level functions use . . . . . 443
Request reference for low-level functions . . . . . 443
Advise (AD function) . . . . . 443
Execute command (EC function) . . . . . 444
Initiate conversation (IC function) . . . . . 445
Peek data (KD function) . . . . . 446
Poke data (PD function) . . . . . 446
Request data (RD function) . . . . . 447
Terminate conversation (TC function) . . . . . 448
Unadvise (UN function) . . . . . 448
DDE access server clipboard functions . . . . . 449
Request reference for clipboard functions . . . . . 449
Copy text (CT function) . . . . . 449
Paste text (PT function) . . . . . 450
Examples using the high-level functions . . . . . 451
Example of using get data . . . . . 451
Example of using insert data . . . . . 451
Example of using run command . . . . . 451
Example of using query application . . . . . 451
Example of using load program . . . . . 452
Example of using lock topic . . . . . 452
Example of using unlock topic . . . . . 452
Examples using the low-level functions . . . . . 453
Example of inserting a data value . . . . . 453
Example of obtaining a data item value . . . . . 453
Example of performing a command . . . . . 454
Example of establishing a link . . . . . 455
Examples using the clipboard functions . . . . . 456
Example of copying text . . . . . 456
Example of pasting text . . . . . 456

**Part 6. System management servers** . . . . . 457

**Chapter 21. System manager server** . . . . . 459
User identification and control functions . . . . . 462
Change password (CP function). . . . . 463
Get signed-on user (GL function) . . . . . 463
Get signed-on user (Year-2000) (GI function) . . . . . 464
Get signed-on PC (GP function). . . . . 465
Get signed-on PC (Year-2000) (GW function) . . . . . 466
Retrieve defined users list (RE function) . . . . . 467
Remote sign-off (RF function) . . . . . 467
Retrieve list of users holding locks on LANDP resources (RK function) . . . . 468
Retrieve signed-on users list (RO function) . . . . . 469
Get signed-on users list (Year-2000) (GO function) . . . . . 470
Sign-Off (SF function) . . . . . 471
Sign-On (SN function) . . . . . 471
Sign-On (Year-2000) (SO function). . . . . 473
User profile and application program data maintenance . . . . . 474
Add user to user profile file (AU function). . . . . 476
Add user to user profile file (Year-2000) (AX function) . . . . . 476
Delete user in user profile file (DU function) . . . . . 477
Retrieve application program data (GU function) . . . . . 477
Retrieve user profile by record number (RN function). . . . . 478
Retrieve user profile by record number (Year-2000) (XN) . . . . . 479
Retrieve user profile (RU function) . . . . . 480
Retrieve user profile (Year-2000) (RX function). . . . . 481
Update application program data (SU function). . . . . 482
Update user profile (UU function) . . . . . 483
Update user profile (Year-2000) (UX function) . . . . . 484
Date and time synchronization . . . . . 484
Retrieve system date and time (GD function) . . . . . 485
Set system date and time (SD function) . . . . . 486
LANDP workgroup system status and common data maintenance . . . . . 487
Get system status (GG function) . . . . . 487
Retrieve LANDP workgroup common data (RD function) . . . . . 489
Update alerts status (UA function) . . . . . 489
Update LANDP workgroup common data (UD function) . . . . . 490
Update LANDP workgroup identification (UI function). . . . . 491
Update logging status (UL function) . . . . . 491
Update operator messages status (UM function) . . . . . 492
Update message operator receiver (UO function) . . . . . 492
Retrieval of defined record structures . . . . . 493
Retrieve record format (RR function) . . . . . 493
Data validation with defined record structures . . . . . 497
Validate record (VR function). . . . . 497
Alerts management . . . . . 498
Generation and transmission of alerts and resolutions . . . . . 498
Alerts notification (AN function) . . . . . 499
Send message to NetView operator (MO function) . . . . . 502
Resolution notification (UN function) . . . . . 503
EHCALRH user exit. . . . . 503
Check NMVT (CN function) . . . . . 503

System and user LOG management . . . . . 505
LOG file header . . . . . 505
LOG record format . . . . . 505
Error number and message tables . . . . . 507
Retrieve LOG record (RL function) . . . . . 507
Retrieve LOG record (R1 function) . . . . . 509
Write LOG record (WL function) . . . . . 510

**Chapter 22. Local resource manager** . . . . . 513
Status codes . . . . . 515
Communication status . . . . . 515
Pending communication function status . . . . . 515
Printer attending mode status . . . . . 515
Pending printer attending mode status . . . . . 516
Printer activity mode status . . . . . 516
Emulator attending status . . . . . 516
Application intervention status . . . . . 516
End listing control status . . . . . 517
Printer/Port status . . . . . 517
Using the local resource manager . . . . . 518
Functions relating to the IBM 3270 display/keyboard emulator . . . . . 519
End connection (EC function) . . . . . 519
Get status (GS function) . . . . . 520
Functions relating to the IBM 3287 printer emulator . . . . . 521
Activate application intervention (AI function) . . . . . 522
Activate service (AS function) . . . . . 523
Change printer assignment (CA function) . . . . . 524
Cancel pending functions (CC function) . . . . . 526
Change print density mode (CM function) . . . . . 528
Change LU priority (CP function) . . . . . 529
End operator intervention (EI function) . . . . . 531
End listing (EL function) . . . . . 533
Get status (GS function) . . . . . 534
Initiate session (IS function) . . . . . 536
Suspend service (SS function) . . . . . 537
Terminate session (TS function) . . . . . 539
Functions relating to the printer manager server . . . . . 541
Get status (GS function) . . . . . 541
Set alternate mode (SA function) . . . . . 542
Set exclusive mode (SE function) . . . . . 544
Functions relating to the banking printer program (BPP) . . . . . 546
Cancel pending functions (CC function) . . . . . 546
Get status (GS function) . . . . . 547
Change printer to remote mode (HP function) . . . . . 548
Change printer to local mode (LP function) . . . . . 549

**Part 7. Facilities and utilities** . . . . . 553

**Chapter 23. Running LANDP for DOS applications in OS/2 or Windows workstations** . . . . . 555
How the MVDM relay works on LANDP for OS/2 . . . . . 555
How the MVDM relay works on LANDP for Windows . . . . . 556
Sample files . . . . . 557
Configuration file . . . . . 557
Server name substitution . . . . . 558
Asynchronous events . . . . . 558
User exits . . . . . 559
Session release . . . . . 562
Auto connect . . . . . 562

**Chapter 24. Running LANDP for DOS applications in Linux workstations** 563
How the DosBox relay works on LANDP for Linux . . . . . 563
Sample files . . . . . 564
Configuration file . . . . . 564
Server name substitution . . . . . 565
Asynchronous events . . . . . 565
Session release . . . . . 567
Auto connect . . . . . 567

**Chapter 25. ASCII-EBCDIC translation** . . . . . 569
Double-byte character sets . . . . . 570
ASCII-EBCDIC translation server request reference . . . . . 571
ASCII-EBCDIC translation (AE function) . . . . . 571
EBCDIC-ASCII translation (EA function) . . . . . 572
Translation routines between ASCII and EBCDIC . . . . . 573
Routines _AE and _EA . . . . . 573
Routines AE and EA . . . . . 574

**Chapter 26. IBM 4707 monochrome display** . . . . . 575
Changing display mode using the IBM LANDP-supplied routine . . . . . 575

**Chapter 27. Object postbox server and batch machines** . . . . . 577
Server characteristics . . . . . 577
Using the object postbox server . . . . . 578
Request reference . . . . . 580
Backout operation (BO function) . . . . . 580
List input queue (LI function) . . . . . 580
List output queue (LO function) . . . . . 581
Register device and check status (RD function) . . . . . 583
Read message (RM function) . . . . . 584
Send message and message header (SM function) . . . . . 584
Write message data (WM function) . . . . . 585
Examples of use . . . . . 586
Batch machines . . . . . 587
Building your programs for batch processing . . . . . 588
Sample programs that use batch processing . . . . . 588

**Part 8. IBM LANDP emulators** . . . . . 591

**Chapter 28. LANDP 3270 emulator API** . . . . . 593
Characteristics of the low-level language API . . . . . 593
Invocation of the API . . . . . 593
API services . . . . . 594
Query gate ID . . . . . 595
Session information services . . . . . 595
Keyboard services . . . . . 599
Copy service . . . . . 606
Operator information area service . . . . . 609
Example using 3270 emulator API . . . . . 610

**Chapter 29. LANDP 3270 emulator high-level language API** . . . . . 613
Characteristics of the high-level language API . . . . . 613
LANDP HLLAPI control flow . . . . . 614
LANDP HLLAPI function calls . . . . . 617
High-level language application programming interface services . . . . . 618
Operator services . . . . . 618
Query system (function 20) . . . . . 619
Query sessions (function 10) . . . . . 620
Query session status (function 22) . . . . . 621
Set session parameters (function 9) . . . . . 622
Send key (function 3) . . . . . 624
Wait (function 4) . . . . . 625
Pause (function 18) . . . . . 626
Start host notification (function 23) . . . . . 627
Query host update (function 24) . . . . . 628
Stop host notification (function 25) . . . . . 629
Reset system (function 21) . . . . . 630
Presentation services . . . . . 630
Connect to presentation space (function 1) . . . . . 630
Disconnect from presentation space (function 2) . . . . . 632
Search presentation space (function 6) . . . . . 632
Query cursor location (function 7) . . . . . 634
Query field attribute (function 14) . . . . . 634
Search field (function 30) . . . . . 635
Find field position (function 31) . . . . . 636
Find field length (function 32) . . . . . 637
Set cursor (function 40) . . . . . 638
Copy services . . . . . 639
Copy string to field (function 33) . . . . . 639
Copy field to string (function 34) . . . . . 640
Copy OIA (function 13) . . . . . 641
Copy presentation space (function 5) . . . . . 643
Copy presentation space to string (function 8) . . . . . 644
Copy string to presentation space (function 15) . . . . . 645
Device services . . . . . 646
Reserve (function 11) . . . . . 646
Release (function 12) . . . . . 647

Communication services . . . . . 648
Send file (function 90) . . . . . 649
Receive file (function 91) . . . . . 651
Utility services . . . . . 652
Convert position or rowcol (function 99) . . . . . 652
Compatibility with other IBM 3270 emulator application programming interfaces 653
Portability considerations . . . . . 653
3270 extended data stream . . . . . 655

**Chapter 30. LANDP 3287 printer emulator API** . . . . . 657
IBM 3287 printer emulator characteristics . . . . . 657
Accessing the IBM 3287 printer emulator. . . . . 658
Facilities not supported . . . . . 659
Host computer communication . . . . . 659
SNA character string control codes. . . . . 660
User exits . . . . . 661
Writing the EXFS server . . . . . 662
Data received (DR function) . . . . . 663
End listing (EL function) . . . . . 664
Print data (PD function) . . . . . 664

**Part 9. Appendixes** . . . . . 667

**Appendix A. Connectivity programming request block** . . . . . 669

**Appendix B. Switch printer mode within an application (Z6 function)** . . . . 671

**Appendix C. Notices** . . . . . 673
Trademarks and service marks . . . . . 674

**Glossary** . . . . . 677

**Bibliography** . . . . . 703
IBM LANDP Family . . . . . 703
IBM Financial Branch System Services Licensed Programs . . . . . 703
IBM Financial Branch System Integrator Licensed Programs . . . . . 703
IBM Transaction Security System . . . . . 703
Banking Self-Service . . . . . 703
IBM workstations. . . . . 703
IBM Local Area Network . . . . . 704
IBM 3270 . . . . . 704
Wide Area Communications . . . . . 704
IBM NetView . . . . . 705
IBM Financial I/O Devices. . . . . 705
Encryption and Decryption . . . . . 706
IBM VisualAge C++ . . . . . 706
IBM VisualAge Generator . . . . . 706
IBM VisualAge Smalltalk . . . . . 707
Java . . . . . 707

IBM Personal Communications . . . . . 707
IBM Communications Server . . . . . 707
WorkSpace On-Demand . . . . . 707
MQSeries . . . . . 707

**Index** . . . . . 709

# Figures

1. Encryption and Decryption of Data Using a Key . . . . . 75
2. Encryption Using Cipher Block Chaining . . . . . 77
3. Decryption to Recover the Original Data . . . . . 77
4. Query Server Components and their Interaction with Applications and Databases . . . . . 301
5. ODBC query server Components and their Interaction with Applications and Databases 353
6. Component Interaction of LANDP HLLAPI . . . . . 615

© Copyright IBM Corp. 1992, 2002

# Tables of Function Codes


1. Function Codes used in the Supervisor Local Functions . . . 3
2. Reply DATA values when 'T' entered in Request PARMLIST . . . 15
3. Reply DATA values when 'L' entered in Request PARMLIST . . . 15
4. Function Codes used in the SNA Communication Server . . . 36
5. Function Codes used in the Compression Server . . . 36
6. Function Codes used by the Cryptographic Interface . . . 76
7. Function Codes used with the Native X.25 Communication Server . . . 91
8. Function Codes used in the Program-to-program Communication Server . . . 105
9. Function codes used in the LANDP MQSeries Link server. . . . 120
10. CPRB fields on request. . . . 120
11. CPRB fields on reply . . . 121
12. Reply PARMLIST values . . . 124
13. Reply PARMLIST values . . . 124
14. Reply PARMLIST values . . . 125
15. Request PARMLIST values . . . 126
16. Reply PARMLIST values . . . 126
17. Reply PARMLIST values . . . 127
18. Request PARMLIST values . . . 127
19. Reply PARMLIST values . . . 128
20. Reply data values . . . 128
21. Request PARMLIST values . . . 131
22. Request DATA values . . . 131
23. Reply PARMLIST values . . . 132
24. Request PARMLIST values . . . 135
25. Request DATA values . . . 135
26. Reply PARMLIST Values . . . 135
27. Reply PARMLIST Values . . . 136
28. Function Codes used in the Financial Printer Server . . . 141
29. Function Codes used in the 4748 Printer Server . . . 167
30. Function Codes used in the 4770 Printer Server . . . 189
31. Function Codes used in the Printer Manager Server . . . 199
32. Function Codes used in the MSR/E Server . . . 204
33. Function Codes used in the PIN Pad Server . . . 224
34. Function Codes used in the Shared-File Server . . . 255
35. Function Codes used in the Query Server . . . 304
36. Function Codes used in the ODBC query server . . . 356
37. Data types allowed in structured query. . . . 366
38. Function Codes used in the Electronic Journal Server . . . 379
39. Function Codes used in the Store-for-Forwarding Server . . . 405
40. Function Codes used in the CICS interface server . . . 427
41. Function Codes used in the DDE Access Server . . . 435
42. Function Codes used in the System Manager Server . . . 459
43. Function Codes used in the Local Resource Manager . . . 514
44. Function Codes used in the ASCII–EBCDIC Translation Server . . . 569
45. Function Codes used in the Object Post-Box Server . . . 577
46. Function Codes used in the Local Resource Manager—functions used with the 3270 emulator low-level interface . . . 593


© Copyright IBM Corp. 1992, 2002

47. Function Codes used in the LANDP 3270 emulator HLL API . . . . . 613
48. Function Codes used with the 3287 printer emulator . . . . . 658
49. Function Codes used in the EXFS User-written Exit Server . . . . . 662
50. Trademarks and service marks . . . . . 674

# About this book

This book provides information about the following IBM® LAN Distributed Platform (LANDP®) Family products:

- LANDP Family Version 6.0 with its components:
  - LANDP for DOS
  - LANDP for OS/2®
  - LANDP for Windows NT
  - LANDP for Linux

**Note:** This book no longer provides information about IBM LANDP for AIX®. Refer to the previous edition of this book, *LANDP Family Programming Reference Version 5.0*, SC34-5531-00, for information about this product.

This book provides information on how to design and develop user servers and applications for a LANDP environment.

## Who should read this book

This book is written for system analysts, system programmers, and application programmers.

Individuals responsible for host computer systems and host computer applications can find useful information on how to include LANDP workstations systems in wide area networks.

## What you need to know

You should be familiar with the operating systems that support your LANDP environment and the programming language you are using.

Also, if you are involved in the development of application programs using wide area communications, you should be familiar with System Network Architecture (SNA) protocols and Synchronous Data Link Control (SDLC), X.25 Data Link Control, or Token-Ring Data Link Control.

## How this book is organized

The book is divided into the following parts.

Part 1, “Supervisor” contains information on how to program in a LANDP environment. The supervisor local functions are presented in this part.

Part 2, “Wide area communication servers” presents the server functions provided by the LANDP family to enable communication between application programs in a LANDP workgroup and typically with those in a host computer.

© Copyright IBM Corp. 1992, 2002

Part 3, “I/O device servers” presents the server functions provided by the LANDP family to manage the input/output (I/O) devices that you can attach to the workstations in a LANDP workgroup.

Part 4, “Data management servers” presents the server functions provided by the LANDP family to manage data in a LANDP workgroup.

Part 5, “Application integration servers” presents the server functions provided by the LANDP family to enable access to non-LANDP environments and simplify resource and data sharing with non-LANDP programs.

Part 6, “System management servers” presents the server functions provided by the LANDP family to manage the system and the common services for all workstations in the LANDP workgroup.

Part 7, “Facilities and utilities” presents some miscellaneous servers and utility programs provided by the LANDP family.

Part 8, “IBM LANDP emulators” presents the application programming interfaces provided with the LANDP for DOS emulators.

The appendixes summarize the contents of the Connectivity Programming Request Block (CPRB) and the function codes used by LANDP servers. A bibliography, glossary, and index are provided at the back of the book.

## Conventions used in this book

This book uses a conventional notation for the system environments in which LANDP server functions operate, for the values of function codes, return codes, and event codes, and for describing fields that contain blanks.

### Operating environments

Throughout the book there are tables that indicate the functions that are supported by each IBM-supplied LANDP server. Not all functions operate in all platforms, and the tables use the following convention to show the operating environment:

**0** LANDP for DOS
**2** LANDP for OS/2
**N** LANDP for Windows
**L** LANDP for Linux

For example, “`-2N-`” denotes a function available from a LANDP server that runs in the LANDP for OS/2, LANDP for Windows environments, but is not available on LANDP for DOS or LANDP for Linux.

Also, the same information is given in words for each function request, or a global statement is made at the start of the chapter that describes the server.

## Function, return, and event codes

References made in this book to the contents of the following fields use a symbolic notation to simplify the understanding of the actual values that these fields contain:

- Function code
- Router return code
- Server return code
- Asynchronous event code

**Function codes and asynchronous event codes**

are unsigned two-byte integers. Their values have been chosen so that if you represent their numeric value in ASCII form, you get a significant string.

For example, the value X'494E' represents the function code **IN**.

| Value | Symbol |
|---|---|
| X'49' | I |
| X'4E' | N |

On **Intel Machines**, because the function code field is a two-byte integer and stored in *Intel reversed format*, the field contains X'4E' in the first position and X'49' in the second memory position.

**Router and server return codes**

are unsigned four-byte integers. To get the symbol for the numeric value, apply the previous rules to the two least significant bytes. To convert a symbol into a numeric value, use the same rule as for the function code and the asynchronous event code to get the two least significant bytes. Examples are given in the next table.

| Examples of LANDP Return Codes | | |
|---|---|---|
| **Value** | **Two least significant bytes** | **Symbol** |
| X'01004C38' | 4C38 | L8 |
| X'01005031' | 5031 | P1 |

## Bit positions

The highest order bit in a byte is labelled bit 7, and the lowest bit 0.

## DB2 Universal Database®

In this book, all references to DB2® or DB2/2 apply to IBM DB2 for OS/2 and to IBM DB2 Universal Database®.

## Windows

In this book, all references to Windows apply to Microsoft Windows NT, Microsoft Windows 2000, and Microsoft Windows XP.

## Linux

In this book, all references to Linux apply to RedHat, SuSE, TurboLinux, and Caldera distributions of Linux, unless indicated otherwise.

File names in Linux will usually be lowercase versions of those shown in this book. In addition, for executable Linux files (for example, the LANDP servers), there are no filename extensions. For example, the LANDP for Windows TCP/IP wide area communication server is EHCTCP.EXE and the LANDP for Linux TCP/IP wide area communication server is *ehctcp*.

## Related Information

The LANDP family is supported by the following books. In this book, references to other LANDP books use the shortened title shown here. For the full title, order number of these publications, an comprehensive list of LANDP-related literature, refer to "Bibliography" on page 703.

*LANDP Introduction and Planning*
: This book provides a brief description of the components and features of the LANDP family, and gives information about planning a LANDP system.

*LANDP Installation and Customization*
: This book provides information about installing, customizing, and distributing the LANDP family.

*LANDP Programming Reference*
: This book describes the application programming interfaces that are used to develop user servers and client applications.

*LANDP Programming Guide*
: This book gives guidance on writing application programs to use the interfaces described in the *LANDP Programming Reference*.

*LANDP Problem Determination*
: This book describes how to use trace tools, diagnostic programs, alerts, and return codes to debug code while developing LANDP applications and user servers, or resolve problems while using LANDP family components.

*LANDP Servers and System Management*
: This book provides detailed information on the LANDP servers, and describes how to manage and administer a LANDP system.

## Web site

For more information about LANDP please visit our web site at:

**http://www.ibm.com/software/ts/landp/**

# Summary of Changes

This manual has been updated to reflect enhancements made to LANDP in Version 6. The major changes in this version are:

- The removal of support of LANDP for AIX.
- The addition of support for Linux. LANDP for Linux can run on any of the following Linux distributions:
  - Red Hat
  - Caldera
  - SuSE
  - TurboLinux
- The addition of Java wrappers and Java Beans for the most popular LANDP servers.
- The addition of support for mixed NetBIOS and TCP/IP LAN transports. This enables easy migration of workgroups from NetBIOS to TCP/IP.
- The addition of the LANDP link server. This enables clients in one LANDP workgroup to use servers in another LANDP workgroup.
- The use of InstallShield to make it easier to install LANDP.
- The addition of a new graphical editor for customization.

© Copyright IBM Corp. 1992, 2002

# Part 1. Supervisor

This part describes the functions of the LANDP supervisor, which can be used by LANDP client programs. It contains one chapter:

**Chapter 1, “Supervisor local functions” on page 3**
This chapter describes how to request services in the local supervisor from the LANDP client/server mechanism.

© Copyright IBM Corp. 1992, 2002

# Chapter 1. Supervisor local functions

LANDP provides, through the client/server mechanism, a set of *supervisor local functions* that clients can use to:

- Manage asynchronous events
- Manage OS/2, Linux and Windows semaphores
- Control servers
- Set and terminate timers
- Enter an emulator session

Except where stated, all functions operate in a **LANDP for DOS, LANDP for OS/2, LANDP for Linux, or LANDP for Windows** environment on the workstation in which the client application is running.

*Table 1. Function Codes used in the Supervisor Local Functions*

| Function code | Description | Env. | Page |
|---|---|---|---|
| AA | Ask for asynchronous events | 02NL | 6 |
| AS | Associate semaphore | -2NL | 7 |
| DS | Dissociate semaphore | -2NL | 9 |
| EE | Simulate hot-key | 0--- | 10 |
| EJ | Disconnect an application program | 02NL | 11 |
| ES | Unload LANDP | 02NL | 12 |
| II | Inquire information | -2NL | 14 |
| IN | Initialize | 02NL | 16 |
| QE | Query event | 02N- | 17 |
| SI | System status information | 02NL | 18 |
| SP | Start posting events | 02N- | 20 |
| T0–T8 | Activate and deactivate timers | 02NL | 21 |
| TP | Stop posting events | 02N- | 23 |
| WM | Wait for asynchronous events | 02NL | 24 |
| Z4 | Asynchronous request — asking for TT request | 02NL | 29 |
| Z5 | Asynchronous request — event notification or cancellation | 02NL | 30 |
| ZN | Extended asynchronous event notification | -2NL | 31 |

The first and second columns give the function code and the name of the function.

The third column shows the operating environment of the function, as explained in “Operating environments” on page xxvi. “02NL” means that it is available from LANDP for DOS, LANDP for OS/2, LANDP for Windows, or LANDP for Linux supervisors.

The last column refers to the page where you can find the function described.

© Copyright IBM Corp. 1992, 2002

## Using the supervisor local functions

This section provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields. If you need more information about the CPRB fields, see Appendix A, “Connectivity programming request block” on page 669.

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | 26 | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 26 | 2 | 26 | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | SPV | Server name |

The following fields are variable and are discussed in each function call description:

- Function code
- Request DATA length
- Reply DATA length

The CPRB fields returned from the supervisor local functions are:

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 44 | 2 | | Replied PARMLIST length |
| 46 | 2 | | Replied DATA length |

If the request was successful, the *router return code* and the *server return code* are both X'00000000'. In all other cases, see the appropriate section in the *LANDP Problem Determination* book, to see whether you should take any action. If the request was not successful the values returned in the Reply PARMLIST and Reply DATA fields should be ignored.

The following fields are variable and are discussed in each function call description:
- Replied PARMLIST length
- Replied DATA length

**PARMLIST:** The Request PARMLIST and Reply PARMLIST fields for the local functions are discussed in the description of each local function request.

**DATA:** The Request DATA and Reply DATA areas for the local functions are discussed in the description of each local function request.

## Linux semaphore-handling

Linux provides a choice of two semaphore-handling mechanisms, both of which are different from that provided by OS/2 and Windows. One is based on the System V specification and one is based on the POSIX specification. LANDP is providing support for the System V version of semaphores.

System V semaphore-handling works in the following way. The semaphore is indicated by two identifiers, the first being the System V semaphore set identifier and the second being the number of the semaphore within the set. Each semaphore set that is shared between unrelated processes is associated with a key. This key is a unique identifier that acts in the same way as a name does on OS/2 or Windows.

LANDP supports only semaphores that have been set up to be shared between unrelated processes. System V semaphores are essentially counters that you can increment or decrement depending on the operation. In order to provide the 'wait and post' method of operation that LANDP uses, the following behaviour is required:
- The value of the semaphore indicates the state of the semaphore. A semaphore is posted if the value is 0, and a semaphore is in Reset (a non-signalled state) if the value is 1.
- All wait operations wait for the semaphore value to be 0.
- When associating a System V semaphore with a LANDP semaphore, both the semaphore key and the number within the set associated with the key, are required.

Refer to your Linux manuals for a definition of the System V Semaphores. Related functions are semget, semop and semctl.

## Request reference

This section describes the functions that client programs can request from the local LANDP supervisor. They are listed in alphabetical order of function code.

## Ask for asynchronous events (AA function)

The client uses the **AA** function to obtain a list of pending asynchronous events. These events can be from the keyboard, the host computer, the I/O devices, user servers, or any set timer. The LANDP for OS/2 and LANDP for Windows supervisors also provide mouse and semaphore events. The LANDP for Linux supervisor provides semaphore events, but not mouse events.

The **AA** function allows you to ask for pending asynchronous events without entering a wait multiple state. For background information on the use of events, see the "LANDP event notification support" section in the "Writing client programs" chapter of the *LANDP Programming Guide*.

- If the **AA** function does not return any asynchronous event, the application program can either continue processing whatever is required, or it can issue the **WM** function if no further processing is needed.
- The **AA** function returns a list of pending events, but it does not remove an event from the event queue. Use the **WM** function to remove the event from that queue.

**Note:** Events generated by the communication servers are handled differently, see "Wait for asynchronous events (WM function)" on page 24.

| CPRB Field | Content/Description |
|---|---|
| Function code | AA |
| Request DATA length | 0 to 1020 (102 entries of 10 bytes each) |
| Request PARMLIST length | 0 |
| Reply DATA length | 10 to 1020 (102 entries of 10 bytes each) |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 to 1020 |
| Replied PARMLIST length | 0 |

| Request DATA Values | | | |
|---|---|---|---|
| **Offset** | **Length** | **Content** | |
| 0 | 2 | **X'nnnn'** | Asynchronous event code to ask for |
| 2 | 8 | **SERVNAME** | Name of the server that originates the asynchronous event (uppercase characters, left-justified, and padded with blanks) |
| 10 | 2 | **X'nnnn'** | Asynchronous event code to ask for |

| Request DATA Values | | | |
|---|---|---|---|
| **Offset** | **Length** | **Content** | |
| 12 | 8 | **SERVNAME** | Name of the server that originates the asynchronous event (uppercase characters, left-justified, and padded with blanks) |
| **Note:** Request DATA is optional. The pattern above shows two entries. If you need to define more entries, concatenate them without any delimiter, up to a maximum of 102 entries. | | | |

If Request DATA length is set to X'0000', the **AA** function does not specify any input list. Here the **AA** function returns the list of all the pending asynchronous events. The possible asynchronous events are described in “Wait for asynchronous events (WM function)” on page 24.

| Reply DATA Values | | | |
|---|---|---|---|
| **Offset** | **Length** | **Content** | |
| 0 | 2 | **X'nnnn'** | Pending asynchronous event code |
| 2 | 8 | **SERVNAME** | Name of the server that originates the asynchronous event |
| 10 | 2 | **X'nnnn'** | Pending asynchronous event code |
| 12 | 8 | **SERVNAME** | Name of the server that originates the asynchronous event |
| **Note:** The pattern above shows two entries. It can be repeated up to a maximum of 102 entries. | | | |

If no event is pending, the Replied DATA length is 0.

## Associate semaphore (AS function)

The **AS** function operates in the LANDP for OS/2, LANDP for Windows, or LANDP for Linux environments. Application programs can use events (a multiple-tasking facility) to solve problems of synchronizing and signalling between concurrent threads or processes.

A client, operating in 32-bit processing mode, can use the **AS** function to associate an OS/2, Windows, or Linux event with a LANDP event. For background information on the use of events, see the “LANDP event notification support” section in the “Writing client programs” chapter of the *LANDP Programming Guide*.

The LANDP semaphore ID is a 2-byte value. This value is used to build the event ID that is used with the supervisor functions **WM** and **SP**.

The event ID consists of the concatenation of the LANDP semaphore ID and “SPV ” (with five blanks). Event IDs can be used with the WM or SP event lists so that the client can be notified when the OS/2, Windows, or Linux event has been posted.

**Notes:**

1. Windows: Events and semaphores are separate types of synchronization objects under Windows. The **AS** function works with Windows events, but not with semaphores.
2. Linux: For LANDP for Linux, only the System V semaphore-handling mechanism is supported. Refer to “Linux semaphore-handling” on page 5 for further information.

| CPRB Field | Content/Description |
|---|---|
| Function code | AS |
| Request DATA length | 4 or 8 |
| Request PARMLIST length | 0 or 1 |
| Reply DATA length | 2 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 2 |
| Replied PARMLIST length | 0 |

**Note:** If the Request PARMLIST length is 0, the Request DATA length must be 4. If the Request PARMLIST length is 1, the Request DATA length must be 4.

| Request DATA Values (OS/2 and Windows) | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 4 | Semaphore handle<br><br>(Provided by the operating system when creating the semaphore) |

| Request DATA Values (Linux) | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 4 | Associated System V semaphore key |
| 4 | 4 | Semaphore number within the System V semaphor set |

| Request PARMLIST Values (Linux) | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | ’0’ |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | LANDP semaphore event |

## Dissociate semaphore (DS function)

The **DS** function operates in the LANDP for OS/2, LANDP for Windows, or LANDP for Linux environments.

The client, operating in 32-bit processing mode, uses the **DS** function to disassociate an OS/2, Windows, or Linux event from a LANDP event—see also “Associate semaphore (AS function)” on page 7. For background information on the use of events, see the “LANDP event notification support” section in the “Writing client programs” chapter of the *LANDP Programming Guide*.

| CPRB Field | Content/Description |
|---|---|
| Function code | DS |
| Request DATA length | 2 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | LANDP semaphore ID |

## Simulate hot-key (EE function)

The **EE** function operates in the LANDP for DOS environment. It allows a client to start:

- One of the 3270 emulators
- Operator interface
- System manager operator interface
- Trace tools

It allows the application program to present to the workstation user, for example, a 3270 emulator panel.

The effect of the **EE** function is the same as if the user had pressed the hot-key. This means that, after issuing the call to the client/server mechanism, the application program becomes inactive and the specified program receives control. When the operator leaves the started program, the application program regains control.

| CPRB Field | Content/Description |
|---|---|
| Function code | EE |
| Request DATA length | 1 or 8 (see below) |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

### DATA values

There are two ways to use this function:

1. Specifying in Request DATA the hot-key scan code associated with the program you want to enter. Request DATA length must then be X'0001'.

   **Note:** When the BIOS video mode is not alphanumeric (TEXT), that is, 0, 1, 2, 3, or 7, the hot-key call is cancelled, and the alarm sounds to alert the workstation operator. The contents of the panel might be lost if the video mode is 0 or 1 and the page number is nonzero.

2. Specifying in Request DATA one of the following:
   - 3270 emulator long name
   - 'OPER' (operator interface)
   - 'SMOP' (system manager operator interface)
   - 'DDT' (trace tools)

   These names are eight bytes long (padded with blanks), so the Request DATA length is X'0008'.

## Disconnect an application program (EJ function)

The **EJ** function is used by the client to disconnect from LANDP services. It can also be used by a LANDP for OS/2, LANDP for Windows, or LANDP for Linux server to unregister its service names. You should note that:

- To disconnect from LANDP, a client must set the CPRB originator resource name field to:
  - All blanks
  - Zeros
  - The filename of the client (up to eight characters, padded (if necessary) with trailing blanks
- To unregister one of its service names, a LANDP for OS/2, LANDP for Windows, or LANDP for Linux server must set the service name in the CPRB originator resource name field.

After the application program issues an **EJ** function request, it cannot send requests to LANDP until it has issued the next **IN** function request.

It is strongly recommended that you request the **EJ** function when the LANDP resources are no longer needed. If a LANDP for OS/2, LANDP for Windows, or LANDP for Linux application ends abnormally without having requested the **EJ** function, resources are freed automatically by LANDP for OS/2, LANDP for Windows, or LANDP for Linux.

After unregistering a service name, a LANDP server does not receive further requests for this service until it has issued a `SRVINIT` call for the service.

| CPRB Field | Content/Description |
|---|---|
| Function code | EJ |
| Request DATA length | 0 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |
| Originator resource name | All blanks, zeros, or the filename of the client (to disconnect from LANDP), or a valid service name (that is to be unregistered) |

## Unload LANDP (ES function)

The client either terminates by unloading LANDP and all servers, or unloads one local server only from the computer from which the **ES** function request was issued.

If you want to unload LANDP and all servers, set Request DATA length to zero or enter 'SPV ' in Request DATA.

If you want to unload a specific LANDP for OS/2, LANDP for Windows, or LANDP for Linux server, you can unload the server using one of the following:

- The server executable file name.

  The exec name must be specified in Request DATA.
- Any of the service names for the server.

  A valid service name (one of the service names registered by the server) must be specified in Request DATA.
- The server's PID.

  The byte at Offset 1 in Request PARMLIST must be set to 'P', and a valid LANDP PID must be specified in Request DATA.

When using a name (exec or service name), the **ES** process first checks whether a server exists that has registered this service name. If such a server does exist, it is unloaded.

If no server exists that has registered this service name, the **ES** process then checks whether the name matches the exec name for one or more servers, and unloads all servers with this exec name.

It is not possible to unload individual servers.

Additional parameters allow you to define two modes for unloading a server or LANDP:

- *Forced mode*, causes the client/server mechanism to terminate the process as specified using OS/2 functions.
- *Weak mode*, causes the client/server mechanism to unload the server using ordinary LANDP functions. This is the default mode that is assumed if either the Request PARMLIST length, or the Request PARMLIST address, is 0.

| CPRB Field | Content/Description |
|---|---|
| Function code | ES |
| Request DATA length | 0, 2, or 8 |
| Request PARMLIST length | 0, 1, or 2 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Mode used for unloading a server:<br>**W** Unload server in weak mode<br>**F** Unload server in forced mode |
| 1 | 1 | Data type used for unloading a server:<br>**P** Request DATA area contains a LANDP PID |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 or 8 | LANDP PID (for length 2), or<br>Exec or service name (for length 8) |

**Note:** If you use the ES function in a workstation that has servers accessed by other workstations in the LANDP workgroup, the other workstations lose the possibility to access these servers.

## Inquire information (II function)

The **II** function provides information about LANDP status. It operates in the LANDP for OS/2, LANDP for Windows, or LANDP for Linux environments. (See also “System status information (SI function)” on page 18 for a similar function.)

Depending on the parameter value that is entered in Request PARMLIST, different values are returned in Reply DATA. When parameter ’L’ is entered, the *PC ID* and *LAN ID* are returned in Reply DATA. When parameter ’T’ is entered, the *Load Table* is returned in Reply DATA.

**Note:** The EHCINF.EXE program gives the same output as the **II** function. The information returned when the ’T’ parameter is entered is the same as the information displayed by the LANDP `EHCINFO` program. `EHCINFO` is a utility program, described in the *LANDP Servers and System Management* book.

| CPRB Field | Content/Description |
|---|---|
| Function code | II |
| Request DATA length | 0 |
| Request PARMLIST length | 1 |
| Reply DATA length | 10 (for PARMLIST value ’L’) or 25 (for PARMLIST value ’T’) to 57499 |
| Reply PARMLIST length | 0 or 2 |
| Replied DATA length | 0 to 57499 |
| Replied PARMLIST length | 0 or 2 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | **L** Obtain *PC ID* and *LAN ID*<br>**T** Obtain *Load Table* |

| Reply PARMLIST values when ’T’ entered in Request PARMLIST | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Length of *Load Table* |

If the parameter 'L' is used, no Reply PARMLIST is returned.

*Table 2. Reply DATA values when 'T' entered in Request PARMLIST*

| Reply DATA values when 'T' entered in Request PARMLIST | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Router flag<br>**R** This process can be requested by both local and remote workstations<br>**L** This process is loaded locally and is accessible only by remote workstations |
| 1 | 8 | Server name |
| 9 | 8 | Server alias |
| 17 | 2 | PC ID |
| 19 | 1 | Operating system |
| 20 | 1 | Status<br>**0** This process is offline<br>**1** This process is online |
| 21 | 4 | Process ID |
| **Note:** The above pattern is repeated for each server application that is loaded | | |

*Table 3. Reply DATA values when 'L' entered in Request PARMLIST*

| Reply DATA values when 'L' entered in Request PARMLIST | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | PC ID |
| 2 | 8 | LAN ID |

## Initialize (IN function)

The **IN** function allows a client to connect to the LANDP services. **IN** must be the first function request issued.

After the application program issues the **IN** function request, the workstation ID assigned to this workstation is provided in the CPRB. The program version is also returned in the Reply PARMLIST.

If, during customization, the access authorization level F is specified, using the LEVELF keyword, the **IN** function and the **EJ** function (see 11) are the only ones that can be used without a previous sign-on. See Chapter 21, “System manager server” on page 459 and the *LANDP Installation and Customization* book for more information.

| CPRB Field | Content/Description |
|---|---|
| Function code | IN |
| Request DATA length | 0 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 6 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 6 |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 5 | Program version: version, release, modification and PTF level |
| 5 | 1 | LANDP program:<br>’0’ = LANDP for DOS<br>’1’ = LANDP for OS/2<br>’4’ = LANDP for Windows<br>’5’ = LANDP for Linux |

## Query event (QE function)

The **QE** function operates in the LANDP for DOS, LANDP for OS/2, or LANDP for Windows environments.

Using this function, an OS/2 PM, Windows 3.1/3.11, or Windows NT graphical user interface application program can obtain the 10-byte *event ID* of the event that caused an OS/2 message to be posted by LANDP. For background information on the use of events, see the “LANDP event notification support” section in the “Writing client programs” chapter of the *LANDP Programming Guide*.

The information received with a LANDP event OS/2 or Windows message is the message identity associated with this event in the SP function, plus an event handle that is returned in the low word of the *message parameter 1*. If the complete event ID is needed, this function should be used to obtain it.

It is recommended that your application program calls the **QE** function from inside its message processing routine.

| CPRB Field | Content/Description |
|---|---|
| Function code | QE |
| Request DATA length | 2 |
| Request PARMLIST length | 8 |
| Reply DATA length | 10 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 10 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 4 | Window handle that received the OS/2 or Windows message that is to be queried. |
| 4 | 4 | OS/2 message identity. |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Event handle. (Received with the low word of the OS/2 or Windows message *parameter 1* by the client window procedure.) |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 10 | Event ID that caused the message to be posted |

## System status information (SI function)

The **SI** function provides information about the services that are available to the local workstation, and about locally provided processes that are available to local or remote clients. (See also “Inquire information (II function)” on page 14 for a similar function.)

**Note:** In LANDP for DOS, the EHCCINFO.EXE program gives the same output as the SI function.

| CPRB Field | Content/Description |
|---|---|
| Function code | SI |
| Request DATA length | 0 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 to 57498 (but allocate enough space for number of processes) |
| Reply PARMLIST length | 2 |
| Replied DATA length | 0 to 57498 |
| Replied PARMLIST length | 2 |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Number of processes returned in the Reply DATA area |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| For each process: | | |
| 0 | 1 | **Identifier flag**:<br><br>Bit 7 1=service<br>0=application<br><br>Bit 6 1=service accessible from local requesters<br>0=service not accessible from local requesters<br><br>Bit 0 service status<br>1=online<br>0=offline |
| 1 | 8 | Process name (service or application) |
| 9 | 8 | Alias name |
| 17 | 2 | Process identifier (PID) |
| 19 | 2 | Process location (PC ID) |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 21 | 1 | Operating system:<br>**'0'** DOS<br>**'1'** OS/2<br>**'4'** Windows<br>**'5'** Linux |
| 22 | 2 | Reserved |
| Followed by (where N is the number of processes multiplied by 24): | | |
| N | 2 | PC ID (local workstation identification) |
| N+2 | 1 | Local PC type:<br>**'0'** DOS<br>**'1'** OS/2<br>**'4'** Windows<br>**'5'** Linux |
| N+3 | 8 | LAN ID |
| N+11 | 6 | Supervisor version level (as in the reply to IN) |
| N+17 | 2 | FBSS-compatible supervisor version level |
| N+19 | 6 | First adapter physical address (MAC) |
| N+25 | 6 | First adapter logical address |
| N+31 | 4 | First adapter IP address |
| N+35 | 6 | Second adapter physical address (MAC) |
| N+41 | 6 | Second adapter logical address |
| N+47 | 4 | Second adapter IP address |

## Start posting events (SP function)

The **SP** function operates in the LANDP for DOS, LANDP for OS/2, or LANDP for Windows environments.

An OS/2 PM, Windows 3.1/3.11, or Windows NT graphical user interface client can use the **SP** function to instruct LANDP to begin posting the events specified by the *event IDs*, to the message queue for the application program. The events are posted with the message identity associated in this function. For background information on the use of events, see the “LANDP event notification support” section in the “Writing client programs” chapter of the *LANDP Programming Guide*.

The event ID is appended to an internal event list that specifies the list of events to be posted. One internal event list is maintained for each process/window. The **SP** function can be requested according to application program requirements for building the event list.

The event list used with the **SP** function cannot be empty. **SP** does not accept mouse or keyboard events.

| CPRB Field | Content/Description |
|---|---|
| Function code | SP |
| Request DATA length | A multiple of 10 |
| Request PARMLIST length | 8 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 4 | Window handle that is to receive the OS/2 or Windows message |
| 4 | 4 | Message identity to associate with the OS/2 or Windows message |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | Multiple of 10 | Event IDs to be added to event list |

## Activate and deactivate timers (T0 through T8 functions)

These functions initialize the application program timers, T0 through T8. You can start them with values from 0 seconds to 24 hours. The value 0 deactivates a previously set timer. You can specify parameters that allow you to:

- Select when the timer should expire:
  - Current time plus the time specified in Request DATA
  - At the time specified in Request DATA
- Define the format and units for your time value.

When the specified interval expires, the timer T0 generates a key stroke and the timers T1 through T8 generate a time elapsed event.

| CPRB Field | Content/Description |
|---|---|
| Function code | T0 through T8 |
| Request DATA length | 4 or 6 |
| Request PARMLIST values length | 8 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | **'L'** Time specified in hours, minutes, and seconds<br>**' '** Time specified in minutes and seconds |
| 1 | 1 | **'T'** Clock time specified<br>**' '** Time specified to be added to current time |

| Request DATA values (minutes and seconds specified) | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | X'nnnn' Minutes (byte 0 most significant) |
| 2 | 2 | X'nnnn' Seconds (byte 2 most significant) |

| Request DATA values (hours, minutes, and seconds specified) | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | X'nnnn' Hours (byte 0 most significant) |
| 2 | 2 | X'nnnn' Minutes (byte 2 most significant) |
| 4 | 2 | X'nnnn' Seconds (byte 4 most significant) |

## Programming notes

If the application program initializes the T0 function just before it starts reading from the keyboard, and if the value elapses before the operator completes the entry, the system terminates the entry by inserting an ESC scan code into the input buffer. ESC is the default. You can change it during customization using the `TMR0ASC` keyword. See the *LANDP Installation and Customization* book for more information about using this keyword.

Otherwise, the operation completes normally. The data is read, and the application program should reset T0 to zero and continue processing. You can use this function to prevent your application program from locking when:

- The operator leaves during a keyboard field entry
- The operator leaves without finishing a transaction, thus preventing, for example, the shared files from being closed

## Stop posting events (TP function)

The **TP** function operates in the LANDP for DOS, LANDP for OS/2, or LANDP for Windows environments.

An OS/2 PM, Windows 3.1/3.11, or Windows NT graphical user interface client, can use the **TP** function to instruct LANDP to stop posting the events specified by the *event ID*s, to the message queue for the application program. For background information on the use of events, see the "LANDP event notification support" section in the "Writing client programs" chapter of the *LANDP Programming Guide*.

The **TP** function has the reverse functions of the **SP** function (see "Start posting events (SP function)" on page 20). The **TP** function deletes the event ID from the current event list that is to be posted.

By specifying an empty event list (Request DATA length = 0), the application program can stop all events from being posted to its message queue.

| CPRB Field | Content/Description |
|---|---|
| Function code | TP |
| Request DATA length | 0 or a multiple of 10 |
| Request PARMLIST length | 8 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 4 | Window handle that was associated with the OS/2 or Windows message |
| 4 | 4 | OS/2 or Windows message identity |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 or a multiple of 10 | Event IDs to be deleted (or all) from event list |

## Wait for asynchronous events (WM function)

The **WM** function allows the client to enter an idle state. It then waits until one or more specified asynchronous events occur. For background information on the use of events, see the “LANDP event notification support” section in the “Writing client programs” chapter of the *LANDP Programming Guide*.

**Windows 3.1/3.11 users**

Before using the **WM** function, read the “LANDP for DOS and Windows 3.1/3.11” section in the “Writing client programs” chapter of the *LANDP Programming Guide*.

These events can be from the keyboard, the host computer, the I/O devices, any server, any set timer, or from the timeout of the **WM** function itself. Also, LANDP for OS/2 and LANDP for Windows provide events generated by the mouse and semaphores. LANDP for Linux provides events generated by semaphores, but not by the mouse. The client/server mechanism checks the asynchronous events for which the application is waiting, in the order specified in Request DATA. If the application program does not specify asynchronous events, **WM** terminates when an event occurs.

While this function is running, the application program remains idle.

| CPRB Field | Content/Description |
|---|---|
| Function code | WM |
| Request DATA length | 0 to 1020 |
| Request PARMLIST length | 1 or 7 |
| Reply DATA length | 0 (see note) |
| Reply PARMLIST length | 10 |
| Replied DATA length | 0 (see note) |
| Replied PARMLIST length | 10 |
| **Note:** When used with the ZN supervisor local function, the Reply DATA length must allow for the maximum length of the data to be received. The Replied DATA length is the actual length of that data. | |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Timeout flag. If this flag is set to ’T’ (X'54') the following bytes contain the timeout value for this function (see Notes below) |
| 1 | 2 | X'nnnn'. Timeout value in hours |
| 3 | 2 | X'nnnn'. Timeout value in minutes |
| 5 | 2 | X'nnnn'. Timeout value in seconds |

## Notes on the use of a WM function timeout:

- This parameter is used to specify a timeout for the **WM** function. This timeout is deactivated automatically when a specified asynchronous event occurs and control is returned to the application program.
- If the parameter 'T' is not used, the application program remains idle indefinitely, waiting for any of the asynchronous events set by this function call.
- The maximum timeout value is 24 hours. If 'T' is specified but the timeout value is 0, the list of asynchronous events is checked once and control is returned to the application program immediately. If there was any event pending, it is dispatched to the application.
- The LAN timeout value specified during customization and the value specified in the *ehctimeout* (timeout per request) CPRB field do not apply to the **WM** function.

| **Request DATA Values** | | | |
|---|---|---|---|
| **Offset** | **Length** | **Content** | |
| 0 | 2 | **X'nnnn'** | Asynchronous event code to wait for |
| 2 | 8 | **SERVNAME** | Name of the server that originates the asynchronous event |
| 10 | 2 | **X'nnnn'** | Asynchronous event code to wait for |
| 12 | 8 | **SERVNAME** | Name of the server that originates the asynchronous event |
| **Notes:** Providing Request DATA is optional. The pattern above shows two entries; if you need to define multiple entries, concatenate them without any delimiter; you can specify up to a maximum of 102 entries.<br><br>For LANDP for OS/2 or LANDP for Windows, an *alias name* can be specified in place of SERVNAME. | | | |

The character # has a special meaning when used in a server name. It behaves as a place-holder for any valid character. For example, if a server with the name PRINTER# is loaded and PRINTER# is specified as server name in Request DATA, clients receive events from servers called PRINTER1, PRINTER2, PRINTERA, and so on (as the names have been defined during customization).

The possible asynchronous events are:

| **Server Events** | | |
|---|---|---|
| **Server name** | **Event Code** | **Event/Explanation** |
| SNAnn<br><br>or<br><br>X25NATnn<br><br>or<br><br>PPCnnnn | CP | Shows waiting for a host message from the LU with the nn session ID. If nn is ##, the host message can be from any of the LUs the application program can use. The application program gets the message transferred to its own Reply DATA buffer using the communication server read function. The indication is not cleared until the read function is issued and there are no pending messages to be read. |
| | ## | Shows waiting for any event from the server. |
| Any other | xx | Shows waiting for the event identification code xx from the server SERVNAME, which can be a LANDP server or a user server. When this entry is received, the indication is removed. |
| | ## | Shows waiting for any event from the server. When this entry is received, the indication is removed. |

| **System Events** | | |
|---|---|---|
| **Server name** | **Event Code** | **Event/Explanation** |
| SPV | Tm | Shows waiting for the expiration of the specified started user timer, where m is a value between 1 and 8. If the user timer T0 has been started, its expiration is regarded as a 'KB' entry for the WM function. |
| | KB | Shows waiting for any pressed keyboard key. |
| | TE | Shows the expiration of the timeout specified in the WM function call. |
| | MO | LANDP for OS/2 or LANDP for Windows: Shows waiting for any mouse event. |
| | Sn | LANDP for OS/2, LANDP for Windows, or LANDP for Linux: Shows waiting for any semaphore event where n is a value returned by the AS function. |
| Any Server | | Shows waiting for process disconnection from the LANDP or user server. When this entry is received, the indication is removed. |
| | && | Shows waiting for process connection from the LANDP or user server. When this entry is received, the indication is removed. |

It is recommended that the Request DATA area is used, although its use is optional. If the Request DATA length is set to X'0000', no input list is specified for **WM**. The following assumptions are therefore made when an asynchronous event occurs.

The application program receives any server event and the following system events: T0 through T8, KB, TE, &&, and **.

## Windows applications

Asynchronous LANDP events are notified to your application through the standard Windows protocol. Windows messages are queued in the application queue.

You must not use the Wait for Asynchronous Events (WM) function call because it might conflict with the Windows dispatching mechanism, unless the timeout interval is zero.

To deal with asynchronous events, your Windows applications must use the following supervisor function calls:

- SP (Start posting events)
- QE (Query event)
- TP (Stop posting events)

Your Windows application must deal with system events (keyboard, mouse, timers, and so on) through the standard Windows protocol. These events are not available to LANDP.

The application program receives any server event and the following system events: T0 through T8, KB, and TE. For LANDP for OS/2 and Windows applications that use alias names, it is strongly recommended that **WM** and **AA** functions are used to define the list of events that are waited for. Where an application does not specify asynchronous events for which it is waiting in Request DATA, LANDP for OS/2 and Windows differentiate between the following processing options:




- If the workstation where the application is running has only a single client defined (without an alias) for a remote resource, the asynchronous event is notified to the application with the SERVNAME field set to the server name.
- If a single alias is defined for the client in the case above, the SERVNAME field is set to the alias name.
- In any other case (where either more than one client alias is defined on a workstation for a single remote resource, or both client and server are on the same workstation), the SERVNAME field is set to “????????”.

| Reply PARMLIST Values | | | |
|---|---|---|---|
| **Offset** | **Length** | **Content** | |
| 0 | 2 | **X'nnnn'** | Asynchronous event code that caused the indication |
| 2 | 8 | **SERVNAME** | Name of the server that originated the asynchronous event (for LANDP for OS/2 and Windows, an *alias name* can be specified in place of SERVNAME) |

**Note:** On Intel machines, the event code is byte reversed (that is, ’TE’ appears as ’ET’).

Some possible values returned in Reply PARMLIST are:

1. Host messages received from the LANDP communication servers.

The identification code is 'CP' and the server name can be 'SNAnn', 'X25NATnn', or 'PPCnnnn'. ('nn...' is the session ID or the conversation ID.)

2. Timer expiration.

   The event identification code is 'T0' through 'T8' and the server name is 'SPV'. If the expired timer is the timeout specified in the WM function call, the event identification code is 'TE'.

3. Keyboard key pressed by the operator.

   The event is specified with the code 'KB' and the server name is 'SPV'.

4. Asynchronous events received from other LANDP or user servers.

   The event identification code is specified.

5. Process connection or disconnection received from any LANDP and user servers.

   '&&' is the event identification code.

6. Mouse movement or mouse click.

   'MO' is the event code and the server name is 'SPV'.

7. Semaphore cleared.

   'Sn' is the event code and the server name is 'SPV'.

## Reply DATA

When this function is used to receive notifications sent by the ZN supervisor local function, the Reply DATA area contains the data sent. Otherwise, this area is not used.

## Preventing locking

To prevent your application program from being locked, it is recommended that you use the timeout parameter 'T' in the WM function call. If you want to wait for any event, set Request DATA length to X'0000' and specify a timeout value in the Request PARMLIST to delimit the time waiting for any event.

## Asynchronous request—asking for TT request (Z4 function)

The **Z4** function (which is used only from a server program), starts, cancels, or modifies the frequency of the periodic timer requests (TT) issued by the client/server mechanism. Asynchronous requests are sent with the `RMTAREQ` routine.

With the value in the first two bytes of Request PARMLIST, you specify the effect of the asynchronous request **Z4**.

| CPRB Field | Content/Description |
|---|---|
| Function code | Z4 |
| Request DATA length | 0 |
| Request PARMLIST length | 2 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |
| Server name | SPV |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | X'nnnn'<br><br>Interval time as multiples of 50 milliseconds.<br><br>A zero value cancels the periodic TT requests |

When replying to the TT request, the server can update the interval time for receiving TT requests by modifying the content of the first word of the Reply PARMLIST.

To restart the client/server mechanism for sending periodic TT requests, the server must again issue a Z4 request.

## Asynchronous event notification or cancellation (Z5 function)

The **Z5** function (which is used only from a server program) sends, or cancels, an asynchronous event notification in the client/server mechanism's event queue. In the Request PARMLIST you specify whether an asynchronous event is to be sent or cancelled.

You must specify the destination Process ID and destination Workstation ID fields of the incoming CPRB, in the destination CPRB fields for this function, so that the requesting client can receive the event.

| CPRB Field | Content/Description |
|---|---|
| Function code | Z5 |
| Request DATA length | 0 |
| Request PARMLIST length | 3 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |
| Resource origin | C'cccccccc'<br><br>Server name left-justified and padded with blanks; can be any valid server ID for this server |
| Destination workstation ID | C'cc' |
| Destination process ID | X'xxxx' |
| Server name | SPV |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | X'nnnn'<br><br>2-byte value showing the reason for the asynchronous request |
| 2 | 1 | X'nn' (reason code)<br><br>X'30' resets a previous notification with the same reason code as this request<br><br>Any other value sets it |
| **Note:** The reason code is a value that is agreed by convention between the server and the application. | | |

## Extended asynchronous event notification (ZN function)

The **ZN** function (which is used only from a server program and only with LANDP for OS/2, LANDP for Windows, or LANDP for Linux) sends or cancels an asynchronous event notification in the client/server mechanism's event queue. It extends the function provided by **Z5** (see page 30) by allowing:

- "State changed" events—events which are automatically set off once the event notification has been delivered to the destination process
- Data to be sent with the event notification

You must specify the destination Process ID and destination Workstation ID fields of the incoming CPRB, in the destination CPRB fields for this function, so that the requesting client can receive the event.

You must also set appropriate values for the *ehc_flags* and *ehc_ZNid* fields of the CPRB. In *ehc_flags* you specify whether the asynchronous notification is to be sent or cancelled and whether the notification is an event or a state changed notification. In *ehc_ZNid* you give the reason code for the event.

| CPRB Field | Content/Description |
|---|---|
| *ehc_flags* | B'....1...': bits 1 and 4 have significant values<br><br>B'...X....': bit 4 value<br><br>B'.....X.': bit 1 value<br><br>See below for the meanings of bits 1 and 4. Other bits are reserved or used for other purposes. |
| Function code | ZN |
| Request DATA length | Length of data to be sent |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |
| *ehc_ZNid* | Reason code |
| Resource origin | C'cccccccc'<br><br>Server name left-justified and padded with blanks; can be any valid server ID for this server |
| Destination workstation ID | C'cc' |
| Destination process ID | X'xxxx' |
| Server name | SPV |

The *ehc_flags* byte is treated as follows:

**B'...11.1.'**
Cancel asynchronous notification (equivalent to Z5 with offset 2 in the Request PARMLIST set to X'30').

**B'...01.1.'**
Send asynchronous notification (equivalent to Z5 with offset 2 in the Request PARMLIST set to a value other than X'30').

**B'...11.0.'**
Cancel state change notification.

**B'...01.0.'**
Send state change notification.

## Request DATA

The data to be presented with the event notification.

## Event notification with RMTRPLY

By setting the *ehc_flags* and *ehc_ZNid* fields in the CPRB, you can send or cancel asynchronous event notifications in a reply, using `RMTRPLY`. However, you cannot send event data with the reply.

# Part 2. Wide area communication servers

The LANDP family provides servers to enable workstations to communicate with host computers, other LANDP workgroups, or communication controllers through a supported link.

This part of the book contains the following chapters:

**Chapter 2, "SNA communication server" on page 35**
This chapter describes the functions provided by the SNA server for servicing requests originating in application programs or other servers. It also provides some programming notes that you should consider when designing your application programs.

**Chapter 3, "The cryptographic interface" on page 75**
This chapter describes the cryptographic application programming interface supported by LANDP for DOS. It also provides the programming information required to write programs that use this cryptographic application programming interface.

**Chapter 4, "Native X.25 communication server" on page 91**
The native X.25 server provides application-to-application, high-level, protocol-free communication using X.25 data link control. This server can coexist with the SNA server in a personal computer system.

This chapter describes the server characteristics and the functions provided for servicing requests originating in application programs or other servers.

**Chapter 5, "Program-to-program communication server" on page 105**
The PPC communication server allows application programs to communicate with a partner application program using the LU 6.2 node.

This chapter describes the functions provided by the server for servicing requests originating in application programs or other servers. It also provides some programming notes that you should consider when designing your application programs.

**Chapter 6, "MQSeries Link server" on page 119**
The MQSeries Link server allows workstations in a LANDP workgroup to add messages to, and retrieve messages from, an MQSeries Queue Manager being run by MQSeries for OS/2 or MQSeries for Windows.

© Copyright IBM Corp. 1992, 2002

This chapter describes the functions provided by the server for servicing requests originating in application programs or other servers. It also provides some programming notes that you should consider when designing your application programs.

**Chapter 7, “TCP/IP Wide Area Communications server” on page 137**

The TCP/IP Wide Area communications server allows programs to communicate with host computers and other workstations over a TCP/IP network.

This chapter provides some programming notes that you should consider when designing your application programs.

# Chapter 2. SNA communication server

The Systems Network Architecture communication server (SNA server) allows an application program running under DOS, OS/2, or Windows to communicate with a host computer application program. The workstation in the LANDP workgroup with the SNA server installed must have a connection to the host computer. This workstation is called the *gateway*.

Additionally, the LANDP TCP/IP wide area communications server emulates the SNA server and allows LANDP application programs to use the same set of functions for SNA over TCP/IP, or TELNET tn3270 communications. It is available in LANDP for DOS, LANDP for OS/2, LANDP for Windows, or LANDP for Linux environments.

If the gateway is a DOS workstation, synchronous data link control (SDLC), token-ring, and X.25, protocols are all supported. If PC/Integrator is installed, the device cluster attachment (DCA) data link control is also supported.

If the gateway is an OS/2 workstation, the communication functions are based on IBM Communications Server for OS/2 Warp and the Conventional LU Application (LU  A) Custom Feature. It is essential that Communications Server for OS/2 Warp is installed in the gateway. The protocols supported by LANDP for OS/2 are the same as those supported by the LU  A.

If the gateway is a Windows workstation, the communication functions are based on the following SNA communications providers:

- IBM Communications Server for Windows NT and Windows 2000, Version 6.1.1
- IBM Personal Communications, Version 5.0.1, for Windows, and IBM Personal Communications, Version 4.3, for OS/2
- Microsoft SNA Server, Version 4.0
- Microsoft Host Integration Server 2000

and on the Conventional LU Application (LU  A) Custom Feature. It is essential that the SNA communications provider is installed in the gateway. The protocols supported by LANDP for Windows are the same as those supported by the LU  A.

LU pooling support is provided by the LANDP SNA communication server on all supported operating system platforms. For further information about LU pooling support, see the *LANDP Installation and Customization* book.

Except where stated, all functions operate in the LANDP for DOS, LANDP for OS/2, LANDP for Windows, or LANDP for Linux environments.

© Copyright IBM Corp. 1992, 2002

*Table 4. Function Codes used in the SNA Communication Server*

| Function code | Description | Env. | Page |
|---|---|---|---|
| CL | Close | 02NL | 53 |
| CN | Connect | 02NL | 54 |
| DC | Define connection | 02-L | 55 |
| GS | Get status | 02NL | 58 |
| OP | Open | 02NL | 58 |
| QC | Query connection | 02-L | 60 |
| RH | Read host | 02NL | 60 |
| RL | Release | 02NL | 64 |
| SH | Send host | 02NL | 64 |
| ZI | Server information | 02NL | 66 |

The first and second columns give the function code and the name of the function. The third column shows the operating environment of the function, as explained in "Operating environments" on page xxvi. "02NL" means that it is available from LANDP for DOS, LANDP for OS/2, LANDP for Windows, or LANDP for Linux servers. The last column refers to the page where you can find the function described.

**Note:** In LANDP for Windows, Data connection (DC) and Query connection (QC) functions are provided by the Microsoft SNA Server.

Data compression is supported in LANDP for DOS, which supports RLE compression by calling the EHCCOMP server. The interface of this compression server has two functions:

*Table 5. Function Codes used in the Compression Server*

| Function code | Description | Env. | Page |
|---|---|---|---|
| CM | Compress data | 0--- | 71 |
| UC | Decompress data | 0--- | 71 |

## SNA server characteristics

This section presents those server characteristics that can be useful when writing an application program. The protocols supported, and also the parameters influencing programming, are shown.

The SNA server supports the following SNA features:

- Physical unit (PU), type 2.0.
- Logical unit (LU), type 0. Transmission services (TS) and function management (FM), profiles 3 and 4.
- Logical unit (LU), type 1. TS and FM, profile 3.
- Logical unit (LU), type 2. TS and FM, profile 3.
- Format identifier (FID) 2.

The SNA server-supported protocols have the following characteristics and restrictions:

- Each SNA server supports as many physical units as the communications subsystem can handle.
- Each LANDP for DOS SNA server represents one physical unit.

  When used with X.25 data link control, the LANDP for DOS SNA server can represent more than one PU. Then, each PU is associated with an X.25 virtual circuit.
- Each SNA server supports as many LUs as the communications subsystem can handle. In LANDP for DOS, the maximum number of LUs supported per server is 240. The different types of logical units should be used for different purposes, as follows:

  **LU 0** For communication from an application program in the computer in the LANDP workgroup to a CICS® or IMS™ application program in a host system.

  **LU 1** For printing an SNA Character Stream (SCS). The 3287 printer emulator receives the SCS and prints it on an IBM 4201 Proprinter® or a functionally equivalent device.

  **LU 2** For 3270 emulation.
- Sessions can be opened by the primary or the secondary LU.
- Sessions can be closed by the primary or the secondary LU.
- The host system communication parameters must be consistent with these rules and the BIND parameters, which are described in the following section.
- **LANDP for DOS:**
  - The maximum segment size used with SDLC is 256 bytes, with token-ring it is 512 bytes, and with X.25 it is 1024 bytes.
  - The data link control dynamically reserves message storage in a buffer pool. The number of buffers in the pool is defined during customization.

## BIND parameters

The SNA server checks the BIND parameters. It sends a negative response on reception of invalid BIND parameters. BIND parameters must adhere to the following:

- LU types 0, 1, or 2.
- SNA compression is supported by LANDP for DOS. For LANDP for OS/2 and LANDP for Windows, compression can be supported by the SNA communications provider.
- Normal flow in send and receive modes can be:
  - Half duplex flip flop
  - Half duplex contention
- The maximum size of the outbound request unit (RU) is 4096 bytes.
- The first speaker must be the secondary station.
- Format headers are allowed in both directions.
- Delayed request control mode is not supported.
- Alternate code set cannot be used.
- Outbound pacing is zero to 15.
- Inbound pacing is supported.
- Multiple RU chains are supported in both directions.
- Chain response protocols supported in both directions are:
  - Definite
  - Exception
  - Either definite or exception
  - No response
- Bracket protocol *must* be used for LU 1.
- The host computer application program name is not checked by the SNA server. However, the complete BIND message is passed to the application program, which can decide to end a session if it disagrees with the host computer application program name or with any other BIND parameter.

## BIND protocols

The BIND specifies the communication protocols used. The protocols supported by LANDP are presented below.

### Segmentation protocol

The SNA server delivers complete RUs, even though they were received in segments.

### Bracket protocol

The SNA server manages the bracket protocol that informs the application program of the incoming bracket status. It also handles the change of direction and the bracket indicators.

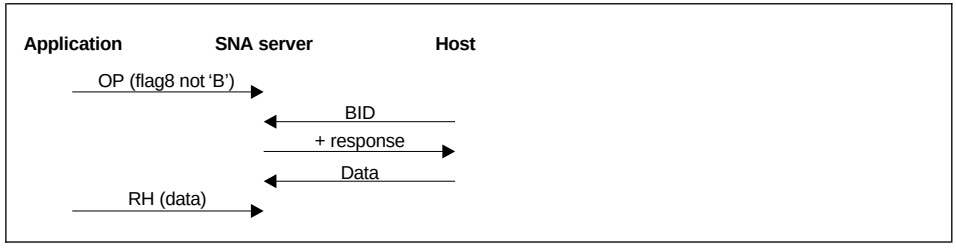
The server is in the *between brackets* state after it has received an *end-bracket* and before it receives a new *begin-bracket*. When it is in the *between brackets* state, the first message of a chain is sent with a *begin-bracket* indicator.

If the BIND session specifies that the application program can send *end-bracket*, the first message of a chain is sent without *end-bracket*, by default. To change it, SH function flag3 must be set to '**]**'.

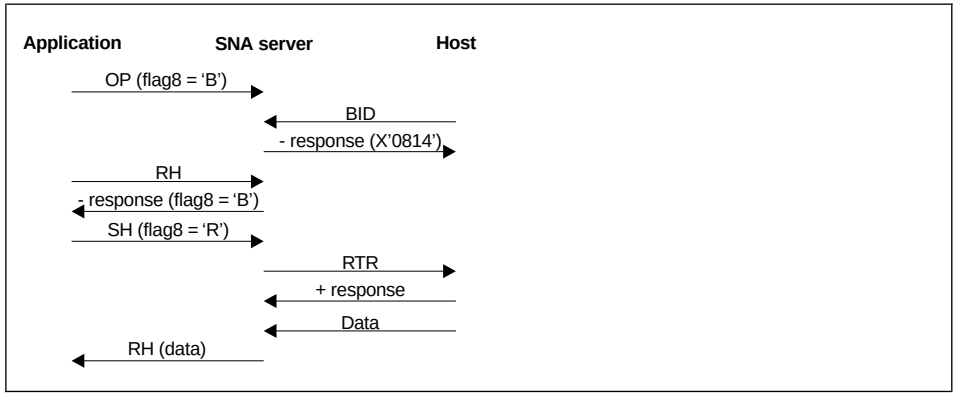
## BID protocol

At session level, the BID ("bidder's request to initiate a bracket") protocol is managed either by the SNA server or by the application program. The default is set at customization.

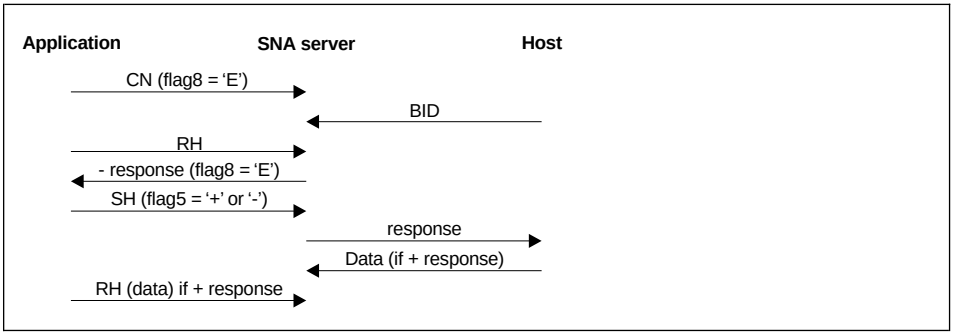
If the SNA server controls the BID protocol, it can generate a positive or negative response. If the LU session is in the *between brackets* state, the response is positive. In this state the BID is accepted. In other cases it generates a negative response (sense code = X'0814'). A negative response means a delay in the acceptance of the BID. When the session changes to the *between brackets* state, the server sends a ready-to-receive (RTR) command to accept data from the host. Normal operation is shown diagrammatically as follows:




If the application program wants to control the BID protocol, flag8 must be set to 'B' in the OP function request. Then the server always generates a negative response and a null message is queued for the application program with flag8 = 'B'. When the application program receives the pending host system unsolicited message, it must notify the server, using the SH function with flag8 = 'R'. Then, the server sends an RTR command to the host system, enabling it to start sending the unsolicited messages. Normal operation is:



In a third case, the BID is managed by the application program, issuing a response to the BID command. When the BID arrives, the server does not respond to it, as that is the responsibility of the application. The SNA server queues a flag8 = 'E' to the application, which must then issue a positive or negative response, with the appropriate sense code.




When the BID has been accepted, the server is in the *pending begin bracket* state and refuses any SH function request from the application program.

## RTR protocol

An RTR command is sent to the host system when an SH is requested with flag8 = 'R'. It can be sent either as a response to a failed BID, or to find out if the host system wants to send a message to the application program.

## Chaining protocol

The SNA server allows the transmission of chains that contain several messages. The application program can receive a message as a part of a chain. It can also build its requests in chains.

Also, the chaining protocol is considered when a transaction involves units of information larger than the maximum length of the request unit parameter in the BIND (4096 bytes). It allows those units of information to be sent as separate parts, but treated as one message at the other end.

The chain element associated with each message in the chain is shown by flag2 of the SH function request. The corresponding values are:

**Flag2 = '('** First in chain (FIC).

**Flag2 = '+'** Middle in chain (MIC).

**Flag2 = ')'** Last in chain (LIC).

**Flag2 = ' '** Only in chain (OIC). This is the default flag setting.

A chain can be canceled by the SH function if flag2 is set to 'C' (Cancel).

The SNA server builds SNA headers from the information in flag2 and sends a change direction indicator (CDI) along with LIC or OIC.

## Change direction protocol

The host computer application program uses a CDI, according to the SNA half duplex flip flop protocol, when sending a message. Then the application program in the computer in the LANDP workgroup receives the message with flag4 = 'X' as the response to an RH function request. Finally, the SNA server's internal protocol state is changed to enable the application program in the computer in the LANDP workgroup to use the SH function. In turn, when the application program in the computer in the LANDP workgroup sends in the last message of a chain, the server sets the CDI.

## Data security protocol

The SNA server supports session-level encryption. Parameters in the BIND show whether cryptography is used.

## Function management headers protocol

Parameters in the BIND show whether function management headers (FMH) can be used.

## Response protocol

The response protocol is managed by the SNA server, based on the session BIND. The following possibilities are available:

- Bits 5 and 4 in byte 5 of the BIND are set to 00: the LU half-session sends no response.
- Bits 5 and 4 in byte 5 of the BIND are set to 01: the LU half-session sends a response only when the response is negative (exception response).
- Bits 5 and 4 in byte 5 of the BIND are set to 10: the LU half-session always sends a response (definite response).
- Bits 5 and 4 in byte 5 of the BIND are set to 11: the LU half-session response depends on the SH function flag9 setting. When set to 'D', it sends a definite response. Otherwise, it sends an exception response.

The corresponding data and command values are displayed below:

| Bits 5 and 4 in byte 5 of BIND | Byte 1 of the request header sent by the SNA server | |
|---|---|---|
| | Data | Commands |
| 00 | X'0' | X'0' |
| 01 | X'9' | X'9' |
| 10 | X'8' | X'8' |
| 11 | Flag9 = 'D' – X'8'<br>Flag9 = ' ' – X'9' | X'8' |

When the application program requests control of the SNA responses to the requests received, by setting flag5 = 'R' in the RH function, it gives either a positive or a negative response, with the appropriate sense code.

The application program must send the response using the SH function with the following values:

**Flag5 = '+'** Involves Request DATA length zero.

**Flag5 = '–'** Involves Request DATA length either 2 (user sense code) or 4 (SNA sense code plus user sense code).

**Request DATA** Contains sense code for negative responses.

The application program does not need to send a response when the RH function returns the following flag settings:

- Flag1 not equal to ' '
- Flag5 = '+' or '–'
- Flag8 = 'K' or 'C'

### Quiesce protocol

An LU might enter a *quiesce* state. Before releasing the *quiesce* state, when either SH or CL is attempted, the application program receives a return code X'01005042'. When the release of the *quiesce* state is received, a message with flag1 = '<' and data size zero is queued for the application program. At that point, the application program regains control.

## SNA connection process

Before it can communicate with a host application program, the LANDP application program must establish an SNA session. The sequence of events is as follows:

1. Application programs must issue the CN function to start an SNA session. They must issue this function before any other SNA communication server function to ensure exclusive use of a logical unit (LU).

   After requesting an RH function, the application program receives one of the following values for flag1:
   - Flag1 = 'L', when the LU is connected successfully
   - Flag1 = '}', if the connection process has failed
   - Flag1 = '{', if an X.25 call packet has been accepted by the network

   For optimal performance, use the following sequence of functions:

   ```
   1. CN                                  SNA server connect function
   2. WM                                  Wait Multiple function for SNA events
   3. RH returning flag1='L','}', or '{' SNA server read host function
   ```

   **Note:** Avoid repetitive useless RH functions.

2. LU–LU session establishment can be initiated either by
   - The LANDP application program
   - Both the application program in the host computer and the LANDP application

   The PAR&SNA customization parameter defines session establishment for LU 0 sessions.

   LU–LU sessions for other LUs can always be initiated by both.

- **Session initiation by the LANDP application program only:**

  A BIND is not accepted until the application program has requested an OP function. The sense code for rejected BINDs is X'0831'. When the application program requests an OP, an SNA INITSELF command is sent to host computer.

  **Session initiation by both:**

  If BIND, STSN, and SDT are received before the LANDP application program has requested the OP function, the application program receives the flag1 = '<' using the RH function. Here, the LU–LU session is established without the need of the OP function. When the application program requests an OP function, an SNA INITSELF command is sent, unless the LU–LU session is already active.

The OP function is not necessary but it is recommended, even if the session has already been established. It can be useful to confirm or change the management of the BID protocol specified during customization, using flag8.

After requesting an RH function, the application program receives the return code flag1 = '<', when the previous INITSELF (if sent) was accepted by the host computer, and BIND, STSN and SDT commands have been received from the host computer and accepted (LU–LU session established).

If INITSELF was rejected by the host computer, the RH function returns the negative response received from host computer with the sense code in the Reply DATA area. To establish or reestablish the session, the LANDP application program must use the OP function with the required data.

Again, for optimal performance, use the following sequence of functions:

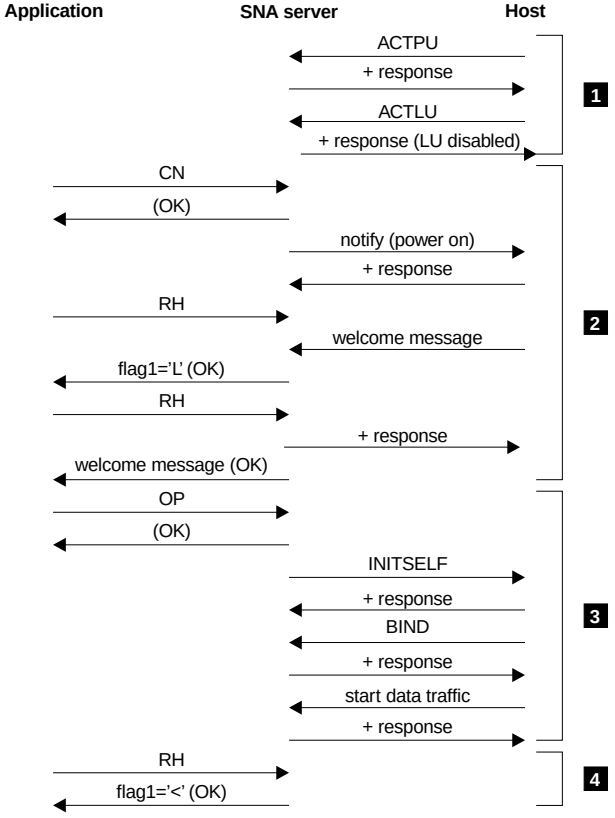
```
1. OP                                  SNA server open function
2. WM                                  Local Wait Multiple function
3. RH returning flag1='<' and data,
             or flag5='-' and data     SNA server read host function
```

**Note:** Avoid repetitive useless RH functions.

The following diagrams show how the session can be established.

This figure shows the start of a secondary LU–LU session.

The link and PU have already been activated before the application issued the CN request. There is also a “welcome” message (an SSCP–LU message), for example, when the VTAM® USSTAB parameter is used.




**Notes:**

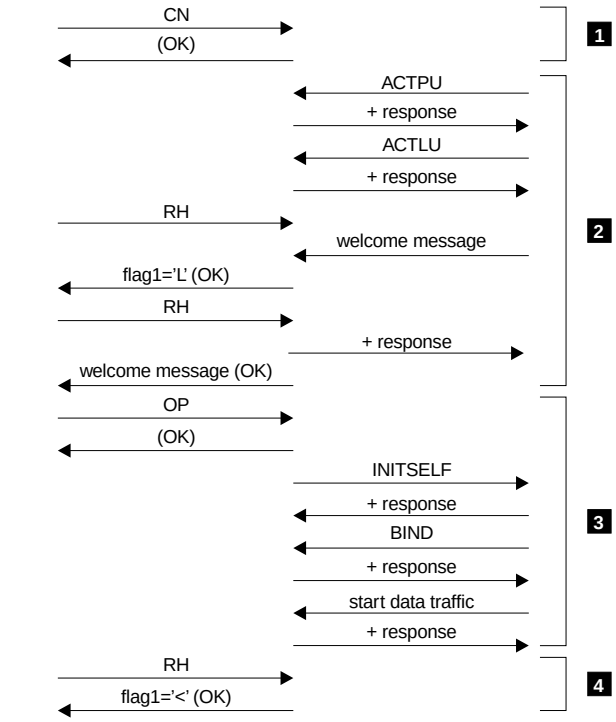
a. **1** Link and PU activation. SSCP–PU session established but SSCP–LU session not, because secondary LU is not ready.

b. **2** CN function provokes a “notify (power on)” that tells VTAM that the secondary LU is ready. SSCP–LU is established and the “welcome message” is received.

c. **3** Start of LU–LU session establishment.

d. **4** Confirmation of LU–LU session establishment is read by the application.

This figure shows the start of a secondary LU–LU session, where the PU activation is performed after the application has issued the CN request. Again, it shows an SSCP–LU message.




**Notes:**

a. 1 Connection initiated by the application.

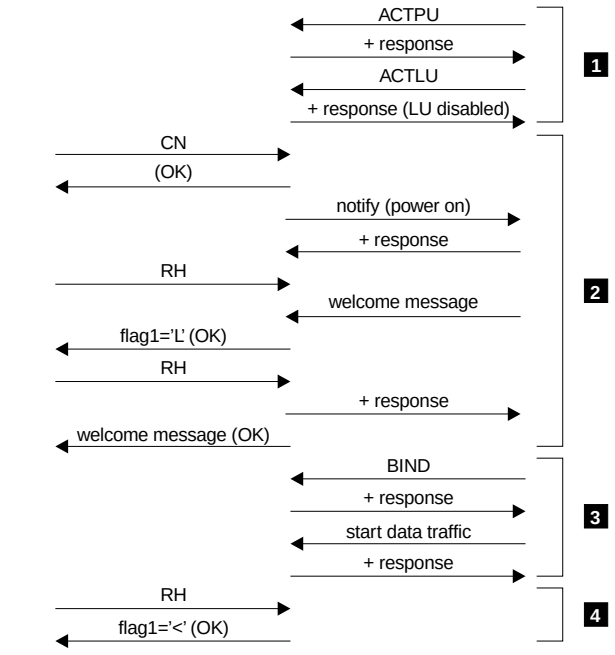
b. 2 PU and LU activation from the host node. SSCP–PU and SSCP–LU sessions are established (the secondary LU was ready) and the "welcome message" is received.

c. 3 Start of LU–LU session establishment.

d. 4 Confirmation of LU–LU session establishment is read by the application.

This figure shows the start of a primary LU–LU session, where the PU activation has already been performed by the application issuing a CN request. Again, it shows a “welcome” message.

In this case, the BIND is automatically sent by the primary node immediately after the SSCP–LU session is established. This process occurs when a LOGONAPPL statement is specified in a VTAM secondary LU definition.




**Notes:**

a. **1** Link and PU activation. SSCP–PU session established but SSCP–LU session not, because secondary LU is not ready.

b. **2** CN function provokes a “notify (power on)” that tells VTAM that the secondary LU is ready. SSCP–LU is established and the “welcome message” is received.

c. **3** BIND request is received from the host.

d. **4** Confirmation of LU–LU session establishment is read by the application.

3. An OP function can be coded in two ways:
   - Providing in the Request DATA area the name of the host application program to establish the session with, and in Request DATA length the length of this name (4 to 8 EBCDIC-coded characters).

     SNA server builds and sends the INITSELF command using this name.
   - Providing the complete INITSELF command in the Request DATA area and setting Request DATA length to the corresponding value. For INITSELF format 0 (X'010681'), the host application name is placed at byte 14 and its length at byte 13. For INITSELF format 1 (X'810681'), the host application name is placed at byte 18 and its length at byte 17.
4. Use of cryptography (not supported by LANDP for Windows or LANDP for Linux):
   - If the SNA server accepts the BIND, the cryptography options are tested. If no session-level cryptography is requested, a positive response is sent and the host sends an SDT command (start data traffic). This establishes the LU–LU session.
   - There are two types of cryptography management:
     – That provided by the LANDP for OS/2 SNA server. When session-level encryption is requested, the LANDP for OS/2 SNA server issues the required calls through Communications Server for OS/2 Warp to the cryptographic interface to decrypt and encrypt session data. In that way, existing SNA server applications can take advantage of cryptographic services without code modification. Also, this feature allows the concurrent use of both SNA session-level encryption and SNA compression.
     – That incorporated in a client application program.

     In the latter case, if the BIND shows that either selective or mandatory session-level cryptography is to be used, a message is queued for the application program containing:
     – Flag10 set to the appropriate value ('S' for selective, 'M' for mandatory)
     – Session cryptography key encrypted under the secondary LU master key.

     After reading this information, the LANDP application program must respond with a random number encrypted with the session key. Then the SNA server responds to the BIND. When the host sends a CRV command, a message with flag10 = 'V' is queued for the application program. The application program must respond with a SH function, a positive or negative response indicating if cryptography verification was successful or not.

   For further information, see "Cryptography support in the SNA server" on page 48, Chapter 3, "The cryptographic interface" on page 75, and the *LANDP Introduction and Planning* section on the cryptographic interface.
5. Verifying the host application name:

   When flag1 = '<' is passed to the application program, both STSN and BIND are returned in the Reply DATA area. The BIND parameter contains the application program name of the host computer that requested the session initiation. The LANDP application program can verify that this name is the one expected by inspecting the BIND bytes 32–40. This protects against a session being established with an unwanted partner.

If the LANDP application program disagrees with this name or with any other BIND parameter, or wants to terminate the session for some other reason (also after data has been exchanged), it requests a CL function from the SNA server. The SNA server then sends a TERMSELF command to the host computer.

6. Before the application program ends, it must request a CL function to finish the session (if active) and an RL function to free the LU it used.

## SSCP–LU session

Besides LU–LU sessions, an application program can use the SSCP–LU session. The messages corresponding to the SSCP–LU session flow are read from the host system (RH function) and sent to the host system (SH function) with flag6 = 'S'.

## Cryptography support in the SNA server

Cryptographic support can be managed by the LANDP for OS/2 SNA server and the DLLs supplied with LANDP for OS/2 (`acsrencr.dll` and `acsrdecr.dll`). Alternatively, if you want your client program to manage cryptographic support, you can follow the procedure described below.

- When the SNA server receives a BIND with cryptography, it keeps the response and queues a message for the application program in the computer in the LANDP workgroup. When the application program reads the message, using the RH function, it gets:

```
Replied DATA length         = X'0008'
Flag10 in Reply PARMLIST    = 'S' or 'M' (Selective or Mandatory
                              cryptography)
Reply DATA                  = 8-byte session cryptography key
                              encrypted under secondary LU
                              master key.
```

- Then, the application program must obtain a random number encrypted with the session key and issue an SH function with:

```
Request DATA length         = X'0008'
Flag5  in Request PARMLIST  = '+'
Flag10 in Request PARMLIST  = 'S' or 'M' (Selective or Mandatory
                              cryptography)
Request DATA                = Random number encrypted with the
                              session key.
```

or, when there is an error:

```
Request DATA length         = X'0000'
Flag5  in Request PARMLIST  = '-'
Flag10 in Request PARMLIST  = 'S' or 'M' (Selective or Mandatory
                              cryptography)
```

Here, the SNA server sends a negative response to the BIND (sense code = X'0848').

- When the SNA server receives cryptography verification (CRV), it keeps the response and queues a message for the application program. When the application program reads the message, using the RH function, it gets:

```
Replied DATA length         = X'0008'
Flag10 in Reply PARMLIST    = 'V' (CRV)
Reply DATA                  = 8 bytes encrypted with the session key.
```

  The application program thus recovering the random number, inverts the first four bytes, and checks if it matches the random number that it originally sent to the primary LU through the BIND response.
- If it is correct, the application program must issue an SH function to the SNA server with:

```
Request DATA length         = X'0000'
Flag5  in Request PARMLIST  = '+'
Flag10 in Request PARMLIST  = 'V'
```

  The SNA server sends a positive response to CRV.

  If there are errors, the application program must reply with an SH to the SNA server with:

```
Request DATA length         = X'0000'
Flag5  in Request PARMLIST  = '-'
Flag10 in Request PARMLIST  = 'V'
```

  Then, the SNA server sends a negative response to the CRV (sense code = X'0848').
- When the cryptography session has been established, the LANDP application program must set flag10 = 'C' in each SH function to show that the RU is encrypted but not padded, or to 'P' to show it is encrypted and padded. Flag10 in the RH function is set to 'P' or 'C' by the SNA server according to the contents of the request header received from the host.

## SNA support for more than 30 user sessions per workstation

A modified SNA interface that allows for more than 30 user sessions per workstation is available when the SNA services are provided from an OS/2, Windows or Linux workstation. When using this interface, the session identifier can be any two ASCII characters. The interface differences are described in the following sections.

### Customization

To customize for the modified interface, use a server name of just SNA, instead of SNA followed by the session identifier, in the CLIENT keyword of the LWSCONF vector. Only one CLIENT=(SNA,xx) keyword is required, no matter how many sessions are to be used

If LU pooling support is required, provide an SES&SNA Server keyword for each session to be pooled. If CLIENT=(SNA,xx) has been specified, the SNA session identifier in the SES&SNA Server keyword can be any two ASCII characters.

### Installation requirements

The SNA session identifier part of the LUA profile name (defined under the appropriate SNA communications provider) can now be any two ASCII characters instead of just two decimal digits.

### Function requests

Requests must have a server name of SNA (instead of SNA followed by the session identifier) and the session identifier must be placed in bytes 11 and 12 of the Request PARMLIST (that is, the two bytes after Flag10).

### Events

SNA event identifiers for sessions connected using the modified interface have an event code of the session identifier (instead of CP) and a resource name of SNA (instead of SNA followed by the session identifier).

### Compatibility

If required both styles of SNA interface can be customized and used within the same workgroup. However, a consistent style must be used for each session. For example, if both CLIENT=(SNA,xx) and CLIENT=(SNA01,xx) appear in the LWSCONF for a workstation, SNA session 01 can be connected using a CN function of either style. However, until session 01 is released again, the same style as was used on the CN function must be used for all other functions on session 01.

## Special considerations for LANDP for DOS

There are some areas that merit special consideration when you use the LANDP for DOS SNA server: the connect (CN) and release (RL) functions (whose use is mandatory—see also the “Moving to the latest version of LANDP for DOS” section in the “Clients and servers” chapter of the *LANDP Programming Guide*.), the ACTPU command, and the ACTLU command.

### CN and RL functions

The CN function is specially relevant when you are using X.25 data link control (DLC). In this case a CN request from an application program results in an attempt to establish an X.25 virtual circuit. If the virtual circuit is already established, it has no effect.

CN and RL functions are also useful with switched synchronous data link control (SDLC) lines. They allow application programs to start and stop the Data Terminal Ready (DTR) circuit. The first CN request sets DTR on, and an RL request from the last active application program sets DTR off.

### ACTPU received

The ACTPU sent by the host to the SNA server carries an SSCP ID that is checked to decide if the command can be accepted or not, depending on customization values. If accepted, a positive response is sent which includes the PUNAME bytes set to ’FBSS’ in EBCDIC.

For SDLC and token-ring data link controls, a list of valid SSCP IDs can be defined during customization. If there is no matching SSCP ID, a negative response is sent. If this list is empty any ACTPU command is accepted.

For X.25 no SSCP ID verification is done, since it is included in the facilities field in the CN function request. SSCP ID might be blank.

### ACTLU received

The ACTLU received from the host for a specific LU gets a positive response if the LU is in the list of LUs defined during customization. Otherwise, the server sends a negative response with sense code X'0801'.

If there is a positive response, the response to the ACTLU carries a control vector indicating whether the LU is operative or not. If not operative, the SNA server notifies the host when this secondary LU becomes available.

When the application program receives flag1 = 'L' on return of RH function request, the SSCP ID is present in the Reply DATA area, if it was received from the host.

## Using the SNA server

This section provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields. If you need more information about the CPRB fields, see Appendix A, "Connectivity programming request block" on page 669.

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | 26 | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 26 | 2 | 26 | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | SNA## or SNA | Server name |
| **Note:** The value of ## is the session ID. | | | |

The following fields are variable and are discussed in each function request description:

- Function code
- Request DATA length
- Reply DATA length

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 44 | 2 | | Replied PARMLIST length |
| 46 | 2 | | Replied DATA length |

If the request was successful, the *router return code* and the *server return code* are both X'00000000'. In all other cases, see the appropriate section in the *LANDP Problem Determination* book to see if you should take any action. The return values in Reply PARMLIST and Reply DATA should be ignored if there is an error. The following fields are variable and are discussed in each function request description:

- Replied PARMLIST length
- Replied DATA length

**PARMLIST:** The Request PARMLIST and Reply PARMLIST fields for the SNA server are:

| Offset | Length | Content |
|---|---|---|
| 0 | 1 | Flag1 |
| 1 | 1 | Flag2 |
| 2 | 1 | Flag3 |
| 3 | 1 | Flag4 |
| 4 | 1 | Flag5 |
| 5 | 1 | Flag6 |
| 6 | 1 | Flag7 |
| 7 | 1 | Flag8 |
| 8 | 1 | Flag9 |
| 9 | 1 | Flag10 |
| 10 | 2 | Session ID if the Server name value is **SNA** |
| 17 | 9 | Transmission (TH) and Request/Response (RH) header |
| **Note:** Fields not used must be filled with blanks (X'20') | | |

**DATA:** Request DATA and Reply DATA contain data to be sent to or received from the server. The use of these fields is explained in the description of each function request.

**Note:** The application program must have enough space in Reply DATA to receive the record. Otherwise the message is truncated. It is recommended to allocate Reply DATA to the maximum size of 4096 bytes.

## Request reference

This section describes the functions that client programs can request from the SNA server. They are listed in alphabetical order of function code.

## Close (CL function)

This function terminates an SNA end-to-end session. The CL function request can be used at any time, even during the reception of a chain.

**Note:** Ending an application program does not imply a closing of communication sessions. You must issue a CL function to close sessions.

The SNA server sends a TERMSELF message to close the communication session with the host application program. The host answers this call with an UNBIND message. When the application program uses the CL function there might still be received messages in the internal buffers. These messages are passed to the application program the next time the RH function is used. The application program must disregard those messages that are not required. After receiving any pending messages, the loss of contact signal, (flag1 = '>', is passed to the LANDP application program after the UNBIND message is received).

The CL function does not deactivate an SNA SSCP–LU session. The application program can establish a new connection with the same or any other host application program if it uses the OP function again.

After a CL function, the BID protocol values are reset to the values defined during customization.

| CPRB Field | Content/Description |
|---|---|
| Function code | CL |
| Request DATA length | 0 to 14 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

### Request DATA values

You provide the host application program name either as a EBCDIC character field or by specifying the complete TERMSELF message.

For TERMSELF, the first three bytes in Request DATA must be X'010683' or X'810683'. Otherwise, the server uses the data as a host application program name.

For example, to terminate the current session, store in Request DATA either X'01068300F3' in the TERMSELF field, or X'C4C2C4C3C3C9C3E2' in the field containing the host application program name.

If the host rejects the TERMSELF message, the SNA server returns, in Reply DATA of the next RH, the sense code sent by the host. The sense code shows the type of SNA error.

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 17 | 9 | Transmission (TH) and Request/Response (RH) header |

### Common reasons for nonzero return codes

- The session has not yet been established.
- There is a message in the input queue that requires a protocol answer (definite response protocol in use). The message must be read before the CL function can be requested. The definite response indicators are automatically generated.
- The length of the data is more than 14 bytes.
- The host has set the workstation to a quiesce state.
- There are no input/output buffers available.
- Invalid session specified.
- The modem is powered off or the link is inoperative.

## Connect (CN function)

This function requests the use of an LU.

When the application program subsequently requests an RH function, flag1 shows whether the session is active:

**'{'** Circuit is established (LANDP for DOS using X.25 DLC).
**'}'** Circuit cannot be established or communication problems.
**'L'** The LU is active (for all other cases).

This request is now mandatory in LANDP for DOS programs. If you are unable to modify application programs that already exist and do not contain a CN request, you can use the `EHCCONN` program to achieve the same effect. For a description of this utility program, see the “Moving to the latest version of LANDP for DOS” section in the “Clients and servers” chapter of the *LANDP Programming Guide*.

| CPRB Field | Content/Description |
|---|---|
| Function code | CN |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag 1:<br>**'N'** No X.25 call packet is sent (when using X.25 DLC)<br>**' '** Requests the use of an LU and sends a call packet |
| 4 | 1 | Flag 5 (see note):<br>**'M'** Establishes only-one-response mode for this LANDP SNA session<br>**' '** Default mode for user application response handling |
| 7 | 1 | Flag 8:<br>**'E'** Application issues responses to the BID command (see "BID protocol" on page 39)<br>**' '** BID operation for this session is as specified during customization |

**Note:** If flag5 in Request PARMLIST is set to 'M', flag5='R' in an RH request is meaningless, and the application must send just one response for each chain (an SH request with flag5='+' or '−'). Negative responses can be sent after reading a first-in-chain, middle-of-chain, last-in-chain, or only-in-chain message. A positive response has to follow a last-in-chain or an only-in-chain message and must not be sent if a negative response has already been sent for the chain.

If flag5=' ' and the application is handling SNA responses (RH has flag5='R'), an SH request with flag5 set to '+' or '−' must be issued after each RH function.

### Common reasons for nonzero return codes

- The modem is powered off or the link is inoperative.
- Invalid session specified.
- The length is not set to zero.
- No circuits are available on the public network.
- A CN function was already requested.

## Define connection (DC function)

This function operates in the LANDP for DOS and OS/2 environments. It is significant only for an SNA server that uses X.25 DLC.

This function is used to change the characteristics of the connection. The defaults are defined during customization. The DC function can only be used when no connection is established for this LANDP SNA session (or for any of the LANDP SNA sessions belonging to the same SNA connection).

When the DC function has completed successfully, all LANDP SNA sessions for the SNA connection concerned, have the same connection characteristics. These changed connection characteristics are used when the next CN function is requested.

| CPRB Field | Content/Description |
|---|---|
| Function code | DC |
| Request DATA length | 108 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

For LANDP for DOS, Request DATA must contain 108 bytes with the information:

| Request DATA Values for LANDP for DOS | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 14 | Called subscriber's identification |
| 15 | 1 | Connection type:<br>**P** Permanent virtual circuit<br>**I** Incoming virtual circuit<br>**O** Outgoing circuit<br>**B** Both ways circuit |
| 16 | 64 | X.25 facilities used as defined by your network provider (PTT). Byte 16 is the length of the part of this field that is used, in binary. The contents of this field override the values defined during customization |
| 80 | 2 | Length used of the following two fields plus four in personal computer binary format |
| 82 | 8 | Called SNA PU name in EBCDIC, padded to eight bytes with the same character as in the host |
| 90 | 4 | Up to four bytes of application program data for the calling packet |
| 94 | 8 | The name of the PU or the calling application program, so that a called process can recognize the originator |
| 102 | 6 | XID value for this connection |

For LANDP for OS/2, Request DATA must contain 108 bytes with the information:

| Request DATA Values for LANDP for OS/2 | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 14 | Called subscriber's identification |
| 15 | 93 | Not used |

**Notes:**

1. Information in bytes 0 through 79 (LANDP for DOS) or 0 through 14 (LANDP for OS/2) can be obtained from the person responsible for ordering your X.25 connection or from the network provider (PTT).

2. LANDP for DOS: All fields are positional and fields not used must be filled with X'FF'.
3. LANDP for DOS: Data supplied in the DC function changes the data created during the configuration process.
4. LANDP for OS/2: All subscriber numbers that could be selected must be defined in the SNA provider and also defined using an X.25 remote directory. See the *LANDP Installation and Customization* book if you require further information.
5. LANDP for OS/2: Although bytes 15 to 107 are not used, values other than X'FF' are accepted to provide compatibility for migrated applications.

## Example

For LANDP for DOS, the example shown below contains a 108 byte data field that could be used with the DC function. For LANDP for OS/2, only the *number to be called* is relevant from this example.

- The number to be called, for example: 231020108.
  Bytes 0-14, desired number in ASCII padded with blanks:
  X'32333130323031303820202020202020'
- Connection type–outgoing.
  Byte 15, ASCII character 'O': X'4F'
- Facilities field, length three bytes, and requesting negotiable window size (X'43') with sending and receiving window size of seven.
  Bytes 16-79, padded with X'FF': X'03430707FFFF...'
- User data field length plus 4. The physical unit name, and application program data field are not used. The length is four, in personal computer integer format, followed by the fields not used filled with X'FF'.
  Bytes 80-81: X'0400'
  Bytes 82-89: X'FF...'
  Bytes 90-93: X'FF...'
- Calling physical unit ID is not used.
  Bytes 94-101: X'FF...'
- Unique XID number specified by the system administrator.
  Bytes 102-107: X'0200017C1D43'

## Common reasons for nonzero return codes

- The circuit has already been established.
- Request DATA length is not equal to 108.
- There are no input/output buffers available.
- X.25 cannot satisfy the request.
- Invalid session specified.

## Get status (GS function)

This function returns the communication status to the LANDP application program, and it shows if there are any messages pending. The Request DATA and Reply DATA lengths must be set to zero.

| CPRB Field | Content/Description |
|---|---|
| Function code | GS |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 (communication status). Possible values are:<br>**'0'** No session link or session established<br>**'1'** X.25 circuit established (DOS only)<br>**'P'** SSCP–PU session established (DOS only)<br>**'L'** LU–SSCP session established<br>**'S'** Start data traffic received<br>**'B'** Bind sent but not yet in session (DOS only) |
| 1 | 1 | Flag2 shows if any messages are pending:<br>**'0'** No message is pending<br>**'1'** Message pending |

### Common reasons for nonzero return codes

- Invalid session specified
- Data lengths are not zero

## Open (OP function)

This function shows an intention to use an SNA communication session. In the OP function the name of the host application program to communicate with is defined. The SNA server uses this name to generate an INITSELF SNA command to initiate the session (see “SNA connection process” on page 42). At the start of the session, the SNA server queues a message for the LANDP application program. When the LANDP application program subsequently requests an RH function, flag1 is '<', indicating that the session has started.

The OP function can also be used to change the BID protocol to values other than those defined during customization. If the session is already established, nothing is sent to the host. After a CL function or if the session is lost, the BID protocol values revert to those defined during customization. Setting flag8 = 'B', the BID protocol is handled by the LANDP application program, if flag8 not equal to 'B', the BID protocol is handled automatically by the SNA server.

| CPRB Field | Content/Description |
|---|---|
| Function code | OP |
| Request DATA length | 1 to 54 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 7 | 1 | Flag8<br>**'B'** BID handled by application program<br>**' '** BID handled by SNA server (default) |

## Request DATA values

You provide the name of the host application program with which you want to communicate, either as a EBCDIC character field or by specifying the complete INITSELF message.

If the entire INITSELF is specified, the first three bytes in Request DATA must be X'010681' or X'810681'. Otherwise, the server interprets the data as a host application program name with a length specified as Request DATA length.

For example, to connect with a host application program called DBDCCICS, store in Request DATA one of:

**INITSELF:** X'0106810040404040404040F308C4C2C4C3C3C9C3E2000000'

**Host application program name:**
X'C4C2C4C3C3C9C3E2'

If the host rejects INITSELF, the SNA server returns the sense code (sent by the host in the next RH function request), which shows the type of SNA error. The sense code is placed in Reply DATA.

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 17 | 9 | Transmission (TH) and Request/Response (RH) header |

## Common reasons for nonzero return codes

- ACTLU has not been received.
- The modem is powered off or the link is inoperative.
- Invalid session specified.

- The length of the data is more than 54 or less than 1 bytes.
- There is a message in the input queue that requires a protocol answer (definite response protocol in use). The message must be read before the OP function can be requested.
- There are no input/output buffers available.
- The corresponding communication server is not loaded.

## Query connection (QC function)

This function operates in the LANDP for DOS and OS/2 environments. It is significant only for an SNA server using X.25 DLC.

This function is used to obtain the current characteristics for the virtual circuit associated with the SNA connection to which the LANDP SNA session belongs. The QC function can be requested at any time. The data obtained can be saved for restoring the original connection characteristics after the use of a DC function.

| CPRB Field | Content/Description |
|---|---|
| Function code | QC |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 108 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 108 |
| Replied PARMLIST length | 26 |
| **Note:** Under LANDP for OS/2, *only* the subscriber number identification is returned. This is the same identification as the subscriber number that was previously called. The remaining fields should be ignored. | |

Reply DATA contains 108 bytes of information. The formats are the same as described in the DC function—see page 55.

### Common reasons for nonzero return codes

- Reply DATA length is not equal to 108 (X'006C')
- X.25 cannot satisfy the request
- Invalid session specified

## Read host (RH function)

This function reads a communication message. The message is stored in Request DATA. The Request PARMLIST flag fields contain session control information.

| CPRB Field | Content/Description |
|---|---|
| Function code | RH |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |

| CPRB Field | Content/Description |
|---|---|
| Reply DATA length | Maximum length of expected record or 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of record actually read or number of pending messages |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 4 | 1 | Flag5 setting. You can use the SH function to control SNA responses, see “BIND parameters” on page 38.<br>**’R’** Response managed by application program<br>**’ ’** Response managed by the SNA server |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 setting<br>**’L’** Logical unit session established<br>**’<’** BIND, STSN, and SDT received; see note 1 on page 62<br>**’>’** Loss of contact; see note 4 on page 63<br>**’{’** X.25 connection established; see note 3 on page 63<br>**’}’** Connection released or connection could not be established; see note 2 on page 63<br>**’ ’** Normal traffic |
| 1 | 1 | Flag2 setting<br>**’(’** First in chain (FIC)<br>**’)’** Last or only in chain (LIC or OIC)<br>**’+’** Middle in chain (MIC) |
| 2 | 1 | Flag3 setting<br>**’[’** Begin bracket (BB)<br>**’]’** End bracket (EB)<br>**’!’** Begin/end bracket (BB,EB). Only message of the bracket |
| 3 | 1 | Flag4 setting<br>**’X’** Change direction indicator (CDI) |
| 4 | 1 | Flag5 setting<br>**’F’** Message includes FMH header<br>**’ ’** Message without FMH header<br>**’+’** Positive response<br>**’–’** Negative response |
| 5 | 1 | Flag6 setting<br>**’ ’** Application program message<br>**’S’** Message from SSCP |
| 6 | 1 | Not used |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 7 | 1 | Flag8 setting<br>**'C'** Cancel from host<br>**'K'** Command from host.<br>**'B'** BID from host.<br>**'E'** BID from host (see "BID protocol" on page 39).<br>**' '** Data from host |
| 8 | 1 | Not used |
| 9 | 1 | Flag10 setting<br>**'S'** Selective cryptography<br>**'M'** Mandatory cryptography<br>**'V'** Cryptography verification command<br>**'C'** RU encrypted but not padded<br>**'P'** RU encrypted and padded before encryption<br>**' '** RU not encrypted |
| 17 | 9 | Transmission (TH) and Request/Response (RH) header |

## If reply DATA length is NOT equal to zero:

- The Replied DATA length field contains the actual length of the message returned in Reply DATA
- The flag fields in Reply PARMLIST are as shown

DOS

If Reply DATA length is equal to zero, then the Replied DATA length field contains the number of pending messages and Flag1 contains the communication status encoded as follows:

**'0'** No session established for SDLC or token-ring, or
No session and circuit established for X.25
**'1'** Circuit established for X.25
**'P'** Physical unit session established
**'L'** Logical unit session established
**'S'** Start data traffic received

The function GS (see page 58) provides the same information.

**Notes:**

1. When flag1 is returned as '<', it has two meanings. If the length is 0, a quiesce state has been released and no data was passed. If the length is not 0, a SDT has been received, and the first five bytes of Reply DATA contain the values of the counters and the flag received from the STSN followed by the BIND image in EBCDIC.

2. **LANDP for DOS:** When you are using X.25 DLC, if flag1 = '}' the Replied DATA length can be returned containing X'0002', which means an unexpected lost connection condition. The two bytes in Reply DATA contain the cause and diagnostic codes, respectively. The cause byte has the following meanings:

| Cause | Diagnostic Code Set by |
|---|---|
| X'FF' | X.25 Adapter. Consult the *IBM X.25 Co-Processor Adapter User's Guide*. This is not used with the X.25 Co-processor. |
| X'00' | The other end. |
| X'xx' | The network or the X.25 Co-processor. |

3. **LANDP for DOS:** When X.25 data link control is used and the circuit has been established, the application program reads a message with flag1 = '{'. If the circuit has been established because of receiving an incoming call, Reply DATA contains 92 bytes with the following information:

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 14 | Calling subscribers identification |
| 15 | 1 | Reserved |
| 16 | 64 | X.25 facilities included in the call packet. Byte 16 is the binary length of the significant facilities field |
| 80 | 8 | Called local session application program |
| 88 | 4 | Up to four bytes of application program data in the call packet |

4. When the loss of contact indicator is set (flag1 returned as '>') either a loss of application program contact or loss of contact with the host has occurred. Use the GS function to determine which is the case.

## Reply DATA

The message is stored without SNA headers and without code translation. If a SNA function management header exists, it is included and flag5 is set to 'F'.

When calling the RH function with Reply DATA length not equal to 0, you must specify a value that is at least equal to the maximum size of expected message.

An RH function with Reply DATA length = 0 takes a message out of the queue.

OS/2

## Common reasons for nonzero return codes

- The modem is powered off or the link is inoperative.
- Invalid session specified.
- The length of the data is more than 4096 bytes or less than the length of the data received.
- There are no messages in the input queue.
- There is a message in the SNA server input queue that requires a protocol answer (definite response). You must send this response before using RH.

## Release (RL function)

This function release the use of an LU. If all LUs are released, when using X.25 DLC, the virtual circuit is also released.

The session is released even if it has not been closed (CL function). Any pending host messages that the application program has not read so far are lost.

| CPRB Field | Content/Description |
|---|---|
| Function code | RL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

### Common reasons for nonzero return codes

- The circuit has not yet been established.
- There are no input/output buffers available.
- Invalid session specified.
- The modem is powered off or the link is inoperative.
- No previous CN requested.

## Send host (SH function)

This function sends a message from the LANDP application program to the communication server for transmission to the host. The server queues the message for transmission. You can use the SH function to send SNA commands and to control SNA responses, see “BIND parameters” on page 38.

| CPRB Field | Content/Description |
|---|---|
| Function code | SH |
| Request DATA length | Data length in bytes |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Not used |
| 1 | 1 | Flag2 setting<br>**' '** Only in chain (OIC)<br>**'('** First in chain (FIC)<br>**'+'** Middle in chain (MIC)<br>**')'** Last or only in chain (LIC)<br>**'C'** Cancel chain |
| 2 | 1 | Flag3 setting<br>**' '** Begin bracket (BB) is also sent if bracket protocol is in use<br>**']'** End bracket (EB) is sent if it is allowed by the BIND |
| 3 | 1 | Not used |
| 4 | 1 | Flag5 setting<br>**' '** Message without function management header (FMH)<br>**'F'** Message includes FMH header<br>**'+'** SNA positive response to previously received message<br>**'–'** SNA negative response to previously received message |
| 5 | 1 | Flag6 setting<br>**' '** Message to application program<br>**'S'** Message to SSCP |
| 6 | 1 | Not used |
| 7 | 1 | Flag8 setting<br>**'R'** RTR. Application program is ready to receive the unsolicited message previously denied by SNA server<br>**'K'** The supported commands to the host are:<br>• SIGNAL to LU<br>• LUSTAT to LU<br>• LUSTAT to SSCP |
| 8 | 1 | Flag9 setting<br>**'D'** Definite response solicited. To be used only if the BIND allows optional response |
| 9 | 1 | Flag10 setting<br>**'C'** RU encrypted but not padded<br>**'P'** RU encrypted and padded before encryption<br>**' '** RU not encrypted<br>**'S'** Used in encryption<br>**'M'** Used in encryption<br>**'V'** Used in encryption |

## Request DATA

Store the message to be sent in Request DATA. The SNA server creates and adds the SNA transmission and request headers. Data conversion ASCII to EBCDIC might be necessary.

The command RU must be coded into Request DATA.

### Request DATA length

The request data length must not exceed X'1000' bytes. For specific flag settings the following request DATA lengths are not required:

| Flag setting | Request DATA length |
|---|---|
| Flag5 = '+' | 0 (or 8 is BIND with cryptography) |
| Flag5 = '–' | 2 or 4 (or 0 is BIND or CRV with cryptography) |
| Flag7 = 'R' | 0 |
| Flag7 = 'K' | 5 |

**Note:** The length of the message being sent must not exceed the maximum inbound RU size received in the BIND. Otherwise the host might close the session or reject the message with unpredictable results for the application program.

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 17 | 9 | Transmission (TH) and Request/Response (RH) header |

### Common reasons for nonzero return codes

- The host has set the workstation to a quiesce state.
- The modem is powered off or the link is inoperative.
- Invalid session specified.
- The length of the message for the server is greater than 4096 bytes, or does not match the length of the type of message being sent.
- There is an SNA bracket or chain protocol violation.
- An attempt was made to cancel a chain that never existed.
- There is a message in the SNA server input queue that requires a protocol answer (definite response). You must read this message before using SH.
- There is a chain or a segment being received.
- An invalid cryptography request.
- There is a response pending to be sent.
- A response to a message previously sent (with definite response) has not arrived.
- There are no input/output buffers available.

## Server information (ZI function)

This function provides information about the server.

| CPRB Field | Content/Description |
|---|---|
| Function code | ZI |
| Request DATA length | 0 |

| CPRB Field | Content/Description |
|---|---|
| Request PARMLIST length | 26 |
| Reply DATA length | 94 for each LU expected |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Number of LUs provided |
| 2 | 1 | Workstation type:<br>**0** DOS<br>**1** OS/2<br>**2** Windows<br>**3** Linux |
| 3 | 1 | Completeness of the Reply DATA:<br>**' '** Data is complete<br>**'+'** Data is incomplete |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | LU name |
| 8 | 1 | LU address |
| 9 | 1 | LU status<br>**'W'** Waiting<br>**'0'** No session link or session established<br>**'1'** X.25 circuit established<br>**'P'** SSCP–PU session established<br>**'L'** LU–SSCP session established<br>**'S'** Start Data Traffic received |
| 10 | 2 | LANDP session—for example, 01 for SNA01. |
| 12 | 2 | PWS identifier |
| 14 | 2 | Process identifier |
| 16 | 1 | BidCommMgm<br>**0** Managed by server<br>**1** Application responds RTR<br>**2** Application + response or − response with X'0813' |
| 17 | 27 | Bind |
| 44 | 12 | PU name |
| 56 | 8 | DLC: 'X25DLC', 'SDLC', 'IBMTRNET', 'TN3270', 'MPTN', 'TCP/IP' or 'DCA' |
| 64 | 1 | LU type: 0, 1, or 2 |
| 65 | 1 | Pooled (0 no pooled, 1 pooled) |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 66 | 2 | Pool group |
| 68 | 20 | Destination address (for X25DLC) |
| 88 | 6 | XID (for X25DLC) |

## DOS compression server

The aim of SNA compression is to decrease the size of normal-flow function management data (FMD) request units (RUs). This process is very useful when the maximum RU size is bigger than the data link control (DLC) segment size, since the number of segments transmitted decrease. An RU whose decompressed length is greater than can be expressed in the compression header is not compressed.

Compression is performed before encryption, and decompression after decryption. It is therefore not possible to use both compression and encryption for the same LU, if the application is responsible for encrypting the data, while the SNA server handles compression. Under LANDP for OS/2, it is possible to use both compression and encryption to the same LU provided the encryption is handled by the SNA server using Communications Server for OS/2 Warp, rather than the application. Under LANDP for OS/2, SNA compression is performed by Communications Server for OS/2 Warp,

Under LANDP for DOS, SNA compression is performed by a compression server exit, supplied with LANDP for DOS. The remainder of this section applies to **LANDP for DOS**.

The supplied server has two functions, CM (compress) and UC (decompress). If you want to implement an algorithm different from that used by the supplied server, you can write your own server to this same interface.

### Request-Unit compression

The compression interface used by the supplied server is a form of length-checked compression, called run-length encoding (RLE). Run-length encoding eliminates strings of repeated bytes. The first three bytes of the RU are the compression header, and the remainder is the compressed data.

The header format is:

**Byte 0** X'11' to denote RLE.

**Bytes 1–2** Length in binary of the noncompressed RU.

The first byte after the compression header is a string control byte (SCB), with the following format:

**Bits 7–6** SCB type:

- **00** Raw data. The following bytes are not compressed. The count field (bits 5–0) gives the number of bytes.
- **01** Reserved.
- **10** Master character. The count field (bits 5–0) indicates the number of blank characters (X'40') compressed.
- **11** Duplicate character. The character that follows this SCB appears *n* times in the raw data.

**Bits 5–0** Count, *n*, in binary. The number of noncompressed bytes that follow (in the case of SCB type 00), or that are generated when this SCB sequence is decompressed (types 10 and 11).

If the RU is not exhausted, another SCB follows *n+1* bytes after this SCB.

SCBs cannot span RUs. If the last SCB in a RU is raw data (type 00), all the raw data to which the SCB refers is in that RU. The master-character SCB (type 10) can be the last byte in a RU. A duplicate-character SCB (type 11) always has its accompanying duplicate character in the same RU.

### Example of compressed data

The data before compression is X'E5E3E7E8 D5F0F5D3 F2F0F05C 5C5C5C5C 5C5C5C5C 40404040 40404040 40404040 40404040 40404040'

The data after compression is X'1100280B E5E3E7E8 D5F0F5D3 F2F0F0C9 5C94' made up as follows:

| | |
|---|---|
| **X'11'** | RLE compression. |
| **X'0028'** | Length of decompressed data (40 bytes). |
| **X'0B'** | The following 11 bytes are raw data. |
| **X'E5E3E7E8 D5F0F5D3 F2F0F0'** | Raw data. |
| **X'C9'** | Duplicate-character SCB. The following character appears 9 times in the raw data. |
| **X'5C'** | The duplicated character. |
| **X'94'** | Master-character SCB. 20 blank characters appear in the raw data. |

## Writing your own compression server

This section provides guidelines to help you use the necessary information in the Request CPRB fields and understand the information you must return in the Reply CPRB fields. If you need more information about the CPRB fields, see Appendix A, “Connectivity programming request block” on page 669.

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | 0 | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 26 | 2 | 0 | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | EHCCOMP | Server name |

The following fields are variable and are discussed in each function request description:
- Function code
- Request DATA length
- Reply DATA length

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 44 | 2 | | Replied PARMLIST length |
| 46 | 2 | | Replied DATA length |

If the request is successful, the *router return code* and the *server return code* must both be set to X'00000000'. The return values in Reply PARMLIST and Reply DATA are ignored if there is an error.

The following fields are variable and are discussed in each function request description:
- Replied PARMLIST length
- Replied DATA length

**DATA:** Request DATA contains data to be compressed or decompressed.

**Note:** The application program must have enough space in Reply DATA to receive the data. Otherwise a nonzero return code should be sent.

## Request reference

This section describes the functions that the SNA server can request from the Compression server. They are listed in alphabetical order of function code.

### Compress data (CM function)

This function compresses the data in Request DATA, returning the compressed data in Reply DATA. The lengths of these areas must be sufficient to hold the data.

| CPRB Fields | Content/Description |
|---|---|
| Function code | CM |
| Request DATA length | Length of data to compress |
| Reply DATA length | Maximum expected length of compressed data |
| Replied DATA length | Actual length of compressed data |

### Decompress data (UC function)

This function decompresses the data in Request DATA, returning the decompressed data in Reply DATA. The lengths of these areas must be sufficient to hold the data.

| CPRB Fields | Content/Description |
|---|---|
| Function code | UC |
| Request DATA length | Length of data to decompress |
| Reply DATA length | Maximum expected length of decompressed data |
| Replied DATA length | Actual length of decompressed data |

## Example program that uses the SNA server

This section presents an example application program to set up the session, exchange data, and close the session. It is valid irrespective of the operating system on which it is installed, and works under any DLC (Data Link Control). The example provides guidelines and hints to get a suitable structure, and to improve performance. The following functions are used inside the program:

**CL** Close function to SNA server
**CN** Connect function to SNA server
**OP** Open function to SNA server
**RH** Read host function to SNA server
**RL** Release function to SNA server
**SH** Send host function to SNA server
**WM** Wait multiple function

The sample application program follows:

```
Begin

  WM for &&;amp; of SNAxx       /* Where xx is the session ID you are using */
  CN
  if return_code not zero
     then check definitions, configuration, installation and equipment
  WM
```

```
RH
if return_code not zero
   then check logic and CPRB parameters
   else { if flag1 = '{'
             then { WM

                    RH
                    if return_code not zero
                       then check logic and CPRB parameters
                  }
          if flag1 = '}'
             then check definitions, configuration, installation
                  and equipment
          if flag1 = 'L'
             then { if replied_data_length = 5
                       then { /* The SNA Server is in DOS LANDP */
                              Store SSCP_ID
                            }

                    if application is responsible for opening session
                       then { OP
                              if return_code not zero
                                 then check logic and CPRB parameter
                            }
                  }
        }
WM

RH
if return_code not zero
   then check logic and CPRB parameters
   else { if negative response to INITSELF
            then check with the administrator the correct definition
                 in host tables.
          if flag1 = '<'
             then { /* The session is correctly established and
                       is able to perform RH and SH as needed */
                    store BIND and STSN.
                  }
        }

RH or SH
RH or SH
.
. .
RH or SH
When data exchange finishes
     then { CL
            WM
            RH
            if return_code not zero
               then check logic and CPRB parameters
               else /* The application program should receive '>'
                       in flag1. This means that the session is
```

```
                              down  */
                RL
              }
End.
```

At any time, the application program can receive a “session lost” message with flag1 = '>', without having requested a CL function. If the application program wants to be in session again, it must reissue an OP function request.

Then, when an application program receives a message with flag1 = '}' without having requested a RL function, some kind of disconnection has occurred. To work again, the application program must reissue a CN function request. That means that it has to start from the beginning. As in the case when the server has been unloaded, the application program must wait for the loading of the server and issue a CN function request.

## Migration considerations

These are the differences between the LANDP for DOS or FBSS (DOS) server and the LANDP for OS/2, LANDP for Windows, or LANDP for Linux servers:

- The number of X.25 physical units (PUs) supported by LANDP depends on the limitations of the underlying communications subsystem.
- The LANDP for DOS and LANDP for OS/2 SNA servers require the connect (CN) and release (RL) functions to be used. (Existing LANDP for DOS applications that do not use these functions are supported for compatibility only.) You should add these functions to your LANDP for DOS and FBSS (DOS) applications to enable their portability to LANDP for OS/2.

  Existing applications using switched lines (X.25) are not affected by this requirement because CN and RL are already mandatory.
- To get the session status, on LANDP for OS/2 SNA, the server function GS must be used instead of the zero-length RH function which can be used with the LANDP for DOS or the FBSS (DOS) SNA server. The LANDP for OS/2 SNA server treats the zero-length RH function as a normal RH function, so it can be used to remove the current message from the message queue. The GS function is also available on LANDP for DOS.
- The RH function, when reading the ACTLU message with LANDP for DOS or FBSS (DOS), returns five bytes of data with the SSCP name, if this is received from the host. This information is not available to the LANDP for OS/2 SNA server, so it returns no data.

## SNA server

# Chapter 3. The cryptographic interface

This chapter describes the cryptographic application programming interface supported by LANDP for DOS and OS/2. For further information about this interface, see the *IBM Transaction Security System, Programming Guide and Reference* book. The SECY cryptographic server is supplied as a feature with the 4755 TSS encryption card.

Cryptography is based on two fundamental operations: *encryption* and *decryption*.

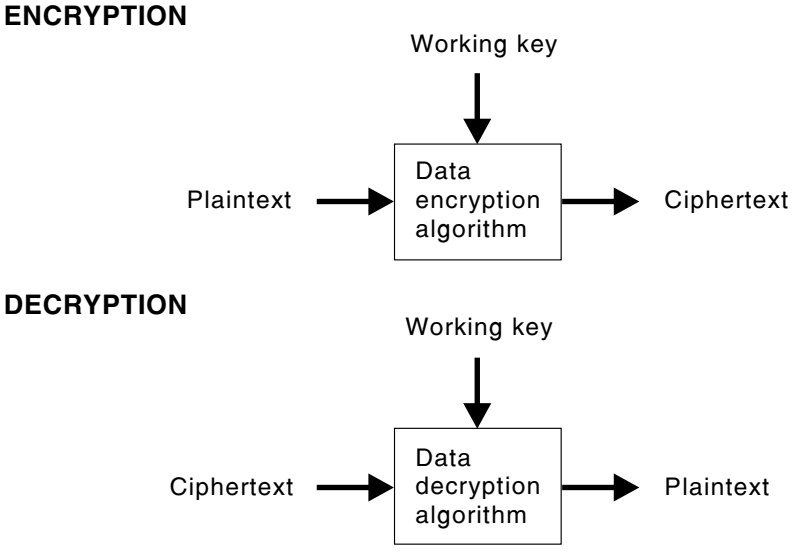



*Figure 1. Encryption and Decryption of Data Using a Key*

When the key and the text have been processed by the data encryption algorithm, the resulting cipher text (encrypted text) can be returned to its original state only if the original working key is available.

The key is an 8-byte value. The text can be any data, even another key.

The algorithm used by the cryptographic interface is based on the *data encryption standard* (DES), as published in the *Federal Information Processing Standard 46* (FIPS PUB 46). The cipher block chaining is described in *Federal Information Processing Standard 81* (FIPS PUB 81).

© Copyright IBM Corp. 1992, 2002

*Table 6. Function Codes used by the Cryptographic Interface*

| Function code | Description | Env. | Page |
|---|---|---|---|
| X'0030' | Encrypt | 02-- | 80 |
| X'0031' | Decrypt | 02-- | 81 |
| X'0040' | Random number generation | 02-- | 82 |
| X'0050' | Session key add | 02-- | 83 |
| X'0051' | Session key update | 02-- | 84 |
| X'0052' | Key record delete | 02-- | 84 |

The first and second columns give the function code and the name of the function. The third column shows the operating environment of the function. "02--" means that it is available from LANDP for DOS and OS/2 servers. The last column refers to the page where you can find the function described.

## Cipher blocks

You can encrypt or decrypt up to 4096 bytes of data with one cryptographic interface function request. However, it is important to remember that the data encryption algorithm always processes text in 8-byte blocks.

If you submit a data string for encryption that is not a multiple of eight bytes, the data is padded.

## Cipher block chaining

Repetitive patterns in multiple blocks of plain text can result in repetitive patterns in the cipher text. To make the cipher text more resistant to analysis cipher block chaining is used.

When the encryption of a given 8-byte block is completed, the encrypted data is used to modify the *next* block's plain text. The modification is an exclusive-or (XOR) on the two blocks. Then the modified plain text is encrypted. This is repeated for each successive block.

The process is reversed when data is decrypted. After a given 8-byte block has been decrypted, the previous eight bytes of the cipher text are used to recover the plain text.

The first block of data is a special case. As there *is no previous block*, you must supply one. You do this by including an *initial chaining value* (ICV) of eight bytes with the encrypt or decrypt request.

The technique of using an ICV can be used also on a single 8-byte block to encrypt or decrypt it.

Both the key and the ICV must be known to decrypt data.

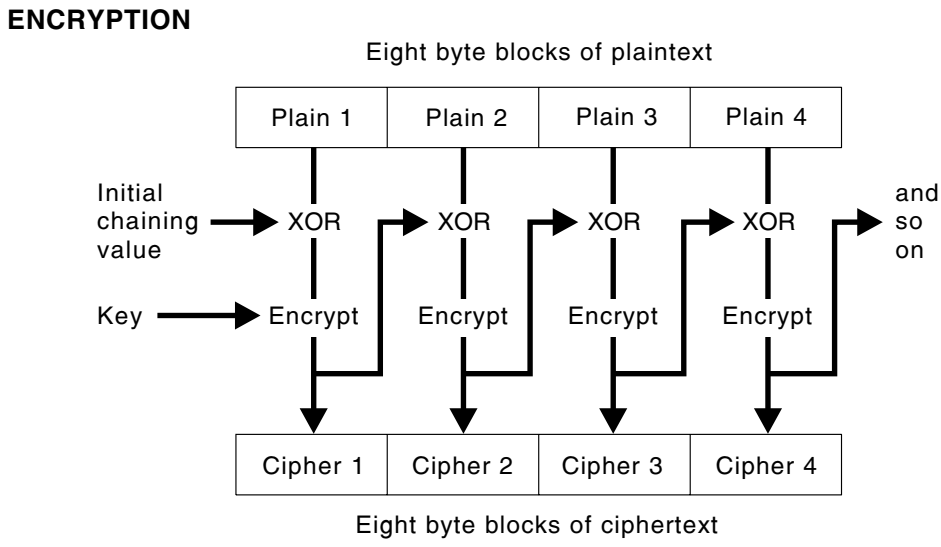



*Figure 2. Encryption Using Cipher Block Chaining*

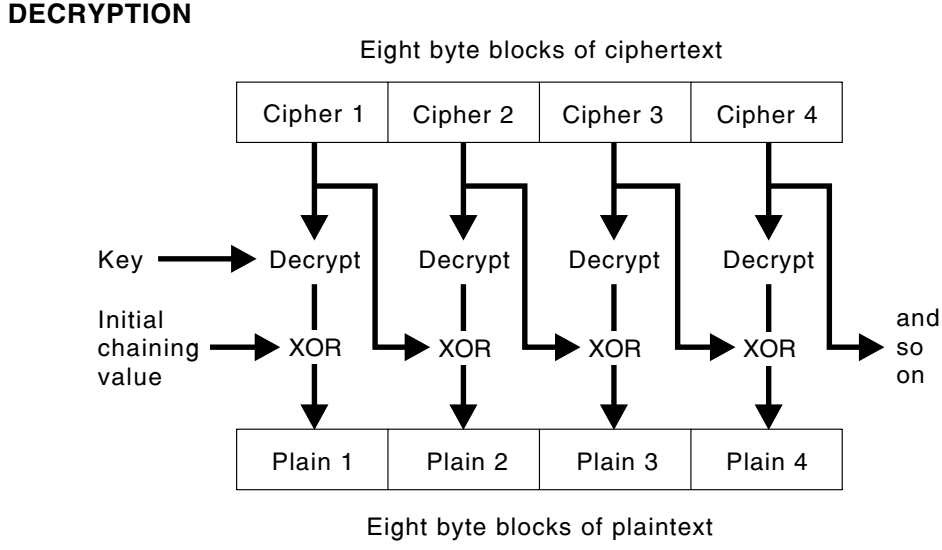



*Figure 3. Decryption to Recover the Original Data*

## Using encryption and decryption

When the encrypted data must be processed at another location, the following is required:

- The data encryption key (in this context the session key) must be available at the other location.

- The other location must have access to the data encryption algorithm and be able to do cipher block chaining.
- The initial chaining value, ICV, must be known at both locations. There are several ways to do this. You should consult one or more of the books on encryption that are listed in the “Bibliography” on page 703 before you use the cryptographic interface.

## The LANDP cryptographic interface

The cryptographic interface described here is used by the 3270 emulator, the 3287 printer emulator, and the remote change management services (RCMS) for session-level encryption. The cryptographic interface can be used in the following ways:

- If SNA session-level encryption is selected as a host communication parameter, the communication with the host is encrypted. See “Cryptography support in the SNA server” on page 48 for more information.
- By application programs to encrypt and decrypt data.
- The institution using LANDP as a basis for developing a cryptographic *server*.

## Key records

The cryptographic interface maintains a set of *key records* containing the cryptographic key and other important information. There are add, update, and delete functions available to operate on the key records.

The key records contain:

**Key label** The key label identifies the key record. The value in this field is a non-blank, 16-byte ASCII character string, padded on the right with blanks, X'20'.

**Key type** This field contains a two-byte value used to show the key type associated with this key record:

| | |
|---|---|
| **X'4000'** | Session key. |
| **X'0200'** | Data encryption key. |
| **X'0800'** | Key encrypting key. |

**Input key** This field contains an 8-byte key that is normally encrypted under the key encrypting key contained in the referenced key record. This key is used as the input key in the session key add function.

**Initial chaining value (ICV)**
This field contains an 8-byte ICV associated with this key label.

**Pad character** This field contains the padding character that is used to fill the last block of the input data up to an 8-byte boundary. Valid values are between X'00' and X'0F'.

**Reference key label**
The reference key label identifies the key record containing the key encrypting key used with the session key add function when creating a session key record. The value in this field must be a non-blank, 16-byte ASCII character string.

**Actual chaining value**
This entry is dynamically updated when a cipher block chain is processed.

## Using the cryptographic interface

This section provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields. If you need more information about the CPRB fields, see Appendix A, “Connectivity programming request block” on page 669.

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST |
| 20 | 2 | | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 26 | 2 | | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | SECY | Server name |

The following fields are variable and are discussed in each function request description:
- Function code
- Request DATA length
- Request PARMLIST length
- Reply DATA length
- Reply PARMLIST length

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 44 | 2 | | Replied PARMLIST length |
| 46 | 2 | | Replied DATA length |

If the request was successful, the *router return code* and the *server return code* are both X'00000000'. In all other cases, see the appropriate section in the *LANDP Problem Determination* book to see if you should take any action. The return values in Reply PARMLIST and Reply DATA should be ignored if there is an error.

The following fields are variable and are discussed in each function request description:
- Replied DATA length
- Replied PARMLIST length

**DATA:** Request DATA and Reply DATA contain data to be sent to or received from the server. The use of these areas is explained in the description of each function request. Global segments must not be used for the data.

**PARMLIST:** The Request PARMLIST and Reply PARMLIST fields for the cryptographic interface are explained in the function request descriptions.

## Request reference

This section describes the functions that client programs can request from the cryptographic interface. They are listed in order of function code.

## Encrypt (X'0030' function)

This function encrypts any data. The specified key label identifies the associated key record and thus the encrypting key to be used (see “Key records” on page 78).

| CPRB Field | Content/Description |
|---|---|
| Function code | X'0030' |
| Request DATA length | Length of plain text |
| Request PARMLIST length | 52 |
| Reply DATA length | Cipher-text buffer size |
| Reply PARMLIST length | 2 |
| Replied DATA length | Cipher-text length |
| Replied PARMLIST length | 2 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 16 | Key label. Used to identify the record containing the encryption key. A key record with this key label must be present |
| 16 | 3 | Reserved. Must be initialized to X'000003' |
| 19 | 1 | ICV flag (numeric). Determines which ICV to use for this operation:<br>**0** Taken from the Request PARMLIST<br>**1** Taken from the ICV entry in the key record<br>**2** Taken from the actual chaining value of the key record |
| 20 | 8 | Initial chaining value. The ICV used in the encryption process if ICV flag = 0 |
| 28 | 24 | Reserved. This must be initialized to 24 bytes of X'00' |

### Cipher block chaining

Encryption of a plaintext string can be broken into segments. When processed in this fashion, the segments must be processed first to last, and all segments except the last must be a multiple of eight bytes. The initial chaining value for the first segment is often the initial chaining value in the key record.

The initial chaining value used for following segments must be obtained from the last eight bytes of the preceding cipher-text segment. This may be done either by the workstation application program or by using the actual chaining value that the interface maintains in the key record. In the latter case, the ICV flag is set to 2.

### Request DATA values

The plain text must be stored in the Request DATA buffer. If the length of the request buffer is not a multiple of eight, it is padded.

The interface extends the plain text before encryption with one to seven bytes where all except the last byte are X'00'. The last byte is an integer with a value equal to the number of bytes being added to the plain text.

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Shows if padding has occurred (numeric value):<br>**0** Plain text is *not* extended<br>**1** Plain text is extended |
| 1 | 1 | Reserved. Contains X'00' |

### Reply DATA values

The cipher text is returned in the Reply DATA buffer. The cipher text can be longer than the input plain text because of padding to an 8-byte boundary. If the Reply DATA length is insufficient, the function is ended.

## Decrypt (X'0031' function)

This function decrypts cipher text. The specified key label identifies the key record containing the encryption key. Parameters in the key record can be overridden by the specifications in Request PARMLIST (see “Key records” on page 78).

| CPRB Field | Content/Description |
|---|---|
| Function code | X'0031' |
| Request DATA length | Length of cipher text |
| Request PARMLIST length | 52 |
| Reply DATA length | plaintext buffer size |
| Reply PARMLIST length | 0 |
| Replied DATA length | Length of plain text |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 16 | Key label. Used to identify the decryption key. A key record with this key label must be present in the key storage |
| 16 | 1 | Reserved |
| 17 | 1 | This value shows if padding should be removed (numeric value):<br>**0** The plain text has no padding<br>**1** Padding should be removed from the plain text |
| 18 | 1 | Reserved. Must be initialized to X'03' |
| 19 | 1 | ICV flag (numeric). Determines which ICV to use for this operation:<br>**0** Taken from the Request PARMLIST (offset 20)<br>**1** Taken from the ICV entry in the key record<br>**2** Taken from the actual chaining value of the key record |
| 20 | 8 | Initial chaining value. The ICV used in the decryption process if ICV flag = 0 |
| 28 | 24 | Reserved. This must be initialized to 24 bytes of X'00' |

### Cipher block chaining

Decryption of a cipher-text string can be broken into segments. When processed in this fashion, the segments must be processed first to last. The initial chaining value for the first segment is often the initial chaining value in the key record.

The initial chaining value used for following segments must be obtained from the last eight bytes of the preceding cipher-text segment. This can be done either by the application program or by using the actual chaining value which the interface maintains in the key record. In the latter case, the ICV flag is set to 2.

### Request DATA values

The cipher text is the data stored in the Request DATA buffer. The length of the cipher text must be a multiple of eight bytes.

### Reply DATA values

The plain text is returned in the Reply DATA buffer. The plain text in the Reply DATA buffer is of the same length as the input cipher text or shortened because of removal of padding. If the Reply DATA length is insufficient, the function is ended.

## Random number generation (X'0040' function)

This function generates an 8-byte random number.

| CPRB Field | Content/Description |
|---|---|
| Function code | X'0040' |
| Request DATA length | 0 |
| Request PARMLIST length | 10 |
| Reply DATA length | 0 |

| CPRB Field | Content/Description |
|---|---|
| Reply PARMLIST length | 8 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 8 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Reserved. This field must contain X'01' |
| 1 | 9 | Reserved. Must be initialized to all X'00' |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | The generated 8-byte random number (key) |

## Session key add (X'0050' function)

This function stores a new session key.

| CPRB Field | Content/Description |
|---|---|
| Function code | X'0050' |
| Request DATA length | 0 |
| Request PARMLIST length | 64 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 16 | Key label. Used to identify the session key record `TMKppnnKS` |
| 16 | 2 | Key type. Must contain X'4000' |
| 18 | 8 | Input key. Eight byte key encrypted under a key-encrypting key (named at offset 48) |
| 26 | 16 | Reserved. Must be initialized to 16 bytes of X'00' |
| 42 | 2 | Reserved. Must be initialized to X'03FF' |
| 44 | 4 | Reserved. Must be initialized to four bytes of X'00' |
| 48 | 16 | Referenced key label. Used to identify the TMK key encrypting key `TMKnnpp`. See page 85 |

The input key field (offset 18) must contain an encrypted session key that is re-encrypted under the master key using the key-encrypting key name supplied (offset 48), and stored into a new key record named at offset 0.

Error conditions occur when:
- The key label is already present
- The key record storage is full

## Session key update (X'0051' function)

This function is used to modify the initial chaining value in the session key record.

| CPRB Field | Content/Description |
|---|---|
| Function code | X'0051' |
| Request DATA length | 0 |
| Request PARMLIST length | 42 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 16 | Key label. Used to identify the session key record |
| 16 | 18 | Reserved. Must be initialized to 18 bytes of X'00' |
| 34 | 8 | Initial chaining value. The new initial chaining value for the session key record |

An error results if the named session key record does not exist.

## Key record delete (X'0052' function)

This function deletes a key record.

| CPRB Field | Content/Description |
|---|---|
| Function code | X'0052' |
| Request DATA length | 0 |
| Request PARMLIST length | 42 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 16 | Key label. Identifies the key record to be deleted |

An error results if the key label does not exist.

## Example of SNA session-level encryption using the cryptographic interface

The abbreviations used in this example are:

**MK** Master key installed in the workstation
**TMK** Key encrypting key used to protect session key transfer (the TMK must be installed in the host *and* the workstation)
**KS** Session key
**CRV** Cryptography verification
**ICV** Initial chaining value
**RU** Request unit
**nn** Session ID
**pp** Workstation ID
**RH** Receive host function in the SNA server
**SH** Send host function in the SNA server
**e*** Denotes encryption

After the MK and the SNA defined TMK key encrypting key are installed in the workstation, an application program can establish a cryptographic session with the host through the procedure described below.

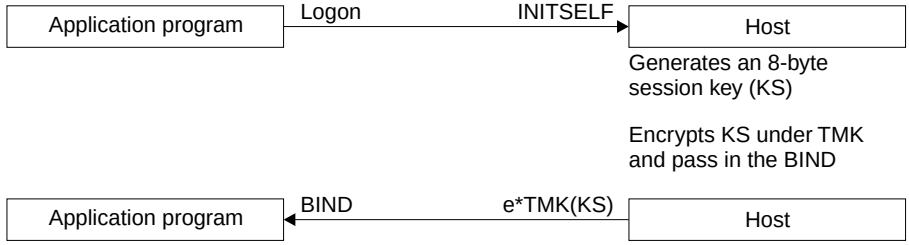
When an application program communicates using SNA session-level encryption, the host generates a KS which is sent to the application program in the BIND. The KS is sent encrypted under the TMK. When an application program, for example the 3270 emulator, detects a session that requires encryption, it can use a cryptographic server for key management, encryption, and decryption.

The encrypted KS must be securely stored in the workstation. The previously stored TMK is used to recover the KS which is then encrypted under the MK for storage in the key storage data set. The KS is now available for encryption and decryption of following session data transfers.

**Note:** See “Cryptography support in the SNA server” on page 48 for detailed information.

## SNA session-level encryption process

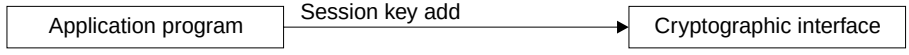
1. Initiate the communication process.




   The application program has a communication message available. An RH function returns the values:

```
Flag10              = S (Selective cryptography)
                      M (Mandatory cryptography)
Replied DATA length = X'0008'
Reply DATA contains:  e*TMK(KS)
```

2. Import the KS, that is, e*TMK(KS) to e*MK(KS).

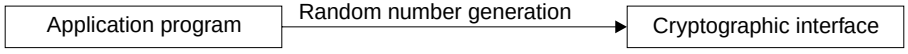



| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 16 | Key label. TMKppnnKS |
| 16 | 2 | Key type. Set to X'0400' meaning session key |
| 18 | 8 | Input key. Eight byte encrypted KS received from the host. e*TMK(KS) |
| 26 | 8 | Reserved. Must be initialized to eight bytes of X'00' |
| 34 | 8 | Initial chaining value. Eight bytes of nulls (X'00') |
| 42 | 1 | Reserved. Must be initialized to X'03' |
| 43 | 1 | Pad value. X'FF' |
| 44 | 4 | Reserved. Must be initialized to four bytes of X'00' |
| 48 | 16 | Reference key label. TMKnnpp |

   If the key label TMKppnnKS already exists, delete it (see “Key record delete (X'0052' function)” on page 84) and try again to make the addition. If the process is unsuccessful for another reason, for example the TMK does not exist, a negative response, with sense code X'0848', is sent to the host using the SH function with the parameters:

```
Request DATA length = X'0000'
Flag5               = -
Flag10              = S/M (Selective/Mandatory)
```

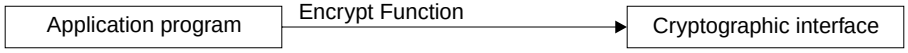
3. If the session key add function was successful, generate an ICV for a communication exchange with the host to test the received KS. The ICV is used in following session data RU encryption.




| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag set to X'00' to get an 8-byte, odd parity, random number |
| 1 | 1 | Reserved. Must be set to X'00' |
| 2 | 8 | ICV, set to X'0000 0000 0000 0000' |

The returned 8-byte random number can be considered as two 4-byte quantities called N1 and N2. The concatenation N1||N2 is equal to the returned number in offset 2.

4. Obtain e*KS(N1||N2) by using the encrypt function.

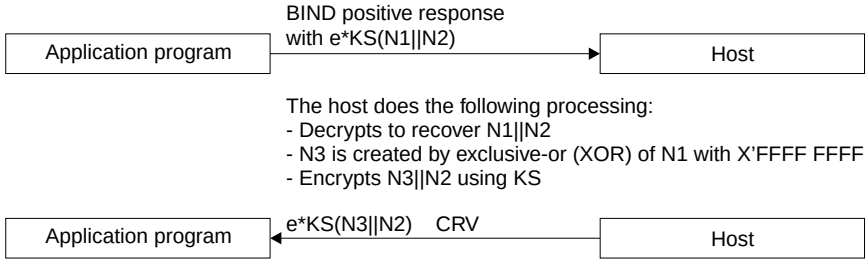



| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 16 | Key label. TMKppnnKS |
| 16 | 2 | Reserved |
| 17 | 1 | Pad value. X'FF' |
| 18 | 8 | Reserved. Must be initialized to X'03' |
| 19 | 1 | ICV flag. Must be initialized to X'00' |
| 20 | 8 | Initial chaining value. Eight bytes of nulls (X'00') |
| 28 | 24 | Reserved. Must be initialized to 24 bytes of X'00' |

5. A response is sent to the host.
   a. If a positive response should be sent, the parameters for the SH command are:

```
Request DATA length = X'0008'
Flag10              = S/M (Selective/Mandatory)
Flag5               = +

DATA contains N1||N2
```




b. If an error is discovered, a negative response is sent to the host (sense code X'0848') to terminate the session:

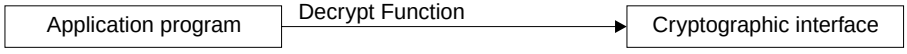
```
Request DATA length = X'0000'
Flag10              = S/M (Selective/Mandatory)
Flag5               = -
```

6. The application program has a communication message available. The RH function returns the following parameters.

```
Flag10              = V (CRV received)
Replied DATA length = 8

Reply DATA contains:  e*KS(N3||N2)
```

7. Obtain N3||N2 by using the decrypt function.




| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 16 | Key label. TMKppnnKS |
| 16 | 2 | Reserved |
| 17 | 1 | Pad value. X'00' |
| 18 | 8 | Reserved. Must be initialized to X'03' |
| 19 | 1 | ICV flag. Must be initialized to X'00' |
| 20 | 8 | Initial chaining value. Eight bytes of nulls (X'00') |
| 28 | 24 | Reserved. Must be initialized to 24 bytes of X'00' |

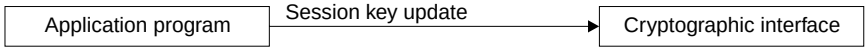
8. Exclusive-or N3 with X'FFFF FFFF' to get N4.

   Compare N4||N2 and N1||N2.

   a. If they are equal, use SH to send a positive response to the host with the parameters:

```
Request DATA length = X'0000'
Flag5               = +
Flag10              = V
```

Use the session key update function to insert N1||N2 as the ICV value.

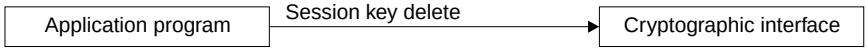



| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 16 | Key label. TMKppnnKS |
| 16 | 18 | Reserved. Must be initialized to 18 bytes of X'00' |
| 34 | 8 | Initial chaining value. N1||N2 |

b. If N4||N2 is not equal to N1||N2, a negative response (sense code X'0848') is sent to the host with an SH command with the parameters:

```
Request DATA length = X'0000'
Flag5               = -
Flag10              = V
```

The session key is subsequently deleted:




| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 16 | Key label. TMKppnnKS |

The use of the SNA session-level encryption is terminated.

## Normal request unit ciphering operations

Plain text and cipher text can be returned.

***Decrypt:*** Use of the decrypt function returns plain text.

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 16 | Key label. TMKppnnKS |
| 16 | 1 | Reserved |
| 17 | 1 | Set to X'00' to show that no padding is used |
| 18 | 1 | Reserved. Must be initialized to X'03' |
| 19 | 1 | ICV flag. The value in this field is set to X'01' |
| 20 | 8 | Initial chaining value. Arbitrary, not used |
| 28 | 24 | Reserved. This must be initialized to 24 bytes of X'00' |

Decryption of a cipher-text string can be broken into segments; see “Cipher block chaining” on page 81.

***Encrypt:*** Use of the encrypt function returns cipher text.

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 16 | Key label. TMKppnnKS |
| 16 | 1 | Reserved |
| 17 | 1 | This field contains X'FF' to specify that the value specified in the key record is to be used. |
| 18 | 1 | Reserved. Must be initialized to X'03' |
| 19 | 1 | ICV flag. The value in this field is set to X'01' to specify that the ICV value is taken from the ICV entry in the key record. |
| 20 | 8 | Initial chaining value. Arbitrary, not used. |
| 28 | 24 | Reserved. This must be initialized to 24 bytes of X'00' |

Encryption of a plaintext string can be broken into segments; see “Cipher block chaining” on page 81.

# Chapter 4. Native X.25 communication server

The native X.25 server allows an application program running under LANDP for DOS or LANDP for OS/2 to communicate with another application program running in a different computer connected to the X.25 network. The workstation in the LANDP workgroup with the native X.25 server installed must be physically attached to the X.25 network. It is called the *gateway*.

The gateway can be a DOS workstation using an X.25 communication server and the appropriate X.25 software support program, or an OS/2 workstation using Communications Server for OS/2 Warp X.25 API support.

*Table 7. Function Codes used with the Native X.25 Communication Server*

| Function code | Description | Env. | Page |
|---|---|---|---|
| CL | Close | 02-- | 95 |
| CN | Connect | 02-- | 95 |
| DC | Define connection | 02-- | 96 |
| GS | Get status | 02-- | 98 |
| OP | Open | 02-- | 99 |
| QC | Query connection | 02-- | 100 |
| RH | Read host | 02-- | 101 |
| RL | Release | 02-- | 103 |
| SH | Send host | 02-- | 104 |

Function codes recognized by the X.25 communication server operating in either the LANDP for DOS or OS/2 environments. The last column refers to the page where you can find the function described.

## Server characteristics

The native X.25 server provides the workstation application programs with access to an X.25 virtual circuit. No high-level protocol functions are provided.

*Proper use of all high-level protocols used in the session are the responsibility of the application program.*

The native X.25 server can be used for communication between any workstation in the LANDP workgroup and any other computer that supports X.25 protocol.

The native X.25 server allows the session ID in the two application ends (the communicating parties) to be a primary or a secondary, but not both at the same time. With the native X.25 server the following terminology is used:

- The *caller* uses the X.25 network to call services from a host or another workstation.
- The *called* application program must be prepared to be called. It can accept or refuse the call.

© Copyright IBM Corp. 1992, 2002

An application program running under OS/2 and LANDP for OS/2 must first request an OP function to start using the native X.25 communication server. To stop using the native X.25 communication server, a CL function must be requested. For an application program running under DOS, these function requests are only recommended.

When the connection is established, the caller and the called application programs can work in exactly the same way.

## Caller session initiation

The application programs working with the native X.25 server establish a connection following these procedures:

1. The caller application program requests an OP function. This is used for identification purposes when the application program is running under OS/2 and LANDP for OS/2. If the application is running under DOS and the gateway is a LANDP for DOS workstation, this is not mandatory.
2. The caller application program requests a connect function, CN, to the native X.25 server.
3. The server then contacts the X.25 network with a call packet containing:
   - Called subscriber number
   - Destination identification
   - User data information and facilities defined at customization time

   This information must either have been defined during customization or made available to the server with a DC function request. The destination identification is used in the remote LAN to locate the workstation and the application program to be requested.
4. The normal response to the connect function is a zero return code. This means that the syntax of the function is correct and no problems exist with the adapter.
5. When the connection has been made, a message is queued for the caller application program. You may receive this message with a receive host (RH) function. Flag1 shows whether the connection was successful. See “Read host (RH function)” on page 101 for the information returned.

## Called session initiation

When the X.25 data link control receives a call, it examines the request for the destination identification. This name should have been defined either during customization or in a DC function request. The caller must ensure that the user data field of the call packet has a destination identification that matches the destination identification customized for the destination workstation.

An application program requests an OP function at the beginning to provide its identification to the native X.25 server.

- When a subscriber identification and destination identification match is found, the following process takes place:
  1. The server notifies the called application program by queueing a message for the application program in the computer in the LANDP workgroup associated with the local session.

2. The called application program reads the message using an RH function. Flag1 is set to '{', and *Reply DATA* contains information about the session, see “Read host (RH function)” on page 101.
3. The native X.25 server accepts the call and the session is established. If the called application program wants to refuse the call, it must issue an RL function.
4. The called application program must issue a CN function to be able to send data.

- When only the destination identification is found and the subscriber identification associated with that configuration is left blank at customization:
  1. The server notifies the called application program by queueing a message for the application program in the computer in the LANDP workgroup associated with the local session.
  2. The called application program should read the message using an RH function. Flag1 is set to '{', and *Reply DATA* contains information about the session, see “Read host (RH function)” on page 101.
  3. If the called application program accepts the call, then it requests a CN function. If the application refuses the call, then it requests an RL function.

## Operation

The application program in the computer in the LANDP workgroup must use an end-to-end protocol that guarantees data integrity and avoids using all available data link buffers.

When an application program in the computer in the LANDP workgroup wants to terminate the session it requests an RL function.

## Using the native X.25 server

This section provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields. If you need more information about the CPRB fields, see Appendix A, “Connectivity programming request block” on page 669.

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | 26 | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 26 | 2 | 26 | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | **X25NAT##** | Server name |
| **Note:** The value of ## is defined during customization and relates to an X.25 virtual circuit. See the *LANDP Installation and Customization* book for more information. | | | |

The following fields are variable and are discussed in each function request description:
- Function code
- Request DATA length
- Reply DATA length

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 44 | 2 | | Replied PARMLIST length |
| 46 | 2 | | Replied DATA length |

If the request was successful, the *router return code* and the *server return code* are both X'00000000'. In all other cases, see the appropriate section in the *LANDP Problem Determination* book to see if you should take any action. The return values in Reply PARMLIST and DATA should be ignored if there is an error.

The following fields are variable and are discussed in each function request description:
- Replied DATA length
- Replied PARMLIST length

**PARMLIST:** The Request PARMLIST and Reply PARMLIST fields for the native X.25 server are:

| Offset | Length | Content |
|---|---|---|
| 0 | 1 | Flag1 |
| 1 | 1 | Flag2 |

**DATA:** Request DATA and Reply DATA contain data to be sent to or received from the server. The use of these areas is explained in the description of each function call.

**Note:** The application program must have enough space in Reply DATA to receive the record. Otherwise the message is truncated.

It is recommended that you allocate Reply DATA to the maximum size of 4096 bytes.

## Request reference

This section describes the functions that client programs can request from the native X.25 server. They are listed in alphabetical order of function code.

## Close (CL function)

This function finishes the X.25 communication. Use it as the last call from your application program.

| CPRB Field | Content/Description |
|---|---|
| Function code | CL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

On return from the CL function, LANDP has updated the return code and workstation ID fields.

### Common reasons for nonzero return codes

- An invalid session was specified
- The length was not set to zero
- The parameter length was smaller than 26

## Connect (CN function)

The workstation application program requests a CN function call to establish the use of an X.25 virtual circuit.

A caller application program uses CN to initiate the call, while a called application program uses CN to accept a call.

| CPRB Field | Content/Description |
|---|---|
| Function code | CN |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

On return from the CN function call, LANDP has updated the return code and workstation ID fields.

After a successful CN function call, the application program receives, asynchronously, a message with flag1 = '{'. The information received in the call confirmation packet is stored in the Reply DATA area of a subsequent RH request. The contents of this information depends on the facilities subscribed from the network.

If the CN function call was not successful, flag1 is set to '}'. The cause and diagnostics of this message are stored in the Reply DATA area of a subsequent RH request.

### Common reasons for nonzero return codes

- The virtual circuit has already been established.
- The modem is powered off or the link is inoperative.
- There are no input/output buffers available.
- Invalid session specified.
- The length is not set to zero.
- No circuits are available on the public network.

## Define connection (DC function)

Initial values of connection characteristics for the Native X.25 server are defined during customization. The application program may use a DC function call to change these characteristics.

This function may only be used before any connection is established for this local session and its associated virtual circuit.

| CPRB Field | Content/Description |
|---|---|
| Function code | DC |
| Request DATA length | 102 or 106 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |

| CPRB Field | Content/Description |
|---|---|
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1<br>**'U'** User Data Field format |

The value of flag1 on request determines the data structure used.

| Request DATA Values when flag1 is not equal to 'U' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 14 | Called subscriber identification (left-justified. filled with X'FF') |
| 14 | 1 | Connection type:<br>**'I'** Incoming virtual circuit<br>**'O'** Outgoing virtual circuit |
| 15 | 64 | X.25 facilities used as defined by your network provider (PTT). Byte 16 is the used length of this field, in binary. The contents of this field override the values defined during customization. |
| 79 | 2 | Used length of the following two fields plus four. Maximum is 12+4=16. |
| 82 | 8 | The destination identification used in outgoing calls that allows the other end to recognize the originator |
| 89 | 4 | Up to four bytes of application program data for the calling packet |
| 93 | 8 | Destination identification for incoming calls |

When flag1 is equal to 'U', the DC function modifies the 16 bytes of the User Data Field. This capability is required when users wish to establish connection with computers other than LANDP Native.

| Request DATA Values when flag1 = 'U' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 14 | Called subscriber identification (left-justified, filled with X'FF') |
| 14 | 1 | Connection type:<br>**'I'** Incoming virtual circuit<br>**'O'** Outgoing virtual circuit |
| 15 | 64 | X.25 facilities used as defined by your network provider (PTT). Byte 16 is the used length of this field, in binary. The contents of this field override the values defined during customization. The first byte of this field contains the length. |
| 79 | 2 | User Data Field length. Maximum is 16 = X'1000', byte-reversed |
| 81 | 16 | User Data Field |

| Request DATA Values when flag1 = 'U' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 97 | 8 | Destination identification for incoming calls |

**Notes:**

1. Information in bytes 0 through 79 can be obtained from the person responsible for ordering your X.25 connection or from the network provider (PTT).
2. All fields are positional and unused fields must be filled with X'FF'.
3. Data supplied in the DC function call changes the data created during customization (file VCM.CFG) for the local session.

### Example

The following example illustrates the content of Request DATA when the DC function is used:

- The number to be called: 231020108
  Bytes 0-14, desired number in ASCII padded with blanks:
  X'32333130323031303820202020202020'
- Connection type—outgoing.
  Byte 15, ASCII character 'O': X'4F'
- Facilities field, length three bytes, and requesting negotiable window size (X'43') with sending and receiving window size of seven.
  Bytes 16-79, padded with X'FF': X'03430707FFFF...'
- User data field length plus 4. The physical unit name, and application program data field are not used. The length is four, in personal computer integer format, followed by the fields not used filled with X'FF'.
  Bytes 80-81: X'0400'
  Bytes 82-89: X'FF...'
  Bytes 90-93: X'FF...'
- Calling physical unit ID is not used.
  Bytes 94-101: X'FF...'

### Common reasons for nonzero return codes

- The circuit has already been established.
- The Request DATA length is not equal to X'0066' (or X'006A' if flag1 = 'U').
- There are no input/output buffers available.
- Invalid session specified.

## Get status (GS function)

This function call returns the communication status and shows available pending messages to the application program for a specific session identifier.

| CPRB Field | Content/Description |
|---|---|
| Function code | GS |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |

| CPRB Field | Content/Description |
|---|---|
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 2 or 26 |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1. The communication status. Possible values are:<br>**'0'** Not connected<br>**'{'** Connection established |
| 1 | 1 | Flag2. Shows if any messages are pending<br>**'0'** No message is pending<br>**'1'** Message pending |

The Request and Reply DATA lengths must be set to zero.

### Common reasons for nonzero return codes

- Invalid session specified
- Data length are not zero

## Open (OP function)

Use this function as the first call, to identify the application program to the native X.25 server.

| CPRB Field | Content/Description |
|---|---|
| Function code | OP |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

On return from the OP function, LANDP has updated the return code and workstation ID fields.

### Common reasons for nonzero return codes

- An invalid session identification was specified
- The length was not set to zero
- The parameter length was smaller than 26

## Query connection (QC function)

The workstation application program can issue a QC function call to determine the current characteristics of the virtual circuit. This function can be used at any time. The data obtained could be saved to restore the original connection characteristics if a DC function call is used to define new characteristics.

| CPRB Field | Content/Description |
|---|---|
| Function code | QC |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 102 or 106 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 102 or 106 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 'U' User Data Field format |

On return, Reply DATA contains 102 (or 106) bytes of information. The value of flag1 on reply determines the data structure used.

| Reply DATA Values when flag1 is not equal to 'U' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 14 | Subscriber address (left-justified, padded with blanks) |
| 14 | 1 | Connection type:<br>**'I'** Incoming virtual circuit<br>**'O'** Outgoing virtual circuit |
| 15 | 64 | X.25 facilities used as defined by your network provider (PTT). Byte 16 is the used length of this field, in binary. The contents of this field override the values defined during customization |
| 79 | 2 | Used length of the following two fields plus four Maximum is 12+4=16 |
| 81 | 8 | The destination identification used in outgoing calls that allows the other end to recognize the originator |
| 89 | 4 | Up to four bytes of application program data for the calling packet |
| 94 | 8 | Destination identification for incoming calls |

When flag1 is equal to 'U', the QC function obtains the 16 bytes of the User Data Field.

| Reply DATA Values when flag1 = 'U' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 14 | Subscriber address (left-justified, padded with blanks) |
| 14 | 1 | Connection type:<br>**'I'** Incoming virtual circuit<br>**'O'** Outgoing virtual circuit |
| 15 | 64 | X.25 facilities used as defined by your network provider (PTT). Byte 16 is the used length of this field. The contents of this field override the values defined during customization |
| 79 | 2 | User Data Field length. Maximum is 16 = X'1000', byte-reversed |
| 81 | 16 | User Data Field |
| 97 | 8 | Destination identification for incoming calls |

### Common reasons for nonzero return codes

- The Reply DATA length is not equal to X'0066' (or X'006A' if flag1 = 'U').
- There are no input/output buffers available.
- Invalid session specified.

## Read host (RH function)

This function receives a message, or obtains the number of pending messages, from the communication server.

Available messages were received by the data link control and queued in internal buffers.

| CPRB Field | Content/Description |
|---|---|
| Function code | RH |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | LANDP for DOS: 0 to 1024<br>LANDP for OS/2: 0 to 4096 |
| Reply PARMLIST length | 26 |
| Replied DATA length | LANDP for DOS: 0 to 1024<br>LANDP for OS/2: 0 to 4096 |
| Replied PARMLIST length | X'0000' or X'0001' |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1<br><br>Possible values are described below |

## Using the RH function

You can use the RH function call in two ways:

- If Reply DATA length is equal to zero when calling, then at the completion of the function call:
  - The Replied DATA length field contains the number of pending messages
  - Flag1 contains the communication status encoded in character format as follows:
    **'0'** No connection established.
    **'{'** Connection established.
    **'X'** Connection lost.

  The function GS call (see "Get status (GS function)" on page 98) provides the same information.
- If Reply DATA length is not equal to zero when calling, then at function completion:
  - The Replied DATA length field contains the actual length of the message returned in Reply DATA.
  - Flag1 contains the communication status encoded in character format as follows:
    **'{'.** Connection established.
    **'}'** Connection terminated.
    **'X'** Connection lost.
    **' '** None of the above.

**Note:** This mechanism only works for a DOS client calling a DOS server. In the other cases, use the GS function to obtain the current status. The GS function can also be used from a DOS client calling a DOS server.

For portability reasons, always define sufficient Reply DATA length with the RH function. If you need to obtain the current status then use GS function. For compatibility reasons the RH function with Reply DATA length = 0 is still maintained for DOS clients calling DOS servers.

When calling the RH function call with Reply DATA length not equal to 0, you must specify a value, at least, equal to the maximum size of the expected message reply.

### Special reply DATA information

- When an incoming call packet is received, the message is stored in Reply DATA. If the received message is a *connection call*, Reply DATA contains 92 bytes with the following information:

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 14 | Calling subscriber's identification |
| 14 | 1 | Reserved |
| 15 | 64 | X.25 facilities included in the call packet. Byte 16 is the binary length of the significant facilities field |
| 79 | 8 | Destination identification |
| 87 | 4 | Up to four bytes of application program data in the call packet |

- When clear is received because of a call packet send failure or an RL received from the other end, Flag1 is either '}' or 'X' and the Reply DATA length is X'0002'. This means that a lost connection condition has occurred. Here, the two bytes in Reply DATA contain the cause and diagnostic codes, respectively. The cause byte has the following meanings:

  **X'FF'** Diagnostic code set by X.25 adapter. Consult the *X.25 Communications Support User's Guide*.

  **X'00'** Diagnostic code set by the other end.

  **X'xx'** Diagnostic code set by the network or the X.25 Co-processor.

### Common reasons for nonzero return codes

- The session has not been established.
- The modem is powered off or the link is inoperative.
- Invalid session specified.
- The length of the data is greater than 1024 bytes for LANDP for DOS, or greater than 4096 bytes for LANDP for OS/2.
- There are no messages in the input queue.

## Release (RL function)

The workstation application program requests an RL function call to terminate use of an X.25 virtual circuit.

A called application program uses RL to refuse a call. Both a caller and called application program use RL to relinquish a virtual circuit.

| CPRB Field | Content/Description |
|---|---|
| Function code | RL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |

| CPRB Field | Content/Description |
|---|---|
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

### Common reasons for nonzero return codes

- The circuit has not yet been established.
- There are no input/output buffers available.
- Invalid session specified.
- The modem is powered off or the link is inoperative.

## Send host (SH function)

This function call sends a message from the workstation application program to the native X.25 server.

The server queues the message for transmission.

| CPRB Field | Content/Description |
|---|---|
| Function code | SH |
| Request DATA length | **LANDP for DOS:** 1 to 1024<br>**LANDP for OS/2:** 1 to 4096 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

### Request DATA values

Store the message to be sent in Request DATA. Data conversion from ASCII to EBCDIC might be necessary.

### Common reasons for nonzero return codes

- The session has not been established.
- The modem is powered off, or the link is inoperative.
- An invalid session has been specified.
- The length of the message for the server is greater than 1024 bytes for LANDP for DOS, and greater than 4096 bytes for LANDP for OS/2.
- There is a message in the native X.25 server input queue.
- There are no input/output buffers available.

# Chapter 5. Program-to-program communication server

The LANDP program-to-program communication (PPC) server, which runs on OS/2 and Windows, allows a LANDP application program to communicate with another application program through an SNA LU 6.2 session.

The LANDP TCP/IP wide area communications server, which runs on DOS, OS/2, Windows, and Linux, allows LANDP application programs to use the same set of functions for TCP/IP, or SNA over TCP/IP, communications.

The server name is *always* PPC, and the LANDP PPC session is defined in the Request PARMLIST area with the *conversation handle*.

*Table 8. Function Codes used in the Program-to-program Communication Server*

| Function code | Description | Env. | Page |
|---|---|---|---|
| CL | Close conversation | 02NL | 111 |
| GA | Get attributes | 02NL | 112 |
| GS | Get status | 02NL | 113 |
| OP | Open conversation | 02NL | 113 |
| RD | Receive data | 02NL | 115 |
| SD | Send data | 02NL | 116 |

The first and second columns give the function code and the name of the function. The third column shows the operating environment of the function. "02NL" means that it is available from LANDP for DOS, LANDP for OS/2, LANDP for Windows, or LANDP for Linux servers. The last column refers to the page where you can find the function described.

## PPC conversation allocation

The PPC server is asymmetric—either end could be the caller (contention winner) or the called (contention loser) side of the conversation, but not both at the same time.

When the conversation is established, the contention winner and loser work in exactly the same way, being able to send and receive data on a half-duplex flip-flop basis.

## Contention-winner application programs

The contention-winner application program requests an OP function with flag1 = 'S'. (The flags are described in "PPC server flag description" on page 110.) This causes the PPC server to send an indication to the partner application program to allocate a conversation.

The recommended connection procedure is as follows:

- The application program requests an OP function with flag1 = 'S'.

© Copyright IBM Corp. 1992, 2002

If the return code is *not* zero, there could be communications problems (permanent situation): Check the application program logic, SNA provider status, connection status, attachment status, and equipment.

If the communication is correct, there are no sessions free to be used (temporary situation): The application program should time out and try again. The timeout is important to avoid overloading the server with excessive requests. Modifying session limits in the SNA subsystem can solve problems of this kind.

If the return code is zero, the conversation is allocated (active) and the application is in SEND state, and is able to send data.

### Contention-loser application programs

The contention-loser application program requests the OP function with flag1 = 'R'. This causes the PPC server to receive and accept the incoming allocate indication from the partner application program. The recommended connection procedure is as follows:

- The application program requests an OP function with flag1 = 'R'.

  If the return code is *not* zero, there could be communications problems (permanent situation): Check the application program logic, SNA provider status, connection status, attachment status, and equipment.

  If the communication is correct, no incoming allocate indications are received (the partner application program did not issue OP function with flag1 = 'S'): The application program should time out and retry the function until it is successful. The timeout is important to avoid overloading the server with excessive requests.

  If the return code is zero, the conversation is allocated (active) and the application program is in RECEIVE state, and is able to receive data but not to send.

## Conversation state changes

The application program is in RECEIVE state, immediately after a successful OP function with flag1 = 'R' was issued. The application program is in SEND state immediately after a successful OP with flag1 = 'S' was issued.

### Application program in SEND state

The application program can issue multiple SD functions without carrying the SEND indicator (flag4 = ' '). Whenever it requests an SD function with flag4 = 'X', the SEND indicator is carried together with data, entering the application program in RECEIVE state and ready to receive data from the partner.

### Application program in RECEIVE state

First issue a local function WM to wait for data from the partner or communication status. No data can be sent in RECEIVE state.

When the WM function returns successfully, the application program must issue an RD function to receive the data.

The sequence WM—RD has to be repeated until RD returns flag4 = 'X', which means that the partner is sending an indicator that allows the application program to enter the SEND state.

The sending or receiving of flag2 = 'C' (cancel unit of data) does not change conversation state.

A positive response sent by the application program does not affect the conversation state, whereas a negative response keeps the local application in RECEIVE mode.

## PPC conversation deallocation

Before ending, you should close conversations used by application programs. Try to synchronize your partner programs to have the conversation deallocated by the application programs when they are in SEND state. If the application programs finish without closing the conversations, the server could abnormally end them and also the sessions.

## PPC responses management

Normally, when the local LU receives a message that requires CONFIRMATION, the PPC server sends it automatically to the partner when the application program reads this message.

The application program is responsible for these responses depending on every RD function it requests:

- RD function with flag5 = ' ' makes the server responsible for the response.
- RD function with flag5 = 'R' makes the application program responsible for the response. If data is successfully received, the application program must give a response by requesting an SD function with flag5 = '+' (CONFIRMED) or flag5 = '–' (SEND_ERROR) and Request DATA length equal to zero.

A nonzero return code is sent when the application program tries to send or receive data if the previous RD was issued with flag5 = 'R', so that the server is waiting for a response from the application program. The application program must then give the response. Also, a nonzero return code is sent when the application program tries to send a response not related to a previous RD with flag5 = 'R' (response not necessary).

The PPC server can confirm data when the partner (for example, a transaction program running on the host using an SNA LU 6.2 interface) requests confirmation.

## Server exit for user data processing

You can write a user server for data pre- and post-processing. This server must be named EXFS and must accept at least two functions:

**PR (Process Read Data)** The PPC server requests a PR function from the EXFS server just before passing the data received from the line to the application program during the receive data process.

**PS (Process Send Data)** The PPC server requests a PS function from the EXFS server just before sending data from the application program to the line during the send data process.

The Request DATA and Reply DATA areas are the same, so that the server can copy Request DATA into its own internal buffer before copying the Reply DATA. The Replied DATA length can be different from the requested one but up to 4096 bytes.

(See also Chapter 30, “LANDP 3287 printer emulator API” on page 657 for another use of the EXFS server.)

## Conversation types

The advised and fully supported conversation type used in the server is the MAPPED conversation. The conversation type of a transaction program should be configured as MAPPED.

A BASIC conversation cannot be allocated (OP with flag1 = ’S’), but an incoming allocate can be accepted (OP with flag1 = ’R’) if the transaction program is configured as BASIC. Here the application program is completely responsible for matching the Request DATA length, and for setting the 2-byte length header when sending data to the partner. Otherwise it may get a P7 return code (logical record started but not finished).

BASIC conversations are not recommended unless very special reasons require it.

## Using the PPC server

This section provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields. If you need more information about the CPRB fields, see Appendix A, “Connectivity programming request block” on page 669.

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | 26 | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 26 | 2 | 26 | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 96 | 8 | PPC | Server name. (The convention used in previous versions is also supported.) |

The following fields are variable and are discussed in each function request description:
- Function code
- Request DATA length
- Reply DATA length

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 44 | 2 | | Replied PARMLIST length |
| 46 | 2 | | Replied DATA length |

If the request was successful, the *router return code* and the *server return code* are both X'00000000'. In all other cases, see the appropriate section in the *LANDP Problem Determination* book to see if you should take any action. The return values in Reply PARMLIST and Reply DATA should be ignored if there is an error.

The following fields are variable and are discussed in each function request description:
- Replied PARMLIST length
- Replied DATA length

**PARMLIST:** The Request PARMLIST and Reply PARMLIST fields for the PPC server are:

| Offset | Length | Content |
|---|---|---|
| 0 | 1 | Flag1 |
| 1 | 1 | Flag2 |
| 3 | 1 | Flag4 |
| 4 | 1 | Flag5 |
| 5 | 2 | Conversation Handle |
| 6 | 1 | Flag7 |
| 7 | 1 | Flag8 |
| 8 | 1 | Flag9 |
| 17 | 4 | (Reply PARMLIST only) Contains sense data after an active conversation is lost (Flag1 = '>') |
| **Note:** Flag7 overlays part of the conversation handle. There is no conflict because the handle is used on request and the flag is used on reply. | | |

**DATA:** Request DATA and Reply DATA contain data to be sent to or received from the server to communicate with the partner LU. The use of these areas is explained in the description of each function request.

## PPC server flag description

These flags were chosen to provide uniformity with the traditional LU 0,1,2,3 SNA Server flags.

Flag1

**'S'** Set by the application program in the OP function, this requests allocation of a conversation, using a session where the local LU is the contention winner (the one that can send data immediately after session setup—SEND state). The protocol for exchanging data allows both LUs to send and receive data in an orderly way.

**'R'** Set by the application program in the OP function, this requests allocation of a conversation, using a session where the local LU is the contention loser (the one that has to wait for data immediately after session setup—RECEIVE state). The protocol for exchanging data allows both LUs to send and receive data in an orderly way.

**'>'** Received at any time, this informs the transaction program that the previously active conversation is lost, because the local or remote transaction program released it or because of communications problems. Bytes 17—20 of Reply PARMLIST contain SNA sense data, if available. If no SNA sense data is available they contain X'00000000' if the conversation terminated normally, or X'FFFFFFFF' otherwise. SNA sense data is described with other function management headers in the *System Network Architecture (SNA) Formats* book.

**'<'** Received during the RD function, this informs the transaction program that the conversation has been established. This flag is returned only if the OP function was issued with flag8 = 'A', requesting asynchronous notification.

Flag2

**'C'** Set by the transaction program in an SD function, this tells the server to cancel the stream of data currently being sent and free the LU to send a new one (SEND state). It can also be received by a transaction program in an RD function when the remote transaction program cancels the data currently being sent.

Flag4

**'X'** 'X' shows a change of direction of the conversation.

Set by the application program in an SD function, and received at completion of an RD function, this informs the server that the data carries ('X') or does not carry (' ') the SEND indicator. It sets the conversation state to SEND or RECEIVE mode. The data received is the group of incoming data messages

during RECEIVE mode, before the receiving of this SEND indicator. The data sent is the group of outgoing data messages during SEND state before sending the SEND indicator.

Flag5

**'R'** Set by the transaction program in an RD function, this tells the server that the confirmation related to the data to be received will be managed and issued by the application program. Otherwise the PPC server manages it.

**'+' or '–'**
Set by the transaction program in an SD function, this tells the server to send the previously pending response, either positive or negative. It can also be received by a transaction program in an SD function when confirmation was requested.

Conversation Handle

**nn** Set by the transaction program in all functions. These two ASCII numeric characters identify the conversation to which the current function is related. The conversation handle must always be the same for a specific conversation. Different conversations must use *different* conversation handles. If you use the server name convention supported in previous versions, the conversation handle is not required.

Flag7

**'+'** Received by the transaction program in an RD function call, this means that an incomplete data record has been received because the Reply DATA length is too small or due to a pacing contention.

Flag8

**'A'** Set by an application in an OP function call, this asks for asynchronous notification of the establishment of the conversation. Control is returned immediately to the application. A zero return code means that the server has accepted the request to process it.

Flag9

**'D'** Set by an application in an SD function call, this asks for confirmation that the data has been sent. The SD function is blocked until the positive or negative confirmation is received, or until the LAN timeout expires.

## Request reference

This section describes the functions that clients can request from the PPC server. They are listed in alphabetical order of function code.

## Close conversation (CL function)

This function closes the conversation between the local and partner application program by terminating it in an orderly way. This function releases the use of the PPC server

and all resources, control blocks, and tables related to the conversation. After this function is issued, a new OP must be issued to start a new conversation.

| CPRB Field | Content/Description |
|---|---|
| Function code | CL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 5 | 2 | Conversation Handle |

## Get attributes (GA function)

This function returns information in the Reply DATA area of parameters related to the current conversation.

| CPRB Field | Content/Description |
|---|---|
| Function code | GA |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 100 to 4096 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 91 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 5 | 2 | Conversation Handle |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 64 | Tp_name (EBCDIC). Default is 'FBSSTPNA' |
| 65 | 8 | Partner_LU_alias (ASCII). Default is 'FBSSLUPA' |
| 74 | 8 | Mode_name (EBCDIC). Default is 'FBSSMODE' |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 83 | 8 | LU_alias (ASCII). Default is 'FBSSLULO' |
| **Note:** The bytes not used (for example, offset 64 length 1) are not shown | | |

## Get status (GS function)

This function returns the communication status of the current conversation, and shows if any messages are pending. The Request and Reply DATA lengths must be set to zero.

| CPRB Field | Content/Description |
|---|---|
| Function code | GS |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 5 | 2 | Conversation Handle |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 (communication status). Possible values are:<br>' ' No conversation established<br>**'S'** Conversation in SEND state<br>**'R'** Conversation in RECEIVE state |
| 1 | 1 | Flag2 shows whether any messages are pending:<br>**'0'** No message is pending<br>**'1'** Message pending |

## Open conversation (OP function)

This function identifies and initializes the transaction program to use the PPC server. This must be the first function used. OP also establishes the conversation between the local and partner LUs.

Optionally, the transaction program can provide the tp_name, the partner_LU_alias it wants to be in conversation with, the mode_name, and local LU_alias (parameters defined in Communications Server or Microsoft SNA server configuration file) in Request DATA. If it does not, the PPC server uses some internal default values for these fields when filled with ASCII blanks. The application program can also supply a

user_id and a password if needed, which is sent to the partner. The parameters *tp_name*, *mode_name*, *user_id* and *password* are EBCDIC characters.

The transaction program must also specify if it wants to use a contention-winner session (flag1 = 'S', initial state after session establishment is SEND), or a contention-loser session (flag1 = 'R', initial state after session establishment is RECEIVE). The PPC server tries to provide the kind of session requested if there is one available.

However, it is very important that both sides of the conversation (both LUs) agree in establishing the session, one as a contention winner and the other as a contention loser, so that there are no conflicts or deadlocks.

The PPC server maps LANDP conversations into APPC conversations. These conversations are a means of serially sharing the SNA LU 6.2 sessions. They are mapped into any established LU 6.2 session between the local and partner LU that uses the specified mode.

The conversation handle can be supplied by the application program, or can be left blank. If it is left blank, a unique conversation handle is returned in the reply parameters.

| **CPRB Field** | **Content/Description** |
|---|---|
| Function code | OP |
| Request DATA length | 120 to 4096 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| **Request PARMLIST Values** | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1<br>**'S'** Contention-winner session with the initial state as SEND<br>**'R'** Contention-loser session with the initial state as RECEIVE |
| 5 | 2 | Conversation Handle or 'bb' |
| 7 | 1 | Flag8<br>**'A'** Control immediately returned to the application: request is processed asynchronously by the PPC server<br>**' '** Call blocked until request is completed |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 64 | Tp_name (EBCDIC): default is 'FBSSTPNA' |
| 65 | 8 | Partner LU_alias (ASCII): default is 'FBSSLUPA' |
| 74 | 8 | Mode_name (EBCDIC): default is 'FBSSMODE'<br><br>No entry should be made, for OP with flag1 = 'R'. If you make an entry here, the system ignores it. |
| 83 | 8 | LU_alias (ASCII): default is 'FBSSLULO' |
| 92 | 10 | User_ID (EBCDIC) |
| 103 | 10 | Password (EBCDIC) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 5 | 2 | Conversation Handle |

## Receive data (RD function)

With this function, the LANDP application program receives the communications status flags (flag1) or the data coming from the partner LU.

| CPRB Field | Content/Description |
|---|---|
| Function code | RD |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | Maximum length of expected record (up to 56 Kbytes) |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of record actually read |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 4 | 1 | Flag5<br>**' '** The server has control of the response<br>**'R'** The application program controls the response and must issue SD with the settings flag5 = '+' or '–' and Request DATA length zero, before continuing to send or receive data |
| 5 | 2 | Conversation Handle |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1<br>**' '** Normal data, see Flag4<br>**'>'** Broken conversation (conversation lost)<br>No data is returned, but bytes 17 thru 20 of Reply PARMLIST are set. This flag is received when the conversation is normally deallocated by the partner or when an abnormal deallocation takes place. It is recommended that, where necessary, an OP function is issued to allocate the conversation again<br>**'<'** Conversation is established (when OP request specified flag8 = 'A') |
| 1 | 1 | Flag2<br>**' '** Ignore<br>**'C'** Current data not valid<br>The local transaction program must act appropriately related to this cancellation. The meaning of this cancellation depends on the application. The application program remains in the RECEIVE state and is ready to issue new RD functions |
| 3 | 1 | Flag4<br>**' '** Data without SEND indicator.<br>**'X'** Data with SEND indicator. The application program is in SEND state and is ready to send data to the partner |
| 4 | 1 | Flag5<br>**'+'** Positive response set by a remote transaction<br>**'–'** Negative response set by a remote transaction |
| 6 | 1 | Flag7<br>**'+'** Data record incomplete. The application should issue more RD functions to receive the rest of the record |
| 17 | 4 | If Flag1 = '>' this field contains SNA sense data if it is available. If no SNA sense data is available it contains X'00000000' if the conversation terminated normally, or X'FFFFFFFF' otherwise. SNA sense data is described with other function management headers in the *System Network Architecture (SNA) Formats* book. |

### Reply DATA values

The data is received from the remote application program if flag1 is set to ' '.

## Send data (SD function)

To send data, the LANDP application program must be in SEND state and request a correct function from the PPC server.

| CPRB Field | Content/Description |
|---|---|
| Function code | SD |
| Request DATA length | 0 if the application program is managing responses (see Flag5 below), or 1 to 57344 if data is being sent |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request and Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 1 | 1 | Flag2<br>**' '** Normal data<br>**'C'** Cancels the current block of data by sending an error indication to the remote LU and keeping the local LU in SEND state |
| 3 | 1 | Flag4<br>**' '** Causes the PPC server to send the data to the remote LU but without sending the SEND indicator, so that the local LU is able to go on transmitting<br>**'X'** Causes the PPC server to send the SEND indicator to the remote LU, entering the local LU in RECEIVE state |
| 4 | 1 | Flag5<br>**'+'** See 'R' setting of flag5 in RD function request, together with the paragraph below<br>**'–'** See 'R' setting of flag5 in RD function request, together with the paragraph below<br><br>An application program can manage responses by setting flag5 = '+' (CONFIRMATION) or '–' (negative response) in the SD function, and setting Request DATA length equal to zero. This is mandatory before sending or receiving anything else, if the previous RD function was issued with flag5 = 'R'. A return code is set if a response is pending and not given, or if the application program tries to send a response not requested<br><br>Flag5 on reply<br>**'+'** Data record confirmed (positive confirmation)<br>**'–'** Negative confirmation or timeout |
| 5 | 2 | Conversation Handle on request |
| 8 | 1 | Flag9 on request<br>**'D'** Requests confirmation of the data record. The function is blocked until a positive or negative confirmation is received, or an error indication is received, or until the LAN timeout expires |

### Request DATA values

The data to be sent to remote application program.

# Chapter 6. MQSeries Link server




This chapter describes the MQSeries Link server, which allows workstations in the LANDP workgroup to add and retrieve messages from an MQSeries Queue Manager being run by MQSeries for OS/2 or by MQSeries for Windows NT.

The MQSeries Link server enables LANDP Applications to access a subset of the MQSeries Message Queuing Interface (MQI). This extends the benefits of message queuing to all workstations in a LANDP workgroup without the need for a MQSeries client to be installed in each workstation. The function in MQSeries link server communicates directly with the MQSeries Queue Manager.

Transactional functions are provided so that a group of message changes can be committed or rolled back together. This functionality has the restriction that each transactional unit of work must be contained within a single session.

Additional sessions are requested in the LANDP application by use of the server name field in the CPRB. This is processed in the same way as other multiple service servers such as the SNA communication server.

The transactional capabilities of the LANDP MQSeries Link server do not extend beyond it's local MQSeries Queue Manager. There is no transactional synchronisation with other LANDP servers. Transactional support is 1 phase commit.

Messages being sent or retrieved by the MQSeries Link server are limited in size by the requirements of the CPRB interfaces. For both LANDP for OS/2 and Windows, the maximum message length is 57000 bytes. Segmented messages are not supported by MQSeries Link server.

Data conversion of messages is enabled in the LANDP API with the GQ option EHCGQ_CONVERT. When requested, conversion is attempted by the MQSeries Queue Manager. If further data conversion is required between the LANDP client and the MQSeries Link server, the LANDP application must manage this conversion.

The use of the NoWait option with RMTREQ (Remote Request) is restricted. The requests for a session must be handled one at a time. After issuing RMTREQ with NoWait, GETRPLY must be issued before the next RMTREQ is issued. Failure to do this generates the SB (Session Busy) return code.

Sample LANDP application code and associated files are provided to aid application development with this server. These samples can be found as follows:

For Windows in `\EHC\EHCN600\SAMPLE`
For OS/2 in `\EHC\EHCO600\SAMPLE`

© Copyright IBM Corp. 1992, 2002

*Table 9. Function codes used in the LANDP MQSeries Link server.*

| Function code | Description | Env. | Page |
|---|---|---|---|
| BT | Begin Transaction | -2N- | 123 |
| CP | Checkpoint | -2N- | 124 |
| CS | Close Session | -2N- | 124 |
| CQ | Close Queue | -2N- | 125 |
| ET | End Transaction | -2N- | 126 |
| GQ | MQGET | -2N- | 127 |
| PQ | MQPUT | -2N- | 131 |
| P1 | MQPUT1 | -2N- | 134 |
| RB | Rollback | -2N- | 135 |

The first and second columns give the function code and the name of the function. The third column shows the operating environment of the function as explained in “Operating environments” on page xxvi. For example -2N- means that it is available from LANDP for OS/2 and LANDP for Windows servers. The last column gives the page where you can find the function described.

## Using the MQSeries Link server

This section provides guidelines to help you supply the necessary information in the request CPRB fields and understand the information you receive in the Reply CPRB fields. For more information about the CPRB fields, see Appendix A, “Connectivity programming request block” on page 669 and the first chapter in *LANDP Programming Guide*.

*Table 10. CPRB fields on request*

| Offset | Length | Value | Content |
|---|---|---|---|
| 10 | 2 | | Function code |
| 14 | 2 | Whole number | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | Whole number | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 26 | 2 | Whole number | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | Whole number | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |
| 96 | 4 | EHQMQ## | Server name |
| **Note:** : The value of ## can be defined during customization. The MQSeries Link server uses the value of ## as the application’s session ID. | | | |

The following fields are variable and are discussed in each function request description:

- Function code
- Request PARMLIST length
- Request DATA length
- Reply PARMLIST length
- Reply DATA length

The MQSeries Link server is a multiple service server. Within the constraints defined by workgroup customisation, RMTREQ calls that specify a different server name are processed by the same server executable. When an application defines a new server name in the CPRB, the MQSeries Link server creates a new session to process the request. Subsequent calls from the application that use the same server name are processed by the same session. If the server name is kept the same for all RMTREQ calls from an application, only one session is associated with that application.

**Notes:**

1. If you operate multiple sessions from an application, make sure that you always specify the appropriate server name when issuing a function.
2. A number of the PARMLIST fields are used for input and output. The application developer must ensure that these fields are set to spaces or null characters between function calls.

*Table 11. CPRB fields on reply*

| Offset | Length | Value | Content |
|---|---|---|---|
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 44 | 2 | | Replied PARMLIST length |
| 46 | 2 | | Replied DATA length |

If the request is successful, the router return code and the server return code are both X'00000000'. In all other cases, refer to the appropriate section in *LANDP Programming Reference*, to see if you should take any action. If there is an error, ignore the return values in Reply PARMLIST and DATA.

The following fields are variable and are discussed in each function request description:

- Replied PARMLIST length
- Replied DATA length

## Integration With MQSeries

The MQSeries Link server provides an interface to the MQSeries Queue Manager. To enable this some of the MQSeries behaviour has been predefined. To more fully understand the interaction with MQSeries, and enables development of MQSeries applications to work with the messages processed by this server the following should be noted:

**Naming MQSeries objects**

These rules apply:

- The special characters % and / are not permitted in the names of queue managers or queues.
- The LANDP customisation process restricts its input field for the queue manager name to 24 characters. All alphabetic characters in this list are folded to upper case.

  The LOADER statement does not impose any restriction on the queue manager name. If the customisation process is too restrictive for the required queue name, you can manually edit the AUTOFBSS file to change the LOADER statement's queue manager name.
- It is recommended that Queue manager names are defined in upper case characters.
- Queue names are restricted to a maximum of 24 characters (MQSeries allows 48).

**Creating queues**

Queues used by the MQSeries link server should be predefined. If temporary qyeues are used, any messages left on the queue are purged by MQSeries Queue Manager when CQ is issued by the session that first used the queue.

**Message Descriptor**

The Message Descriptor is as MQMD_DEFAULT and altered as follows:

| | |
|---|---|
| Format | = MQFMT_STRING |
| ReplyToQMgr | = Queue Manager defined by LANDP Customization |
| PutApplType | = MQAT_DEFAULT |
| UserIdentifier | = The user ID as passed with PQ, P1 or GQ |
| CodedCharSetId | = MQCCSI_Q_MGR |
| Encoding | = MQENC_NATIVE |

**Object Descriptor**

The Object Descriptor has been defined as MQOD_DEFAULT and then altered as follows :

| | |
|---|---|
| ObjectType | = MQOT_Q, that is, always a queue |
| ObjectName | = The queue name as passed with PQ, P1 or GQ |
| AlternateUserId | = The user ID as passed with PQ, P1 or GQ |

**Data conversion**

If requested by the application, data conversion is carried out by the MQSeries

Queue Manager. The LANDP MQSeries Link server does not provide any additional data conversion services when communicating with LANDP client machines.

**Error handling with multiple calls to the MQSeries Queue Manager**
Some of the MQSeries Link server functions require multiple calls to be issued to the MQSeries Queue Manager, for example Close Session. If an error is encountered when using the MQSeries API, the first error is used to define the MQSeries Link server's return code and the PARMLIST Reply value for the MQ Reason Code.

## Request reference

These are the functions supported by the LANDP MQSeries Link server. They are listed in alphabetic order of function code.

## Begin Transaction (BT)

The BT function begins a new transaction. A transaction consists of several related MQ requests that appear as a single change to the user.

**Notes:**

1. It is the responsibility of the application to coordinate its transaction between a LANDP MQSeries Link server and any other server that provides transaction facilities, such as a shared file server or another MQSeries server.
2. Transactional support is for 1-phase commit.
3. A transaction must be contained within a single session.
4. A transaction groups a number of changes into a single unit of work that can be cancelled or committed. You can cancel a transaction with the rollback (RB) function or commit it with either an end transaction (ET) or a checkpoint (CP) function.

If the client application or workstation is shutdown, or an EJ function is issued before the unit of work is complete, the changes made to the queue manager are ROLLED back.

A long-running application that issues GQ, PQ or P1 within a transaction but that never issues a CP, RB or ET will cause queues on the MQSeries Queue Manager to fill up with messages that are not available to other applications.

| CPRB field | Content and description |
|---|---|
| Function code | BT |
| Request PARMLIST length | 0 |
| Request DATA length | 0 |
| Reply PARMLIST length | 4 |
| Reply DATA length | 0 |
| Replied PARMLIST length | 4 |
| Replied DATA length | 0 |

*Table 12. Reply PARMLIST values*

| Offset | Length | Content |
|---|---|---|
| 0 | 4 | MQ reason Code |

**MQ Reason Code**
Contains the MQSeries error return code if an underlying MQSeries API call has failed. The LANDP server return code is also non-zero.

If the MQSeries API call is successful this field contains the MQSeries value for MQRC_NONE.

## Checkpoint (CP)

This function validates all operations in the current transaction and begins a new logical transaction. The effect is the same as requesting an end-transaction ET function and a begin-transaction BT function in a single request. For more details on transactional support, refer to “Begin Transaction (BT)” on page 123.

| CPRB field | Content and description |
|---|---|
| Function code | CP |
| Request PARMLIST length | 0 |
| Request DATA length | 0 |
| Reply PARMLIST length | 4 |
| Reply DATA length | 0 |
| Replied PARMLIST length | 4 |
| Replied DATA length | 0 |

*Table 13. Reply PARMLIST values*

| Offset | Length | Content |
|---|---|---|
| 0 | 4 | MQ reason Code |

**MQ Reason Code**
Contains the MQSeries error return code if an underlying MQSeries API call has failed. The LANDP server return code is also non-zero.

If the MQSeries API call is successful this field contains the MQSeries value for MQRC_NONE.

## Close session (CS)

This function closes a specified session. It can only be issued by the process that initiated the session. The session is specified in the CPRB server name field, in which the last two characters identify the session.

The MQSeries Link server is a multiple service server. Within the constraints defined by workgroup customisation, RMTREQ calls that give a different server name are processed by the same server executable. When an application defines a new server

name in the CPRB, the MQSeries Link server creates a new session to process the request. Subsequent calls from the application that use the same server name are processed by the same session.

Close Session implies ET in that an incomplete transaction is committed with the MQSeries Queue Manager. For more details on transactional support refer to the BT function.

Close Session implies CQ for all queues that are open for use by the specified session.

| CPRB field | Content and description |
|---|---|
| Function code | CS |
| Request PARMLIST length | 0 |
| Request DATA length | 0 |
| Reply PARMLIST length | 4 |
| Reply DATA length | 0 |
| Replied PARMLIST length | 4 |
| Replied DATA length | 0 |

*Table 14. Reply PARMLIST values*

| Offset | Length | Content |
|---|---|---|
| 0 | 4 | MQ reason Code |

**MQ Reason Code**
Contains the MQSeries error return code if an underlying MQSeries API call has failed. The LANDP server return code is also non-zero.

If the MQSeries API call is successful this field contains the MQSeries value for MQRC_NONE.

## Close queue (CQ)

This function closes a queue that has been opened as a result of a PQ or GQ function request. To ensure resources are released efficiently, it should be called as soon as work on a queue is finished. CQ is implied when the client application finishes with an EJ or the session is finished with a CS.

As a result of previous GQ and PQ function calls, the LANDP application should be aware of the queue handles that are open and can close them as necessary to release resources.

| CPRB field | Content and description |
|---|---|
| Function code | CQ |
| Request PARMLIST length | 2 |
| Request DATA length | 0 |
| Reply PARMLIST length | 4 |

| CPRB field | Content and description |
|---|---|
| Reply DATA length | 0 |
| Replied PARMLIST length | 4 |
| Replied DATA length | 0 |

*Table 15. Request PARMLIST values*

| Offset | Length | Content |
|---|---|---|
| 0 | 2 | Queue handle |

*Table 16. Reply PARMLIST values*

| Offset | Length | Content |
|---|---|---|
| 0 | 4 | MQ reason code |

**Queue Handle**
The queue handle is the identifier returned from a previous GQ or PQ function call, and identifies to the MQSeries Link server which queue is to be closed. The queue handle is processed as a hexadecimal value.

**MQ Reason Code**
Contains the MQSeries error return code if an underlying MQSeries API call has failed. The LANDP server return code is also non-zero.

If the MQSeries API call is successful this field contains the MQSeries value for MQRC_NONE.

## End Transaction (ET)

This function ends the current transaction and commits the changes to the queue manager.

If the client application or workstation is shut down, or an EJ function is issued before the ET function, the changes made to the queue manager are ROLLED back. If the CS function is issued before the logical unit of work is complete, the ET function is implied and the unit of work is committed.

Transactional support is for 1-phase commit. For more details on transactional support, refer to “Begin Transaction (BT)” on page 123.

| CPRB field | Content and description |
|---|---|
| Function code | ET |
| Request PARMLIST length | 0 |
| Request DATA length | 0 |
| Reply PARMLIST length | 4 |
| Reply DATA length | 0 |
| Replied PARMLIST length | 4 |

| CPRB field | Content and description |
|---|---|
| Replied DATA length | 0 |

*Table 17. Reply PARMLIST values*

| Offset | Length | Content |
|---|---|---|
| 0 | 4 | MQ reason Code |

**MQ Reason Code**
Contains the MQSeries error return code if an underlying MQSeries API call has failed. The LANDP server return code is also non-zero.

If the MQSeries API call is successful this field contains the MQSeries value for MQRC_NONE.

## MQGET (GQ function)

The GQ function issues a MQGET call to MQSeries, which retrieves a message from a queue defined on the queue manager. The MQSeries Link server ensures that the requested queue is open. It is valid for a session to use PQ and GQ on the same queue. When GQ is part of a transaction, the browse options are not permitted.

| CPRB field | Content and description |
|---|---|
| Function code | GQ |
| Request PARMLIST length | 92 |
| Request DATA length | 0 |
| Reply PARMLIST length | 116 |
| Reply DATA length | >= Expected length of message |
| Replied PARMLIST length | 116 |
| Replied DATA length | Length of resulting message |
| | |

*Table 18. Request PARMLIST values*

| Offset | Length | Content |
|---|---|---|
| 0 | 2 | Queue handle |
| 2 | 24 | Queue name |
| 26 | 24 | Message ID |
| 50 | 24 | Correlation ID |
| 74 | 12 | User Identifier |
| 86 | 4 | Wait Interval |
| 90 | 2 | LANDP GQ Message Options |

*Table 19. Reply PARMLIST values*

| Offset | Length | Content |
|---|---|---|
| 0 | 2 | Queue handle |
| 2 | 24 | Queue name |
| 26 | 24 | Message ID |
| 50 | 24 | Correlation ID |
| 74 | 12 | User identifier |
| 86 | 24 | Reply queue name |
| 110 | 2 | Message type |
| 112 | 4 | MQ Reason code |

*Table 20. Reply data values*

| Offset | Length | Content |
|---|---|---|
| 0 | | Resulting Message |

A number of the following fields are used for input and output. The LANDP application developer must ensure that these fields are correctly defined when repeating the MQSeries Link server function calls.

**Queue Handle**

The queue handle is an input/output field. Separate queue handles are issued for destructive and non-destructive GQ function calls. Non-destructive is defined by using one of the BROWSE options.

When multiple GQ calls are being issued, the returned queue handle should be reused. However, note that the queue handle is associated with the queue name, the input value of the user identifier, and whether or not the GQ function is destructive.

To indicate that a new handle is required, the field should be set to the value X'0000'. The queue handle is processed as a hexadecimal value.

**Queue Name**

This is the local name of the queue as defined on the MQSeries queue manager. The name must not contain leading or embedded blanks, but may contain trailing blanks. The first null character and all characters following it are treated as blanks. For more information about queue names, see the *MQSeries Application Programming Guide*.

**Message ID**

The message ID is an input/output field. The LANDP application can define the message ID, in which case it is used as part of the message selection criteria passed to MQGET. If this is not the desired action, set this field to null. The output value is the message ID as found in the matching message.

If both correlation ID and message ID are set to null, MQGET matches any message on the queue.

**Correlation ID**

The correlation ID is an input/output field. The LANDP application can define the correlation ID, in which case it is used as part of the message selection criteria passed to MQGET. If this is not the desired action, set this field to null. The output value is the Correlation ID as found in the matching message.

If both correlation ID and message ID are set to null, MQGET matches the next message on the queue.

**User Identifier**

When the User Identifier is defined for the first GQ request, the underlying MQOPEN call has the option MQOO_ALTERNATE_USER_AUTHORITY and the identifier specified in this field is used as the string value for the AlternateUserId. If the User Identifier field is blank for the first GQ request, the MQOO_ALTERNATE_USER_AUTHORITY option is not used.

When the input value for the user identifier is null or blank, the user ID is determined by the MQseries queue manager. The chosen value varies with the operating system environment. On OS/2 systems, the user identifier is 'OS2'. On Windows, the user identifier is the user name associated with the LANDP MQSeries Link Server process.

For a successful GQ call, the output value is the UserIdentifier from the message descriptor of the matching message.

**Wait Interval**

This is the period of time for which the MQGET waits for a suitable message to arrive on the queue. For this parameter to be processed, the option EHCGQ_WAIT must be defined.

The value of this field is interpreted as follows:

| | |
|---|---|
| **Positive** | A positive value is interpreted as a period of time expressed in tenths of a second. |
| **Zero** | A value of zero specifies the use of the supervisor's LAN timeout period as defined in the CPRB field `ehctimeout`. If `ehctimeout` is defined as X'0000' a default period of 30 seconds is used. |
| **Negative** | A negative value means that the wait period is unlimited. |

**LANDP GQ Message Options**

The LANDP GQ message options use Version 2 of the MQSeries get message options structure. The MQSeries Link server supports a subset of the MQSeries get message options. The supported options are defined as EHCGQ string values:

**EHCGQ_NONE x'0000'**

No options specified.

**EHCGQ_CONVERT x'0001'**

This option requests that the application data in the message should be converted in accordance with the values defined by the message descriptor fields, CodedCharSetId and Encoding. The conversion is done by the MQSeries queue manager and completed before data is made available to the MQSeries Link server. If further data conversion is required between the

MQSeries Link server and the LANDP client workstation, the LANDP application must manage the conversion.

**EHCGQ_FAIL_IF_QUIESCING x'0002'**
This option forces the MQGET call to fail if the MQSeries queue manager is in the quiescing state.

**EHCGQ_WAIT x'0004'**
For the wait interval to be recognized, this option must be defined. Specifying this option ensures that MQGET waits for a message.

**EHCGQ_ACCEPT_TRUNCATED_MSG x '0008'**
Allow truncation of message data. If the Reply Data length is too small to hold the complete message, this option allows the MQGET call to obtain as much of the message as it can. A warning completion code is generated and processing is completed.

**EHCGQ_BROWSE_FIRST x'0100'**
Position the browse cursor on the first matching message in the queue.

**EHCGQ_BROWSE_MSG_UNDER_CURSOR x'0200'**
Using this option, Browse message under browse cursor ignores the input data for message ID and correlation ID. This option can not be used with EHCGQ_WAIT.

**EHCGQ_BROWSE_NEXT x'0300'**
Browse the next message on the queue from the current cursor position.

The GQ call with a BROWSE option is non-destructive. Only one of the BROWSE options can be used on any one GQ call. Use of a BROWSE option establishes a browse cursor, which is always established in front of the queue and then positioned by BROWSE_NEXT or BROWSE_FIRST. A queue handle that is in use for browsing can not be used in a transaction.

**Message Type**
MQSeries Link server supports a subset of the MQSeries message types, which have been defined as EHCMT constants. The values are:

**EHCMT_DATAGRAM x'4447'**
Use a datagram when you do not require a reply from the application that receives the message (that is, gets the message from the queue).

**EHCMT_REQUEST x'5251'**
Use a request message when you want a reply from the application that receives the message. The reply queue name must be defined.

**EHCMT_REPLY x'5250'**
Use a reply message when you reply to another message.

**EHCMT_REPORT x'5254'**
This message type may be generated by MQSeries. Queue Manager. It should not be created by LANDP applications.

**Note:** It is the application's responsibility to ensure that a EHCMT_REPLY is sent in response to a EHCMT_REQUEST. This is not managed by the MQSeries Link server or the MQSeries Queue Manager.

**Reply Queue name**
This is the name of the queue you use to reply to a EHCMT_REQUEST message. If the Message type is not EHCMT_REQUEST the return value is normally null.

**MQ Reason Code**
Contains the MQSeries error return code if an underlying MQSeries API call has failed. The LANDP server return code is also non-zero.

If the MQSeries API call is successful, this field contains the MQSeries value for MQRC_NONE.

## MQPUT (PQ function)

The PQ function issues a MQPUT call to MQSeries, which puts a message on a queue. The MQSeries Link server ensures that the requested queue is open for output. It is valid for a session to use PQ and GQ on the same queue.

| CPRB field | Content and description |
|---|---|
| Function code | PQ |
| Request PARMLIST length | 118 |
| Request DATA length | Length of the message |
| Reply PARMLIST length | 90 |
| Reply DATA length | 0 |
| Replied PARMLIST length | 90 |
| Replied DATA length | 0 |

*Table 21. Request PARMLIST values*

| Offset | Length | Content |
|---|---|---|
| 0 | 2 | Queue handle |
| 2 | 24 | Queue name |
| 26 | 24 | Message ID |
| 50 | 24 | Correlation ID |
| 74 | 12 | User identifier |
| 86 | 24 | Reply queue name |
| 110 | 2 | Message type |
| 112 | 4 | Message expiry |
| 116 | 2 | LANDP PQ message options |

*Table 22. Request DATA values*

| Offset | Length | Content |
|---|---|---|
| 0 | | Message |

*Table 23. Reply PARMLIST values*

| Offset | Length | Content |
|---|---|---|
| 0 | 2 | Queue handle |
| 2 | 24 | Queue name |
| 26 | 24 | Message ID |
| 50 | 24 | Correlation ID |
| 74 | 12 | User identifier |
| 86 | 4 | MQ reason code |

A number of the following fields are used for input and output. Take care that these fields are correctly defined when repeating the MQSeries Link server function calls.

**Queue Handle**
The queue handle is an input/output field. The first time PQ is called, a queue handle is returned. When multiple PQ calls are being issued, the returned queue handle should be reused. However, the queue handle is associated with the queue name and the user identifier. If either of these change, a new queue handle is needed. To indicate that a new handle is required the field should be set to the value X'0000'.

This queue handle is used in the CQ function call.

**Queue Name**
The local name of the queue as defined on the MQSeries queue manager. The name must not contain leading or embedded blanks, but can contain trailing blanks. The first null character and all characters following it are treated as blanks. For more information about queue names, see the *MQSeries Application Programming Guide*.

**Message ID**
The message ID is an input/output field. The LANDP application can define the Message ID, but messages on the MQseries Queue Manager are expected to be uniquely identified by this field. The uniqueness of the Message ID is not enforced by either the MQSeries Link server or the MQSeries queue manager.

If you set the Message ID to null prior to putting a new message, the MQSeries queue manager returns a unique message ID. Another way to obtain an unique message ID is to use the EHCPQ_NEW_MSG_ID option.

**Correlation ID**
The correlation ID is an input/output field. When option EHCPQ_NEW_CORREL_ID is defined, correlation ID is an output field, otherwise it is an input field. A typical use of this field is to identify a group of messages. If each message in the group is given the same correlation ID, it is simple for the GQ function to browse the message group.

**User identifier**
The user of the LANDP application that is to be placed in the MQSeries Message

Descriptor. This user identifier can be the null string, in which case the User ID is determined by the MQseries queue manager. This varies with the operating system environment.

On OS/2 systems, the user identifier is 'OS2'. On Windows,the user identifier is the user name associated with the MQSeries Link server process. The output value is the User Identifier that the MQSeries queue manager transmitted with the message. If an input value is passed in PARMLIST, the MQSeries option MQPMO_SET_IDENTITY_CONTEXT is defined, and the input value is included as part of the message context.

When the User Identifier is defined for the first PQ request, the underlying MQOPEN is given the option MQOO_ALTERNATE_USER_AUTHORITY and this field is be used as the AlternateUserId. (For P1 the equivalent option MQPO_ALTERNATE_USER_AUTHORITY is used.)

**Message Type**

MQSeries Link server supports a subset of the MQSeries message types, which have been redefined as EHCMT constants. The values are :

**EHCMT_DATAGRAM x'4447'**
Use a datagram when you do not require a reply from the application that receives the message (that is, gets the message from the queue).

**EHCMT_REQUEST x'5251'**
Use a request message when you want a reply from the application that receives the message. The reply queue name must be defined.

**EHCMT_REPLY x'5250'**
Use a reply message when you reply to another message.

**EHCMT_REPORT x'5254'**
This message type may be generated by MQSeries. Queue Manager. It should not be created by LANDP applications.

**Note:** It is the application's responsibility to ensure that a EHCMT_REPLY is sent in response to a EHCMT_REQUEST. This is not managed by the MQSeries Link server or the MQSeries Queue Manager.

**Reply Queue name**

This is the name of the queue to which a reply or report message should be sent. This field must be defined when a message type of EHCMT_REQUEST is issued. It may also be used by the MQSeries queue manager when a report message is generated as a result of a problem when processing the MQPUT.

When a reply queue is not needed, this field should be the null string. If the MQSeries queue manager needs a reply queue name for a report, it creates one from its own queue's ReplyToQ definition.

**Message Expiry**

The MQSeries Queue Manager marks a message as eligible for discarding if it exists on the destination queue after the expiry period of time has elapsed.

**Positive** A positive value is interpreted as a period of time expressed in tenths of a second.

**Zero** A value of zero specifies the use of the supervisor's LAN timeout period as defined in the CPRB field `ehctimeout`. If `ehctimeout` is defined as X'0000' a default period of 30 seconds is used.

**Negative** A negative value means that the wait period is unlimited.

**LANDP PQ Message Options**

The LANDP PQ Message Options use Version 2 of the MQSeries put message options structure. The MQSeries option MQPMO_SET_IDENTITY_CONTEXT is always defined and the user identifier is always included in the MQSeries Message Descriptor.

For data conversion, the MQSeries Message Descriptor field CodedCharSetId is set to the constant MQCCSI_Q_MGR and the Encoding field is defined as MQENC_NATIVE.

The MQSeries Link server supports a subset of the MQSeries Put Message Options. The supported options have been defined as EHCPQ values:

**EHCPQ_NONE x'0000'**
No options specified.

**EHCPQ_NEW_MSG_ID x'0001'**
Generate a new message identifier. This option ensures that the MQSeries queue manager returns a new unique message identifier. The input Message ID is ignored.

**EHCPQ_NEW_CORREL_ID x'0002'**
This option works in the same way as EHCPQ_NEW_MSG_ID. It requests the MQSeries queue manager to generate a new unique correlation identifier.

**EHCPQ_FAIL_IF_QUIESCING x'0004'**
This option forces the MQPUT or MQPUT1 call to fail if the MQSeries queue manager is in the quiescing state.

**MQ Reason Code**

Contains the MQSeries error return code if an underlying MQSeries API call has failed. The LANDP server return code is also non-zero.

If the MQSeries API call is successful, this field contains the MQSeries value for MQRC_NONE.

## MQPUT1 (P1 function)

The P1 function issues an MQPUT1 call to MQSeries, which places a single message on a queue. For putting a single message, this function is much more efficient than the PQ function. P1 avoids the need for a CQ. The only difference in the parameter list between P1 and PQ is that P1 does not use the Queue Handle and PQ does. Otherwise all parameters are the same. The parameters are fully described in the text for function PQ.

| CPRB field | Content and description |
|---|---|
| Function code | P1 |
| Request PARMLIST length | 118 |

| CPRB field | Content and description |
|---|---|
| Request DATA length | Length of the message |
| Reply PARMLIST length | 90 |
| Reply DATA length | 0 |
| Replied PARMLIST length | 90 |
| Replied DATA length | 0 |

*Table 24. Request PARMLIST values*

| Offset | Length | Content |
|---|---|---|
| 0 | 2 | Reserved |
| 2 | 24 | Queue Name |
| 26 | 24 | Message ID |
| 50 | 24 | Correlation ID |
| 74 | 12 | User Identifier |
| 86 | 24 | Reply queue name |
| 110 | 2 | Message type |
| 112 | 4 | Message Expiry |
| 116 | 2 | LANDP PQ Message Options |

*Table 25. Request DATA values*

| Offset | Length | Content |
|---|---|---|
| 0 | | Message |

*Table 26. Reply PARMLIST Values*

| Offset | Length | Content |
|---|---|---|
| 0 | 2 | Reserved |
| 2 | 24 | Queue Name |
| 26 | 24 | Message ID |
| 50 | 24 | Correlation ID |
| 74 | 12 | User Identifier |
| 86 | 4 | MQ reason code |

## Rollback (RB function)

This function ends the current transaction's unit of work by undoing all changes to the queue manager since the last CP, BT or RB function request. The changes defined in the incomplete unit of work are removed as if they had never been requested. This function is useful to maintain data integrity when a problem appears in transaction processing. For more details on transactional support, refer to "Begin Transaction (BT)" on page 123.

| CPRB field | Content and description |
|---|---|
| Function code | RB |
| Request PARMLIST length | 0 |
| Request DATA length | 0 |
| Reply PARMLIST length | 4 |
| Reply DATA length | 0 |
| Replied PARMLIST length | 4 |
| Replied DATA length | 0 |

*Table 27. Reply PARMLIST Values*

| Offset | Length | Content |
|---|---|---|
| 0 | 4 | MQ reason Code |

# Chapter 7. TCP/IP Wide Area Communications server

The TCP/IP Wide Area Communications server emulates both the SNA server and the PPC server and can be used in place of those servers to communicate over TCP/IP networks instead of SNA networks. See the *LANDP Servers and System Management* book for information about which service names (SNA or PPC) can be used with which session partners.

See Chapter 5, “Program-to-program communication server” on page 105 for a description of the programming interface to use with the PPC service name. All the function codes are supported. When using this interface with the MPTN protocol the full programming interface as described is supported. When using this interface with the TCP protocol some of the parameters (for example, Mode_name on OP function) are ignored and some returned values are emulated (made up) for compatibility.

See Chapter 2, “SNA communication server” on page 35 for a description of the programming interface to use with the SNA service name. All the function codes except DC and QC are supported. When using this interface with the MPTN protocol the full programming interface as described is supported. When using this interface with the Telnet protocol some of the parameters are ignored and some returned values (for example STSN and BIND) are emulated (made up) for compatibility.

© Copyright IBM Corp. 1992, 2002

# Part 3. I/O device servers

The LANDP family offers several servers that allow application programs written in a high-level programming language to access specific I/O devices. The I/O devices are connected to LANDP for DOS, LANDP for OS/2, LANDP for Windows, or LANDP for Linux workstations that have the corresponding server and device driver installed. The servers provide the capabilities for other computers to share the I/O devices.

This part of the book contains the following chapters:

**Chapter 8, "Financial printer server" on page 141**

This chapter describes the functions that provide access to:

- IBM 4712 Transaction Printer
- IBM 4722 Document Printer
- IBM 4772 Passbook/Document Printer
- IBM 4009 Universal Banking Printer
- IBM 9055-002 Document Printer
- IBM 9068-S01 Multi-Purpose Passbook Printer
- IBM 9069 Multi-Purpose Transaction Printer

**Chapter 9, "IBM 4748 printer server" on page 167**

This chapter describes the functions that provide access to the IBM 4748 Document Printer.

**Chapter 10, "IBM 4770 printer server" on page 189**

This chapter presents the functions that provide access to the IBM 4770 Ink Jet Transaction Printer and the IBM 9068-D01 Multi-Purpose Passbook Printer.

**Chapter 11, "Printer manager server" on page 199**

This chapter describes the functions provided by an additional printer manager server for printers that are attached to the parallel port of a LANDP for DOS workstation. The printer manager server serializes the printer operation. This means that one print request is completed before the next is accepted. With the operator interface or the local resource manager, the printer manager server allows the operator to issue commands to control printing and to display the status at a parallel printer port.

**Chapter 12, "Magnetic stripe reader/encoder server" on page 203**

This chapter describes the functions that provide access to:

- IBM 4717 Magnetic Stripe Reader/Encoder
- IBM 4777 Magnetic Stripe Reader/Encoder

© Copyright IBM Corp. 1992, 2002

- The Magnetic Stripe Reader component of the IBM 4778 Personal Identification Number Key-pad

**Chapter 13, “Personal identification number pad server” on page 223**

This chapter describes the functions that provide access to:

- IBM 4718 Personal Identification Number Key-pad
- IBM 4778 Personal Identification Number Key-pad, including its Magnetic Stripe Reader component

# Chapter 8. Financial printer server

This server processes requests for the following printers:

- IBM 4712 Transaction Printer
- IBM 4722 Document Printer
- IBM 4772 Passbook/Document Printer
- IBM 4009 Universal Banking Printer
- IBM 9055-002 Document Printer
- IBM 9068-S01 Multi-Purpose Passbook Printer
- IBM 9069 Multi-Purpose Transaction Printer (*with MICR only in LANDP for Windows*)

Besides servicing requests for printing, this server also provides access to:

- The magnetic stripe reader of the IBM 4722 Document Printer Model 003, the IBM 4772 Passbook/Document Printer, the IBM 9055-002 Document Printer, or IBM 9068-S01 Multi-Purpose Passbook Printer, and the MICR reader of the IBM 9069 Multi-Purpose Transaction Printer.
- The operator panel display and keyboard of the IBM 4772 Passbook/Document Printer
- The magnetic stripe reader, operator display panel, and keyboard of the IBM 9068-S01 Multi-Purpose Passbook Printer

The 9069 printer can be configured to emulate the 4712 either with or without the addition of the MICR feature. LANDP for DOS and LANDP for OS/2 support the 9069 in 4712-emulator mode, but without MICR support.

Except where stated, all functions operate in the LANDP for DOS, LANDP for OS/2, and LANDP for Windows environments.

This chapter provides information on how the server handles service requests and provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields. If you need more information about the CPRB fields, see Appendix A, “Connectivity programming request block” on page 669.

*Table 28. Function Codes used in the Financial Printer Server*

| Function code | Description | Env. | Page |
|---|---|---|---|
| AR | Arm operator panel for user input | 02NL | 146 |
| CD | Clear operator panel display | 02NL | 148 |
| CH | Check printing status | 02NL | 148 |
| CL | Close printer | 02NL | 149 |
| DF | Format parameter load | 02NL | 149 |
| DU | Download user characters | 02NL | 155 |
| EC | Get error counters | 02NL | 156 |
| LL | Set lights | 02NL | 158 |
| OP | Open printer | 02NL | 158 |
| RC | Read code page | -2NL | 159 |

© Copyright IBM Corp. 1992, 2002

*Table 28. Function Codes used in the Financial Printer Server (continued)*

| Function code | Description | Env. | Page |
|---|---|---|---|
| RD | Read data | 02NL | 160 |
| RM | Set redirection mode | 0--- | 161 |
| SC | Set code page | 02NL | 162 |
| WR | Write to printer | 02NL | 162 |
| And these functions of the local resource manager: | | | |
| CC | Cancel pending functions | 0--- | 546 |
| GS | Get status | 0--- | 547 |
| HP | Change printer to remote mode | 0--- | 548 |
| LP | Change printer to local mode | 0--- | 549 |

The first and second columns of the table give the function code and the name of the function. The third column shows the operating environment of the function, as explained in “Operating environments” on page xxvi. “02NL” means that it is available from LANDP for DOS, LANDP for OS/2, LANDP for Windows, and LANDP for Linux servers. The last column refers to the page where you can find the function described.

The printer can only be used by one application at any given time. To allow for sharing of the printer, the application should always be programmed to:
1. Acquire the printer using the OP function
2. Perform the printing or magnetic stripe read/encode operation
3. Release the printer for other applications, using the CL function

You can also use the printer manager server to control access to these printers (see Chapter 11, “Printer manager server” on page 199).

The financial printer server has two modes of operation:
- Formatted mode
- ASCII unformatted mode

To use formatted mode, a DF function call must be issued to set the document characteristics such as page length, line length, writing pitch, and so on. These characteristics are entered during customization, stored in a disk file, and read when the server is loaded.

Control codes allowed in this mode are a subset of the overall control code set available for the printers. The correct page format is activated by means of the DF function, which establishes a customized format definition. This format definition can be designated to be used by only one or by both operators. The following table indicates the possible printer combinations and whether the operator A/B keys must be pressed in order to start the printer.

| Printer Sharing and A/B Key Usage | | |
|---|---|---|
| **DF Function Setting** | **Printer Defined As Shared** | **Printer Defined As Unshared** |
| Formatted:<br>Start key required<br>Shared using A/B keys | A/B key pressing required | Invalid |
| Formatted:<br>Start key required<br>Not shared using A/B keys | Either A or B key pressing required | Either A or B key pressing required |
| Formatted:<br>No start key required<br>Not shared using A/B keys | A/B key pressing not required | A/B key pressing not required |
| Unformatted: | A/B key pressing required | A/B key pressing not required |

In formatted mode, the financial printer server does not support all the control sequences that relate to page media change and page length change. The printer server does not discard these control sequences. However, using them may yield unpredictable results.

In unformatted mode, all the control codes of the printer are available. The server passes the entire data stream transparently to the printer.

After a successful OP function, the printer is available and ready to work in unformatted mode. To enter formatted mode, the format definition is loaded using the DF function. When working in formatted mode you can switch to unformatted mode by issuing a DF function call with value 0 in the data length field. During customization the working mode is determined: shared or unshared printer mode and shared or unshared formatted page mode. This provides for either automatic activation of the printer or the requirement for A or B key pressing to start printing. Refer to the previous table.

## Using the financial printer server

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | 26 | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 26 | 2 | 26 | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 96 | 8 | PR47X2## | Server name |
| **Note:** During customization, a value is defined for ## relating it to a serial or parallel port to which the printer is attached | | | |

The following fields are variable and are discussed in each function request description:
- Function code
- Request DATA length
- Reply DATA length

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 44 | 2 | | Replied PARMLIST length |
| 46 | 2 | | Replied DATA length |

If the request was successful, the *router return code* and the *server return code* are both X'00000000'. In all other cases, see the appropriate section in the *LANDP Problem Determination* book to see if you should take any action. The return values in Reply PARMLIST and Reply DATA should be ignored if there is an error.

The following fields are variable and are discussed in each function request description:
- Replied PARMLIST length
- Replied DATA length

**PARMLIST:** The Request PARMLIST and Reply PARMLIST fields for the financial printer server are:

| Offset | Length | Content |
|---|---|---|
| 0 | 1 | Flag1 |
| 1 | 1 | Flag2 |
| 2 | 1 | Flag3 |
| 3 | 1 | Flag4 |
| 4 | 3 | Reserved |
| 7 | 1 | Flag8 |
| 8 | 1 | Flag9 |
| 9 | 2 | Number of written or encoded characters. It contains the number of application program data stream characters that have been printed or encoded. Control characters are not counted. |

**DATA:** Request DATA and Reply DATA contain data to be sent to, or received from, the server. The use of these areas is explained in the description of each function request.

## Financial printer server flag descriptions

Flag1 (returned by the server or set by the application program)

**'D'** In the DF function, if this flag is 'D', Request DATA contains a format description. Any other value shows that Request DATA contains a format description name.

**'M'** In the AR function, if this flag is 'M', the keyboard input of the operator panel of the 4772 or 9068-S01 printer is to be masked.

**'P'** The return code corresponds to the previous operation.

Flag2 (set by the application program)

**'M'** The function is to read data from the MICR reader of the 9069 printer.

**'O'** The function is for the operator panel of the 4772 or 9068-S01 printer.

**'R'** The function is for the read/encode magnetic stripe component. However, during an open function with flag2 = 'R', both the read/encode magnetic stripe and printer components are opened.

Flag3 (set by the application program)

**'R'** In an OP or CL function, this flag causes the resetting of the printer configuration. Resetting means:

- End emphasized
- End NLQ
- Reset font ROM
- Set bidirectional printing
- End subscript or superscript style
- End double width or condensed printing
- End double height and double line feed (4009 and 4772)
- End overscore or underscore mode
- Set text line spacing to 1/6 inch and text character spacing to 10 characters per inch
- Set Character Set 1 decode
- Set default tabs
- Set format type to Cut-form
- Reset 4772 or 9068-S01 margins
- Set 4009 vertical units to 1/216 inch
- Set top of form

**'S'** By default, the WR function is asynchronous. The flag value 'S' makes it synchronous.

Flag4 (returned by the server)

**'Y' or 'N'**

In the CH function on LANDP for OS/2 and LANDP for Windows, the flag

value 'Y' is returned if a passbook or sheet of paper is in the printer. 'N' is returned if there is nothing in the printer.

Flag8 (returned by the server)
**'A'** Active interface
**'B'** Active interface

If a form has been inserted, the printer starts the interfaces as follows:

Interface A is started.
In journal mode and key A is pressed.
When processing passbook/document forms and key A is pressed.

Interface B is started.
In journal mode and key B is pressed.
When processing passbook/document forms and key B is pressed.

The printer only prints when you do one of the following:
- You select "auto-start" in the format definition
- You have to press a key and either interface is active
- You select shared mode and the proper interface is active.

All the functions are reflected in the values of flag8 covering all the possibilities that can occur in setting the active interface.

Flag9 (returned by the server) indicates which display key of the 4772 or 9068-S01 printer has been pressed.
**'1'** Function key 1
**'2'** Function key 2
**'3'** Function key 3
**'4'** Enter key

Flag9 (set by the application program)
**'4'** Select IBM 4720 Emulation Mode.

The Financial Printer Server converts the IBM 4720 Data Stream Control Sequences into equivalent IBM 4722 Data Stream Control Sequences.

## Request reference

This section gives you the programming information required to write application programs or your own servers that issue service requests to the financial printer server. The information is listed in alphabetical order of function code.

### Arm operator panel for user input (AR function)

This function arms the operator panel of the IBM 4772 or 9068-S01 printer to accept user input. It also writes data, such as a prompt or operator message, to the panel.

This function must be used before any further data is written to, or read from, the operator panel, using an RD (read data), WR (write to printer), or CD (clear operator display panel) function request.

When the AR function is requested, an asynchronous Z4 request is sent to the supervisor to start periodical timer requests (see “Asynchronous request—asking for TT request (Z4 function)” on page 29). The events requested are no longer valid after an RD or CD function, so the server generates a Z5 request at that time to cancel the timer requests (see “Asynchronous event notification or cancellation (Z5 function)” on page 30).

There are several ways of using this function:

- Request the WM supervisor local function (see “Wait for asynchronous events (WM function)” on page 24) to wait for the asynchronous notification from the Z5 function, and then read the data using the RD function (see “Read data (RD function)” on page 160). Possible reason codes from the Z5 request are:

  **X'0000':** An error has occurred. Use RD or CD to discover the error code.

  **X'3031':** Function key 1 pressed.

  **X'3032':** Function key 2 pressed.

  **X'3033':** Function key 3 pressed.

  **X'3034':** Enter key pressed.
- Request the RD function repetitively.
- Request the CD function to cancel the input.
- Request the AA supervisor local function repetitively (see “Ask for asynchronous events (AA function)” on page 6) to wait for asynchronous notification, and then request the RD function.

**Notes:**

1. Until a CD function is used to clear the data or an RD function reads the data, no other function (except WM or AA) can be requested.
2. The events are no longer valid after a CD or RD function request.

| CPRB Field | Content/Description |
|---|---|
| Function code | AR |
| Request DATA length | 2, or in the range 4 to 36 |
| Request PARMLIST length | 2 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | **'M'** Mask keyboard input (not displayed)<br>**other** Do not mask keyboard input |
| 1 | 1 | **'O'** Operator panel<br>**other** Not operator panel: ignore this request. |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Start position (X'00' to X'1F') of the display field that is to be defined for input |
| 1 | 1 | Input field length in hexadecimal |
| 2 | 1 | Start position (X'00' to X'1F') of the display field that is to be written |
| 3 | | Data to be written to the display panel (up to 32 bytes) |
| **Note:** If the Request DATA length is 2, no message is sent to the display. | | |

## Clear operator panel display (CD function)

This function clears the display of the operator panel of a 4772 or 9068-S01 printer. It also cancels any pending timer requests that were set up during the AR function.

| CPRB Field | Content/Description |
|---|---|
| Function code | CD |
| Request DATA length | 0 |
| Request PARMLIST length | 2 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 1 | 1 | **'O'** Operator panel<br>**other** Not operator panel: ignore this request |

## Check printing status (CH function)

This function requests the return code from the last operation on the printer. The function can only be used for a printer attached to a serial port.

Use this function to check the completion of a write or magnetic stripe encoding with an asynchronous WR function.

| CPRB Field | Content/Description |
|---|---|
| Function code | CH |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 9 | 2 | Print or encode character count from previous operation |
| 4 | 1 | Flag4 setting (LANDP for OS/2 and LANDP for Windows): Y (document inserted) or N (no document inserted) |

## Close printer (CL function)

This function closes the financial printer server and releases it for use by another application program. Use this function, when your application program is finished with the printer, to free it for use by another application program.

Flag3 can be used to reset the printer.

| CPRB Field | Content/Description |
|---|---|
| Function code | CL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 11 |

## Format parameter load (DF function)

This function starts the financial printer server formatting mode and begins page formatting with the customized format defined in Request DATA. If the Request DATA length field is zero, no formatting is done and unformatted mode is set.

The server can store two different formats—one for the print device, and another for the magnetic stripe encoder of the IBM 4722 Model 003 Document Printer. The value stored in flag2 of Request PARMLIST is used to specify the parameters for a magnetic stripe format. This function can only be used for a printer attached to a serial port.

| CPRB Field | Content/Description |
|---|---|
| Function code | DF |
| Request DATA length | **0** Unformatted mode<br>**8** Name of format (flag1 not 'D')<br>**11 or 13** Format definition (flag1 = 'D') |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 11 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 setting |
| 1 | 1 | Flag2 setting |

## Request DATA values

**Flag1 ≠ 'D'** Request DATA must contain the name of the desired format as assigned during customization. The Request DATA length field contains X'0008'.

**Flag1 = 'D'** Request DATA must contain the format definition. In most cases the information provided in the following tables should be sufficient. For more information, see the *IBM 4700 Financial I/O Devices: Programming Guide.*

## Format definition for 4009, 4712, 4722, 4772, and 9068-S01 printers

**Note:** Bytes 11 and 12 apply only to 4009, 4772, and 9068-S01 printers.

| Byte | Journal | Passbook | Cut-forms |
|---|---|---|---|
| 0 | X'02' or X'82' | X'00' | X'01' |
| 1 | Page size in lines | Page size in lines | Page size in lines |
| 2 | Attention line number | Centerfold begin column/line number | Attention line number |
| 3 | Line length | Centerfold skip in number lines/chars | Step offset in number of steps |

| Byte | Journal | Passbook | Cut-forms |
|---|---|---|---|
| 4 | Device bits:<br>7: not used<br>6: 0=ignore byte 6<br>1=use byte 6 extension<br>5, 4: not used<br>3: 0=byte 5 not valid<br>1=valid byte 5 extension<br>2, 1: not used<br>0: 0=10 cpi<br>1=12 cpi | Step offset in number of steps | Line offset in number of lines |
| 5 | Only valid if bit 3 of byte 4 is 1. Bits:<br>7: 0=regular font<br>1=quality font<br>6: not used<br>5: 0=no double strike<br>1=quality font<br>4: 0=not emphasized<br>1=emphasized<br>3,2:<br>extended pitch:<br>00=use byte 4<br>01=10 cpi<br>10=12 cpi<br>11=17.1 cpi<br>1, 0: not used | Line offset in number of lines | Line length |
| 6 | Only valid if bit 6 of byte 4 is 1. Bits:<br>7: 0=no shared/autostart<br>1=shared/start key required<br>6: reserverd<br>5, 4: not used<br>1=quality font<br>3: 0=6 lpi<br>1=8 lpi<br>2–0: not used | Left margin column number | Device bits:<br>7: not used<br>6: 0=ignore bytes 8 and 9<br>1=use bytes 8 and 9 extension<br>5: 0=not shared<br>1=shared<br>4: 0=start switch<br>1=autostart mode<br>3: 0=byte 7 not valid<br>1=valid byte 7 extension<br>2: 0=line length check<br>1=auto new line<br>1: 0=5 lpi<br>1=6 lpi<br>0: 0=10 cpi<br>12 cpi |

| Byte | Journal | Passbook | Cut-forms |
|---|---|---|---|
| 7 | Reserved | Line length | Only valid if bit 3, byte 6 is 1. Bits:<br>7: 0=regular font<br>1=quality font<br>6: not used<br>5: 0=no double strike<br>1=double strike<br>4: 0=not emphasized<br>1=emphasized<br>3, 2: extended pitch<br>00=use byte 6, bit 1<br>01=10 cpi<br>10=12 cpi<br>11=17.1 cpi<br>1, 0: acceptable skew:<br>if initial vertical offset = 3.9 mm:<br>00=1.37 mm<br>01=2.05 mm<br><br>if initial vertical offset = 6.1 mm:<br>00=1.37 mm<br>01=2.05 mm<br>10=2.74 mm<br>11=3.42 mm |
| 8 | Reserved | Device bits:<br>7: 0=horizontal fold<br>1=vertical fold<br>6: 0=ignore byte 10<br>1=use byte 10 extension<br>5: 0=not shared<br>1=shared<br>4: 0=start switch<br>1=autostart mode<br>3: 0=byte 9 not valid<br>1=valid byte 9 extension<br>2: 0=line length check<br>1=auto new line<br>1: 0=5 lpi<br>1=6 lpi<br>0: 0=10 cpi<br>1=12 cpi | Only valid if bit 6 of byte 6 is 1. Bits:<br>7,6: not used<br>5: bottom edge validation:<br>0=off<br>1=on<br>4: initial vertical offset:<br>0=6.1 mm<br>1=3.9 mm<br>3: 0=use bit 1 of byte 6<br>1=8 lpi<br>2–0: not used |

| Byte | Journal | Passbook | Cut-forms |
|---|---|---|---|
| 9 | Reserved | Only valid if bit 3 of byte 8 is 1. Bits:<br>7–4: not used<br>3, 2: extended pitch<br>00=use byte 8 bit 7<br>01=10 cpi<br>10=12 cpi<br>11=17.1 cpi<br>1, 0: acceptable skew:<br>if vertical offset = 3.9 mm:<br>00=1.37 mm<br>01=0.68 mm<br><br>if vertical offset = 6.1 mm:<br>00=1.37 mm<br>01=0.68 mm<br>10=2.05 mm | Horizontal character offset.<br><br>Only valid if bit 6 of byte 6 is 1. |
| 10 | Reserved | Only valid if bit 6 of byte 8 is 1. Bits:<br>7–5: not used<br>4: initial vertical offset<br>0=6.1 mm<br>1=3.9 mm<br>3–0: not used | Reserved |
| 11 | Reserved | Character pitch (overrides that in previous bytes):<br>**X'0A':** 10 cpi<br>**X'0C':** 12 cpi<br>**X'0F':** 15 cpi (4772 only)<br>**X'11':** 17.1 cpi | Character pitch (overrides that in previous bytes):<br>**X'0A':** 10 cpi<br>**X'0C':** 12 cpi<br>**X'0F':** 15 cpi (4772 only)<br>**X'11':** 17.1 cpi |
| 12 | Reserved | Reserved | Device bits:<br>7: character width<br>0=single<br>1=double<br>6: character height<br>0=single<br>1=double<br>5:<br>0=single line feed<br>1=double line feed<br>4–0: not used |

## Format definition for printers with magnetic stripe

| Byte | Read/Encode Magnetic Stripe |
|---|---|
| 0 | X'00' |
| 1 | Device bits:<br>**7:**<br>0=no displaced<br>1=displaced<br>**6:**<br>0=double recording<br>1=single recording<br>**5:** 0=synchronous<br>**4:**<br>0=no check new document<br>1=check new document<br>**3–0:**<br>X'4'=IBM 3604/4704 position 1<br>X'5'=IBM 3604/4704 position 2<br>X'D'=ISO/DIN position 2<br>X'E'=ISO/DIN position 1 |
| 2 | Book width: X'mm' (127 - 210) |

## Example configuration

The following example shows the configuration for a printer in passbook mode:

| **Example Configuration: Passbook X'00 1E 0A 04 00 01 02 50 5E 08 00'** | | |
|---|---|---|
| **Byte** | **Description** | **Value** |
| 0 | Passbook Identifier | X'00' |
| 1 | Page size in lines = 30 | X'1E' |
| 2 | Center field begins at column/line number = 10 | X'0A' |
| 3 | Center fold skip in number lines/chars = 4 | X'04' |
| 4 | Step offset in number of steps = 0 | X'00' |
| 5 | Line offset in number of lines = 1 | X'01' |
| 6 | Left margin column number = 2 | X'02' |
| 7 | Line length = 80 | X'50' |

| Example Configuration: Passbook X'00 1E 0A 04 00 01 02 50 5E 08 00' | | |
|---|---|---|
| **Byte** | **Description** | **Value** |
| 8 | Horizontal fold<br>Use byte 10 extension<br>Not shared<br>Auto-start mode<br>Valid byte 9 extension<br>Auto new line<br>6 lines per inch<br>No information | 0<br>1<br>0<br>1<br>1<br>1<br>1<br>0<br><br>X'5E' |
| 9 | Not used<br>Not used<br>Not used<br>Not used<br>Characters per inch = 12<br>Maximum skew = 1.37 | 0<br>0<br>0<br>0<br>10<br>00<br><br>X'08' |
| 10 | Not used<br>Not used<br>Not used<br>Initial vertical offset = 6.1<br>Not used<br>Not used<br>Not used<br>Not used | 0<br>0<br>0<br>0<br>0<br>0<br>0<br>0<br><br>X'00' |

## Download user characters (DU function)

This function loads user-defined characters into the 4772 or 9068-S01 printer. Up to eight characters can be loaded, corresponding to the ASCII code points X'00' to X'07'.

| CPRB Field | Content/Description |
|---|---|
| Function code | DU |
| Request DATA length | 0 to 72 |
| Request PARMLIST length | 2 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 1 | 1 | **'O'** Operator panel<br>**other** Not operator panel: ignore this request |

### Request DATA values

User-defined character image data. The data consists of images made up of eight horizontal five-dot slices. The format of each character specification is:

```
ASCII code point + eight bytes of image data (one byte per slice)
```

For example,

```
X'01' + X'1F 0E 04 04 0E 15 04 0E'
```

forms this character at code-point 01:



## Get error counters (EC function)

This function retrieves error counters, interface setting, printer engineering change (EC) level data, printer ID data, printer personalization data, and redirection setting.

**Note:** This information is unavailable when the printer is parallel-attached and P1 error (function not supported) is returned.

| CPRB Field | Content/Description |
|---|---|
| Function code | EC |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | LANDP for DOS: 12 or 15<br>LANDP for OS/2: 27 |
| Reply PARMLIST length | 26 |
| Replied DATA length | LANDP for DOS: 12 or 15<br>LANDP for OS/2: 27 |
| Replied PARMLIST length | 11 |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Error counters. This represents the number of soft errors that have occurred (successfully retried). The counters continue to increment until X'FFFF' is reached. |
| 2 | 2 | Error counters. This represents the number of hard (unrecoverable) errors that have occurred. The counters continue to increment until X'FFFF' is reached. |
| 4 | 2 | Power on reset (POR) status bytes 1 and 2. These bytes contain the printer feature bytes as presented to the device driver at POR. |
| 6 | 1 | Printer engineering change (EC) level data. This byte contains the EC level of the control read only storage in the printer. |
| 7 | 1 | Printer identification data. This byte contains the identification value for the printer:<br>X'19': 4712 Transaction Printer Model 001 or 002<br>X'1B': 4722 Document Printer Model 001 or 002<br>X'1C': 4722 Document Printer Model 003<br>X'D0': 4772 Passbook/Document Printer in SBCS mode<br>X'D1': 4772 Passbook/Document Printer in DBCS mode<br>X'1B': 4009 Universal Banking Printer<br><br>**Note:** This identification value can come from any of the printers that are set up to emulate the above-named printers (for example: the 9055 and 9068-S01). |
| 8 | 1 | Printer personalization data. This byte contains the personalization switch settings of the printer. The personality setup switch positions and meanings are defined in the printer user's guide which is contained in the *IBM 4700 Financial I/O Devices: Programming Guide*. |
| 9 | 3 | **LANDP for DOS:** Redirection settings. A maximum of three parallel ports can be redirected with each port having a redirection mode. A value of X'00' shows that no redirection is in effect. A value of X'01' shows that redirection for that parallel port occurs. The first byte is for port 1, the second for port 2, and the third for port 3.<br><br>**LANDP for OS/2 and LANDP for Windows**: not used. |
| For printers with read/encode magnetic stripe: | | |
| 12 | 2 | Magnetic read/encode component POR status bytes 1 and 2. These bytes contain the read/encode magnetic stripe feature bytes as presented to the device driver at POR. |
| 14 | 1 | Printer engineering change (EC) level data for the read/encode magnetic stripe component. |
| For LANDP for OS/2 only: | | |
| 15 | 2 | Read/encode magnetic stripe error counters. This represents the number of soft errors (errors that have been successfully retried) that have occurred. The counters continue to increment until X'FFFF' is reached. |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 17 | 2 | Read/encode magnetic stripe error counters. This represents the number of hard (irrecoverable) errors that have occurred. The counters continue to increment until X'FFFF' is reached. |
| 19 | 2 | Printer link error counters. This represents the number of soft (successfully retried) link errors that have occurred. The counters continue to increment until X'FFFF' is reached. |
| 21 | 2 | Printer link error counters. This represents the number of hard (irrecoverable) link errors that have occurred. The counters continue to increment until X'FFFF' is reached. |
| 23 | 2 | Read/encode magnetic stripe link error counters. This represents the number of soft (successfully retried) link errors that have occurred. The counters continue to increment until X'FFFF' is reached. |
| 25 | 2 | Read/encode magnetic stripe link error counters. This represents the number of hard (irrecoverable) link errors that have occurred. The counters continue to increment until X'FFFF' is reached. |

## Set lights (LL function)

This function turns on or off any combination of the three programmable indicators on the printer. When the printers are attached to the parallel port, this function is not supported.

| CPRB Field | Content/Description |
|---|---|
| Function code | LL |
| Request DATA length | 3 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 11 |

### Request DATA values

The first three bytes of Request DATA are used to set the corresponding indicator. A ’0’ (X'30') sets the indicator off. Any other character sets it on.

## Open printer (OP function)

This function opens and acquires the financial printer server. You must use it before you request any other functions of this server. If flag2 = ’R’, subsequent operations on the read/encode magnetic stripe unit are allowed. If flag3 = ’R’, the printer is reset on a subsequent CL function request.

The printer is assigned to the application program that called it. Subsequent requests from other application programs are rejected with a *busy* return code.

To allow the shared use of this server, every application program must release the printer immediately after the execution of the desired task. This is done with the CL function.

| CPRB Field | Content/Description |
|---|---|
| Function code | OP |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 11 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 1 | 1 | Flag2 setting |
| 2 | 1 | Flag3 setting |

## Read code page (RC function)

This function allows the application program to read the code page ID and font ID currently being used. It operates in the LANDP for OS/2 and LANDP for Windows environments only.

| CPRB Field | Content/Description |
|---|---|
| Function code | RC |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 4 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 4 |
| Replied PARMLIST length | 11 |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 3 | Code page ID currently in use. Possible values (in characters) are described in the Request DATA values for “Set code page (SC function)” on page 162. |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 3 | 1 | Font ID currently in use. Possible values (in characters) are '1' through '8', as described in the Request DATA values for "Set code page (SC function)" on page 162. |

## Read data (RD function)

This function reads the data from the magnetic stripe reader attached to the IBM 4722 Model 003 Document Printer or the IBM 4772 Passbook/Document Printer, data from the MICR reader of the IBM 9069 Multi-Purpose Transaction Printer, data from the operator panel of the 4772 printer, or both sets of data on the IBM 9068-S01 Multi-Purpose Passbook Printer. The magnetic stripe data conforms to the specified format defined by function DF. If the request is for the 4772 or 9068-S01 printer operator panel, any pending timer requests, set up during the AR function, are canceled.

The RD function is always synchronous. This function can only be used for a printer attached to a serial port.

| CPRB Field | Content/Description |
|---|---|
| Function code | RD |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | Up to 105 (magnetic stripe data)<br>Up to 32 (operator panel data)<br>2-66 (magnetic ink character data) |
| Reply PARMLIST length | 26 |
| Replied DATA length | Read data length |
| Replied PARMLIST length | 11 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 1 | 1 | Flag2 setting:<br>**'O'** Read data from the operator panel keyboard of the 4772 or 9068-S01 printer<br>**'R'** Read data from the magnetic stripe reader of the 4722-003, 4772, or 9068-S01 printer<br>**'M'** Read data from the magnetic ink character reader of the IBM 9069 Multi-Purpose Transaction Printer |
| 2 | 1 | Flag3 setting: After read from the magnetic ink character reader of the IBM 9069 Multi-Purpose Transaction Printer:<br>**'E'** Eject cheque<br>**'P'** Move position for printing<br>**'R'** Resend previous data without a re-read<br>**' '** Invalid, gives the return code P1 |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 setting from previous operation |
| 7 | 1 | Flag8 setting |
| 8 | 1 | Flag9 setting |

### Reply DATA values

Reply DATA contains the data read from the magnetic stripe, the operator panel, or the MICR.

## Set redirection mode (RM function)

This function sets the parallel redirection settings. It operates in the **LANDP for DOS** environment only. It must be requested after an OP function, but redirection is not active until the following CL function has taken place.

| CPRB Field | Content/Description |
|---|---|
| Function code | RM |
| Request DATA length | 3 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 11 |

### Request DATA values

A 3-byte value representing the desired setting of the related parallel port:

- Byte 1—parallel port 1
- Byte 2—parallel port 2
- Byte 3—parallel port 3

A '0' (X'30') sets redirection off. Any other value sets the redirection status on.

### Redirection facility in LANDP for OS/2

Redirecting parallel port data to the serial attached printer is handled by a device monitor which supports both the real and protect modes of OS/2. The monitor name is PRTMON.EXE and it is part of the 4722 device support just like the device driver. See the *IBM 4700 Financial I/O Devices: Programming Guide for OS/2*.

## Set code page (SC function)

This function allows the application program to set the printer code page and font.

| CPRB Field | Content/Description |
|---|---|
| Function code | SC |
| Request DATA length | 4 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 11 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 3 | Code page ID to be set (character values). This can be any of the following values:<br><br>For the 4712 and 4722: 000, 437, 850, 851, 860, 862, 863, 864, 865, or 866<br><br>For the 4009: 000, 437, 850, or 858<br><br>For the 4772 or 9068-S01: 000, 437, 850, 851, 852, 855, 857, 860, 862, 863, 864, 865, 866, 868, 869, or 874<br><br>For the 9068-S01: 858<br><br>For the 9069: 437, 850, 851, 852, 855, 857, 858, 860, 862, 863, 864, 865, or 866 |
| 3 | 1 | Font ID to be used. This can be any one of:<br>**'0'** Default printer font (if code page ID = 0)<br>**'1'** Standard matrix font<br>**'2'** Near-letter-quality font<br>and for the 4772 or 9068-S01 printer:<br>**'3'** DP Gothic font, standard<br>**'4'** DP Gothic font, letter quality<br>**'5'** Prestige Elite, standard<br>**'6'** Prestige Elite, letter quality<br>**'7'** Courier 10 font, standard<br>**'8'** Courier 10 font, letter quality |

## Write to printer (WR function)

This function writes data from Request DATA to the printer, or to the operator panel of the IBM 4772 or IBM 9068-S01 printer.

If flag2 = 'R', then the data is encoded using the magnetic stripe encoder of the IBM 4722 Model 003 Document Printer or IBM 9068-S01 Multi-Purpose Passbook Printer.

The following table shows the maximum number of characters that can be sent to the read/encode magnetic stripe unit.

| Record Format | Redundant Records | Data Characters |
|---|---|---|
| IBM 3604/4704 | 1 | 37 |
| ISO/DIN | 1 | 45 |
| IBM 3604/4704 | 0 | 37 |
| ISO/DIN | 0 | 105 |

**Notes:**

1. The IBM 4722 Model 003 Document Printer, the IBM 4772 Passbook/Document Printer, and the IBM 9068-S01 Multi-Purpose Passbook Printer force the first data character to be an 'A' during an IBM 3604 or IBM 4704 encode operation. Therefore, the number of characters that can be specified by the user, during an encode, is one less than shown.
2. If the Set margins code is sent to a 4772 or 9068-S01 printer, the margins are relative to the left margin specified in the device parameters. They must fit the width in the device parameters—that is:

   ```
   0 <= left margin < right margin - 1
   right margin <= linelength + 1
   ```

3. Network-attached laser print support is available under Windows only. For details, read the section entitled "Network attached laser printer support" in the *Installation and Customization*, indexed under `laser printer support`.

If flag2 = 'O', then the data is written to the operator panel of the 4772 printer and the first byte of data is the start position (X'00' to X'1F') of the display field that is to be written.

| CPRB Field | Content/Description |
|---|---|
| Function code | WR |
| Request DATA length | Length of data to be written, in bytes (maximum 4096)<br><br>Up to 33 for the 4772 or 9068-S01 printer operator panel display |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 11 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 1 | 1 | Flag2 setting |
| 2 | 1 | Flag3 setting |

**Flag3** This function can be executed either asynchronously or synchronously. It is asynchronous unless you set flag3 = 'S'.

For asynchronous printing, the server initiates printing and returns control to the application program without waiting for printing to complete.

**Notes:**

1. When a synchronous request is complete, or when an asynchronous request is initiated, control returns to the application program.
2. You can check the result of the asynchronous write operation with a CH function or another WR function.
3. All control codes added by the server for formatting or tracking are not included in the print count.

### Request DATA values

Request DATA must contain the data to be printed, written on the magnetic stripe, or written to the operator panel of the 4772 or 9068-S01 printer. The data includes any control characters imbedded within the text.

**Note:** When writing in formatted mode, the server appends the carriage return and line feed control codes (X'0D0A') to the data stream, unless the last character of the data stream is already a control code.

If the data is for the operator panel of the 4772 or 9068-S01 printer, the first byte is the start position of the display field (X'00' to X'1F'). The remainder, up to 32 bytes, is the data to be written.

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 setting from previous operation |
| 1 | 1 | Flag2 setting unchanged from request |
| 2 | 1 | Flag3 setting unchanged from request |
| 7 | 1 | Flag8 setting |
| 9 | 2 | Print or encode count from present or previous operation |

## Migration considerations

These are the differences between the LANDP for DOS or FBSS (DOS) server and the LANDP for OS/2 server:

- Redirection

  The redirection facility provided in both environments (DOS and OS/2) has different implementations. The LANDP for DOS or the FBSS (DOS) server provides the RM function, because the 4712/4722 device driver in DOS supports this function. The OS/2 4712/4722 device support code provides a device monitor (PRTMON.EXE) to

implement this function. If you want to use redirection in a LANDP for OS/2 workstation, you must load and initialize the parallel port monitor (PRTMON.EXE) instead of using the RM function.

- Functions that only apply to the LANDP for OS/2 and LANDP for Windows financial printer servers

  The LANDP for OS/2 and LANDP for Windows financial printer servers provide two functions, SC (set code page) and RC (read code page), that are not provided in LANDP for DOS or FBSS (DOS). These functions are provided directly by the 4712/22 device driver configuration.

  In OS/2, the code page used by default is the one set during the installation of the operating system. If you want to change it, you can take advantage of these functions if your application can use the LANDP for OS/2 printer server only. You cannot use them if the application also uses the LANDP for DOS or the FBSS (DOS) 4712/4722 printer server.

- Number of printers connected to the serial port

  The OS/2 4712/4722 device driver supports a maximum of three printers connected to the serial port. Therefore the LANDP for OS/2 printer server can have up to three printers connected to the serial port. The LANDP for DOS, LANDP for Windows, and FBSS (DOS) printer servers each support up to four printers.

**financial printer server**

# Chapter 9. IBM 4748 printer server

This server supports the following printers:

- In the **LANDP for DOS** and **LANDP for OS/2** environments, the IBM 4748 Document Printer (all models), the IBM 9055 Model 1 Printer, and the IBM 9068-D01 Multi-Purpose Passbook Printer.
- In the **LANDP for Windows** and **LANDP for Linux** environments, the IBM 9055 Model 1 Printer and the IBM 9068-D01 Multi-Purpose Passbook Printer.

**Notes:**

1. The IBM 4748 Document Printer is ***not*** supported in either the **LANDP for Windows** or the **LANDP for Linux** environment.
2. Check the online documentation shipped with LANDP for the latest information about printer support.

This chapter provides information on how the server handles service requests. All functions operate in the **LANDP for DOS**, **LANDP for OS/2**, **LANDP for Windows**, and **LANDP for Linux** environments.

This chapter also provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields. If you need more information about the CPRB fields, see Appendix A, “Connectivity programming request block” on page 669.

*Table 29. Function Codes used in the 4748 Printer Server*

| Function code | Description | Env. | Page |
|---|---|---|---|
| CH | Check printing status | 02NL | 173 |
| CL | Close printer | 02NL | 173 |
| DF | Format parameter load | 02NL | 173 |
| EC | Get error counters | 02NL | 181 |
| LD | Load user defined character | 02NL | 182 |
| LL | Set lights | 02NL | 184 |
| OP | Open printer | 02NL | 184 |
| RD | Read data | 02NL | 185 |
| WR | Write to printer or REMS | 02NL | 185 |

The first and second columns give the function code and the name of the function. The third column shows the operating environment of the function. “02NL” means that it is available from LANDP for OS/2, LANDP for DOS, LANDP for Windows, and LANDP for Linux servers. The last column refers to the page where you can find the function described.

The 4748, 9055, and 9068-D01 printers can be used by only one application at any given time. To allow for sharing of the printer, the application should always be programmed to:

1. Acquire the printer using the OP function

© Copyright IBM Corp. 1992, 2002

2. Perform the printing operation
3. Release the printer for other applications, using the CL function

The 4748 printer server has two modes of operation:
- Formatted mode
- Unformatted mode

To use formatted mode, a DF function call must be issued to set the document characteristics such as page length, line length, writing pitch, and so on. These characteristics are entered during customization, stored in a disk file, and read when the server is loaded.

Control codes allowed in this mode are a subset of the overall control code set available for the 4748, 9055, and 9068-D01 printers. The correct page format is activated by means of the DF function, which establishes a customized format definition. This format definition can be designated to be used by only one or by both operators. The following table indicates the possible printer combinations and whether the operator A/B keys must be pressed in order to start the printer.

| **Printer Sharing and A/B Key Usage** | | |
|---|---|---|
| **DF Function Setting** | **Printer Defined As Shared** | **Printer Defined As Unshared** |
| Formatted:<br>Start key required<br>Shared using A/B keys | A/B key pressing required | Invalid |
| Formatted:<br>Start key required<br>Not shared using A/B keys | Either A or B key pressing required | Either A or B key pressing required |
| Formatted:<br>No start key required<br>Not shared using A/B keys | A/B key pressing not required | A/B key pressing not required |
| Unformatted: | A/B key pressing required | A/B key pressing not required |

In formatted mode, the 4748 printer server does not support all the control sequences that relate to page media change and page length change. The printer server does not discard these control sequences. However, using them may yield unpredictable results.

In unformatted mode, all the control codes of the 4748 printer are available. The server passes the entire data stream transparently to the printer.

After a successful OP function, the 4748, 9055, and 9068-D01 printers are available and ready to work in unformatted mode. To enter formatted mode, the format definition is loaded using the DF function. When working in formatted mode you can switch to unformatted mode by issuing a DF function call with value 0 in the data length field. During customization the working mode is determined; this can be shared or unshared printer mode, and shared or unshared formatted page mode. This provides for either automatic activation of the printer or the requirement for A or B key pressing to start printing. Refer to the previous table.

## Using the A/B keys for printer sharing

To use the A/B keys for printer sharing between two workstations, you must determine during customization which A/B key will be used for each workstation.

The following is a sample application that uses A/B keys for printer sharing:

```
Begin
  OP    /* opens and acquires the 4748 printer server */
   :
  DF with 'Start key required' and 'Shared using A/B keys'
   :
  (set data to write)
  printf ("Insert paper and press Start key");
  do {
     WR with synchronous mode           /* writes data to 4748 */
     } while (return_code == 'IR')      /* Intervention required*/
   :k
   :k
End
```

## Considerations for IBM DOS H7.0 and IBM OS/2 Warp H3.0

IBM DOS H7.0 and IBM OS/2 Warp H3.0 support the IBM Korean Standard (KS) code (code page 949). However, the font file in the IBM 4748 Document Printer for Korea is based on the previous PC Code standard, hereafter PC code. The 4748 printer server converts DBCS IBM KS codes into PC codes automatically.

The conversion uses the following rules:

- IBM KS code range goes from X'8FA1' to X'FEFE'.
- PC code range goes from X'8140' to X'FCFC' (X'nn7F' is excluded).
- If the given DBCS code is in the IBM KS code range and is defined as a valid character, it is converted to the PC code that relates to that character. Otherwise, no conversion takes place.
- The IBM KS code in the user-defined character (UDC) range is converted to the related PC code as the following table shows.

| IBM KS code UDC range | PC code UDC range |
|---|---|
| X'C9A1'–X'C9FE'→ | X'B040'–X'B09E' (excludes X'B07F') |
| X'FEA1'–X'FEFE'→ | X'B09F'–X'B0FC' |
| X'8FA1'–X'A0FE'→ | X'B140'–X'B9FC' (excludes X'nn7F') |

## User-defined character (UDC) support

The server allows you to download UDC images by using either:

- An LD function request
- A WR function request that includes a character code corresponding to a UDC font defined on DOS/V or OS/2 (automatic UDC font download)

**LD function**: You can download UDC font images dynamically with this function. An application can download two types of UDC image to the printer:

- Single width (supported only in unformatted mode, and not supported on a 4748 printer)
- Double width

**Automatic UDC font download function**: The server tries to download dynamically the UDC font defined on DOS/V or OS/2. It does this when it detects that a DBCS code included in the print data stream of a WR function request corresponds to a UDC code currently defined on DOS/V or OS/2.

**Note:** Automatic UDC font download is supported only on the Taiwanese versions of DOS/V, OS/2, and PRC DOS/V.

**Download after device power-on/reset (POR)**: If the load statement for the server specifies a UDC file, the user-defined character is downloaded dynamically when an application requests the printing of a character code that is included in the UDC file. Additional UDC images downloaded using the LD function are appended to the UDC file.

Even if a UDC file is not specified in the server load statement, UDC images downloaded using the LD function are kept in the server's default UDC file (PR4748.UDC).

**Note:** This default UDC file is always erased and recreated at server loading time. To save and reuse its contents, you can copy PR4748.UDC to another file and specify this file in the server load statement, using the /A parameter. For example:

```
LOADER  PR4748##.EXE  /A:D:\MY_DIR\MY_FILE.UDC
```

Even if the downloaded UDCs are erased from the printer device because it is powered off and reset, subsequent print requests will cause the server to download the UDC image dynamically if the requested character code is included in the UDC file.

The maximum number of UDCs that can be downloaded and registered in the UDC file is 200 for the 9055 Model 1 printer, and 71 for the 4748 printer. If more UDCs are requested, or are required in order to process the print data stream, the request is rejected as a translation check error. You can recover from this by erasing the UDC file, and powering the printer off and on.

## Using the 4748 printer server

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | 26 | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 20 | 2 | | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 26 | 2 | 26 | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | **PR4748##** | Server name |
| **Note:** During customization, a value is defined for ## relating it to a serial port to which the 4748, 9055, or 9068-D01 printer is attached. | | | |

The following fields are variable and are discussed in each function request description:
- Function code
- Request DATA length
- Reply DATA length

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 44 | 2 | | Replied PARMLIST length |
| 46 | 2 | | Replied DATA length |

If the request was successful, the *router return code* and the *server return code* are both X'00000000'. In all other cases, see the appropriate section in the *LANDP Problem Determination* book to see if you should take any action. The return values in Reply PARMLIST and DATA should be ignored if there is an error.

The following fields are variable and are discussed in each function request description:
- Replied PARMLIST length
- Replied DATA length

**PARMLIST:** The Request and Reply PARMLIST fields for the 4748 printer server are:

| Offset | Length | Content |
|---|---|---|
| 0 | 1 | Flag1 |
| 1 | 1 | Reserved |
| 2 | 1 | Flag3 |
| 3 | 1 | Flag4 |
| 4 | 3 | Reserved |

| Offset | Length | Content |
|---|---|---|
| 7 | 1 | Flag8 |
| 8 | 1 | Reserved |
| 9 | 2 | Number of printed characters. It contains the number of application data-stream characters that have been printed. Control characters are not counted. |

**DATA:** Request and Reply DATA contain data to be sent to, or received from, the server. The use of these areas is explained in the description of each function request.

## IBM 4748 printer server flag descriptions

Flag1 (returned by the server or set by the application)

**'P'** The return code corresponds to the previous operation.

**'D'** This is used in the DF function only. If this flag is 'D', Request DATA contains a format description. Any other value shows that Request DATA contains a format description name.

Flag3 (set by the application)

**'S'** By default, the WR function is asynchronous. The flag value 'S' makes it synchronous.

Flag4 (set by the application)

**'G'** By default, the Request DATA of the WR function includes the print data stream to be printed. With the flag value 'G' the function includes both the attribute grid count/data and the print count/data to be printed.

Flag8 (returned by the server)

**'A'** Active interface

**'B'** Active interface

If a form has been inserted, the printer starts the interfaces as follows:

Interface A is started.
When processing passbook or document forms and key A is pressed.

Interface B is started.
When processing passbook or document forms and key B is pressed.

The printer only prints when you do one of the following:

- You select "auto-start" in the format definition.
- You have to press a key, and either interface is active.
- You select shared mode, and the correct interface is active.

All the functions are reflected in the values of flag8 covering all the possibilities that can occur in setting the active interface.

## Request reference

This section gives you the programming information required to write applications, or your own servers, that issue service requests to the 4748 printer server. The function codes are listed in alphabetical order.

## Check printing status (CH function)

This function requests the return code from the last operation on the printer. Use this function to check the completion of a write operation with an asynchronous WR function.

| CPRB Field | Content/Description |
|---|---|
| Function code | CH |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 11 |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 9 | 2 | Print count from previous operation |

## Close printer (CL function)

This function closes the 4748 printer server and releases it for use by another application. Use this function, when your application is finished with the printer, to free it for use by another application.

| CPRB Field | Content/Description |
|---|---|
| Function code | CL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

## Format parameter load (DF function)

This function starts 4748 printer server formatting mode and begins page formatting with the customized format defined in Request DATA. If the Request DATA length field is zero, no formatting is done and unformatted mode is set.

For reader/encoder magnetic stripe (REMS) operation on an IBM 9055 Model 1 or IBM 9068-D01 printer, the DF function for REMS is required. Otherwise, functions for REMS operation are rejected with a return code of P7.

The server can store two different formats:
- For the print component; Flag2 ≠ 'R'
- For the reader/encoder magnetic stripe (REMS) component; Flag2 = 'R'

| CPRB Field | Content/Description |
|---|---|
| Function code | DF |
| Request DATA length | **0** Unformatted mode<br>**8** Name of format<br>**x** Length of format definition data |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 11 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 setting |

## Request DATA values

**Flag1 ≠ 'D'**

Request DATA must contain the name of the desired format as assigned during customization. The Request DATA length field contains X'0008'.

**Flag1 = 'D'**

Request DATA must contain the format definition. The information in the following tables will usually be sufficient, but for more information see the *IBM 4748 Document Printer Programming Guide*, the *IBM 9055-001 Document Printer: Planning and Programming Guide.* or the *IBM 9068-D01 Multi-Purpose Passbook Printer: Planning and Programming Guide.*

## Format definition for printer

| Byte | Passbook | Cutforms |
|---|---|---|
| 0 | X'00' | X'01' |
| 1 | Page size in lines | Page size in lines |
| 2 | Centerfold begin column/line number | Attention line number |
| 3 | Centerfold skip in number lines/chars | Step offset in number of steps |
| 4 | Step offset in number of steps | Line offset in number of lines |
| 5 | Line offset in number of lines | Line length |

| Byte | Passbook | Cutforms |
|---|---|---|
| 6 | Left margin column number:<br>Maximum value is:<br>maximum line length - 0.5 inch | Device bits:<br>**7:** not used<br>**6:**<br>0=ignore byte 8<br>1=use byte 8<br>**5:**<br>0=not shared<br>1=shared<br>**4:**<br>0=start switch<br>1=autostart mode<br>**3:**<br>0=ignore byte 7<br>1=use byte 7<br>**2:**<br>0=line length check<br>1=auto new line<br>**1:**<br>0=5 lpi<br>1=6 lpi<br>**0:**<br>0=10 cpi<br>1=12 cpi |

| Byte | Passbook | Cutforms |
|---|---|---|
| 7 | Line length | Only valid if bit 3 of byte 6 is 1. Bits:<br>**7:**<br>0=normal print mode<br>1=high-speed print mode<br>**6:** not used<br>**5:**<br>0=no double strike<br>1=double strike<br>**4:**<br>0=not emphasized<br>1=emphasized<br>**3, 2:**<br>extended character pitch:<br>00=use byte 6, bit 0<br>01=13.4 cpi<br>10=15 cpi<br>11=reserved<br>**1, 0:** acceptable skew:<br>**For 9055 printer:**<br>00=1.46 mm (0.06 in)<br>01=0.73 mm (0.03 in)<br>10=2.19 mm (0.09 in)<br>11=2.92 mm (0.11 in)<br>**For 4748 printer:**<br>00=1.37 mm (0.05 in)<br>01=0.68 mm (0.03 in)<br>10=2.05 mm (0.08 in)<br>11=2.74 mm (0.11 in<br>**For 9068 printer:**<br>00=1.61 mm (0.06 in)<br>01=0.76 mm (0.03 in)<br>10=2.27 mm (0.09 in)<br>11=3.02 mm (0.12 in) |

| Byte | Passbook | Cutforms |
|---|---|---|
| 8 | Device bits:<br>**7:**<br>0=horizontal fold<br>1=vertical fold<br>**6:**<br>0=ignore byte 10<br>1=use byte 10 extension<br>**5:**<br>0=not shared<br>1=shared<br>**4:**<br>0=start switch<br>1=autostart mode<br>**3:**<br>0=invalid byte 9<br>1=valid byte 9 extension<br>**2:**<br>0=line length check<br>1=auto new line<br>**1:**<br>0=5 lpi<br>1=6 lpi<br>**0:**<br>0=10 cpi<br>1=12 cpi | Only valid if bit 6, byte 6 is 1. Bits:<br>**7, 6:** not used<br>**5:**<br>0=bottom edge validation off<br>1=bottom edge validation on<br>**4:** not used<br>**3:**<br>0=use byte 6, bit 1<br>1=8 lpi<br>**2–0:**<br>print font select:<br>000=default font (Mincho-12)<br>001=DP font-10<br>010=Prestige Elite-12<br>011=Courier-10<br>100=Mincho-12<br>101=Mincho-10<br>110, 111=reserved |

| Byte | Passbook | Cutforms |
|---|---|---|
| 9 | Only valid if bit 3 of byte 8 is 1. Bits:<br>**7:**<br>0=no high speed print mode<br>1=high-speed print mode (effective for 9055 Model 1 printer only)<br>**6:** not used<br>**5:**<br>0=no double strike<br>1=double strike<br>**4:** not used<br>**3, 2:**<br>extended character pitch:<br>00=use byte 8, bit 0 to set character pitch<br>01=13.3 cpi<br>10=15 cpi<br>11=reserved<br>**1, 0:**<br>acceptable skew:<br>For 9055 printer:<br>00=1.46 mm (0.06 in)<br>01=0.73 mm (0.03 in)<br>10=2.19 mm (0.08 in)<br>11=reserved<br>For 4748 printer:<br>00=1.37 mm (0.05 in)<br>01=0.68 mm (0.03 in)<br>10=2.05 mm (0.08 in)<br>11=reserved<br>For 9068 printer:<br>00=1.61 mm (0.06 in)<br>01=0.76 mm (0.03 in)<br>10=2.27 mm (0.09 in)<br>11=reserved | Horizontal character offset<br>Maximum value is:<br>maximum line length - 0.5 inch |

| Byte | Passbook | Cutforms |
|---|---|---|
| 10 | Only valid if bit 6 of byte 8 is 1. Bits:<br>**7–4:** not used<br>**3:**<br>0=use byte 8, bit 1<br>1=8 LPI (for 9055 Model 1 printer)<br>1=reserved (for 4748 printer)<br>**2–0:**<br>print font select:<br>000=default font (Mincho-12)<br>001=DP font-10<br>010=Prestige Elite-12<br>011=Courier-10<br>100=Mincho-12<br>101=Mincho-10<br>110, 111=reserved | Reserved |
| **Note:** The next two bytes are defined for the 9055 Model 1 printer only | | |
| 11 | Centerfold skip add step | Reserved |
| 12 | Reserved | Reserved |

## Format definition for 9055 Model 1 and 9068-D01 Printer with REMS

| Offset | Length | Content/Description |
|---|---|---|
| 0 | 1 | X'00' |
| 1 | 1 | Device bits<br>**7:**<br>0=not displaced<br>1=displaced<br>**6:**<br>0=double recording<br>1=single recording<br>**5:** unused<br>**4:**<br>0=no check for new passbook<br>1=check new passbook (document interlock)<br>**3–0: X'4' = IBM 3604/4704**<br>X'D' = ISO/DIN |
| 2 | 1 | Book width in millimetres (120–213) |

| Offset | Length | Content/Description |
|---|---|---|
| 3 | 1 | Initial positioning after passbook insertion:<br>**X'00'** first print line<br>**X'01'** REMS position (magnetic position is determined by the settings in byte 1) |

**Application programming notes:**

1. If formatted mode is selected, it remains in effect until cancelled, either by a DF command with the data length field set to zero, or by a CL (close) function call. After a subsequent OP (open), the server starts in unformatted mode.
2. Values from the latest DF request for each component are effective.
3. With a print operation, you can print data in either formatted or unformatted mode.
4. With REMS operation, you can print data only in formatted mode.
5. When printing in unformatted mode, the application itself must control media movements, parameter loading and printing.
6. If 'check new passbook (document interlock)' is specified on a DF for REMS, the server sets an interlock bit OFF for RD from REMS operation and ON for WR to REMS. By this means, the application can ensure that both an RD from REMS and a WR to REMS operation are performed on the same document. If the document is ejected after an RD from REMS, the WR to REMS operation is discarded and an error status is returned.
7. When a DF for REMS is issued, the server sets some REMS parameters based on the format data available for the print component. This can be either the defaults set when the server was initially loaded, or those of the most recent DF for the print component, which will override the defaults.
8. A DF for an unavailable component is rejected (return code `P4`).
9. Currently inserted media may be ejected if a DF does any of the following:
   - Tightens the acceptable skew value
   - Changes the book width value
   - Changes the initial position after insertion
   - Changes the passbook fold direction
   - Changes the form type

   The DF itself is accepted (return code `OK`).
10. When both printer and REMS components are available, the DF combinations are as follows:

| Printer | REMS | See: |
|---|---|---|
| Formatted | Formatted | Note 10a on page 181 |
| Unformatted | Formatted | Note 10b on page 181 |
| Formatted | Unformatted | Note 10c on page 181 |
| Unformatted | Unformatted | Note 10d on page 181 |

**Notes:**

a. This is the recommended combination. Print data is formatted according to the DF specification. Your application can request REMS operation.

b. Print data is passed through to the 9055 or 9068-D01 device without formatting. The application itself must control print formatting, media movements, parameter loading, and printing. If the print medium had been brought to the Magnetic Stripe position, but print operations are now required, the application must first issue a Media Positioning Message command (see the documentation shipped with your printer).

   If the ″Initial positioning after passbook insertion″ offset in DF for REMS is set to ″REMS position″, the server sets 1.46 mm as the maximum acceptable skew value.

c. Print data is formatted according to the DF specification. REMS operation is not possible.

d. Print data is passed through to the 9055 or 9068-D01 device without formatting. The application itself must control print formatting, media movements, parameter loading, and printing. REMS operation is not possible.

## Get error counters (EC function)

This function retrieves error counters, interface setting, printer engineering change (EC) level data, printer ID data, and printer personalization data.

| CPRB Field | Content/Description |
|---|---|
| Function code | EC |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 12 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 12 |
| Replied PARMLIST length | 11 |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Error counters. This represents the number of soft errors that have occurred (successfully retried). The counters continue to increment until X'FFFF' is reached. |
| 2 | 2 | Error counters. This represents the number of hard (irrecoverable) errors that have occurred. The counters continue to increment until X'FFFF' is reached. |
| 4 | 2 | Power on reset (POR) status bytes 1 and 2. These bytes contain the printer feature bytes as presented to the device driver at POR. |
| 6 | 1 | Printer engineering change (EC) level data. This byte contains the EC level of the control read only storage in the printer. |
| 7 | 1 | Printer identification data. This byte contains the identification value X'1D' for the 4748, 9055, or 9068-D01 printer. |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 8 | 1 | Printer personalization data. This byte contains the personalization switch settings of the 4748 printer. The personality setup switch positions and meanings are defined in the *IBM 4748 Document Printer Programming Guide.* The function reports the switch settings as follows:<br>SWITCH: 8 7 6 5 4 3 2 1<br>BIT: 7 6 5 4 3 2 1 0<br>A bit set to 1 indicates that the associated switch is in the ON position. |
| 9 | 3 | Reserved |

## Load user-defined character (LD function)

This function loads the user-defined character (Gaiji) to a 9068-D01, 9055 Model 1, or 4748 printer. You can load a maximum of 200 user-defined characters to the 9055 printer, and 71 to the 4748 printer. The loaded font is reset when the printer is powered off or the server is unloaded.

| CPRB Field | Content/Description |
|---|---|
| Function code | LD |
| Request DATA length | 40 for single-width character<br>76 for double-width character |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 11 |

Request DATA contains character code, character type, character image data byte count, and character image data.

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Character Code (not Intel format): it ranges from X'8140' to X'FCFC' for Taiwan and the People's Republic of China, and from X'8FA1' to X'FEFE' for Korea. X'8140' (DBCS space) cannot be used for Taiwan and the People's Republic of China. X'A1A1' (DBCS space) cannot be used for Korea. |
| 2 | 1 | Character type flag:<br>X'0000' for single width<br>X'0002' for double width |
| 3 | 1 | Number of character image data bytes: X'0018' for 24 wires. |
| 4 | 36 or 72 | Character image data. The length is 36 for single-width characters, or 72 for double-width characters. |

**Note:** Your application must ensure that the code points that it passes to the server are valid. This will depend on whether or not translation from BIG 5 to Taiwan PC is required, and also on what the operating system will allow to be registered as a user-font. You might get unexpected results if the code points are not valid.

## Character image data

The following table shows a single-width character image format:

```
ADGJMPSVYbeh
ADGJMPSVYbeh
ADGJMPSVYbeh
ADGJMPSVYbeh
ADGJMPSVYbeh
ADGJMPSVYbeh
ADGJMPSVYbeh
ADGJMPSVYbeh
BEHKNQTWZcfi
BEHKNQTWZcfi
BEHKNQTWZcfi
BEHKNQTWZcfi
BEHKNQTWZcfi
BEHKNQTWZcfi
BEHKNQTWZcfi
BEHKNQTWZcfi
CFILORUXadgj
CFILORUXadgj
CFILORUXadgj
CFILORUXadgj
CFILORUXadgj
CFILORUXadgj
CFILORUXadgj
CFILORUXadgj

 byte0   byte1   byte2                 byte34  byte35
AAAAAAAABBBBBBBBCCCCCCCC.............iiiiiiiijjjjjjjj
```

The following table shows the double-width character image format (24 x 24 dot matrix):

```
ADGJMPSVYbehknqtwz258"%(
ADGJMPSVYbehknqtwz258"%(
ADGJMPSVYbehknqtwz258"%(
ADGJMPSVYbehknqtwz258"%(
ADGJMPSVYbehknqtwz258"%(
ADGJMPSVYbehknqtwz258"%(
ADGJMPSVYbehknqtwz258"%(
ADGJMPSVYbehknqtwz258"%(
BEHKNQTWZcfilorux0369*-)
BEHKNQTWZcfilorux0369*-)
BEHKNQTWZcfilorux0369*-)
BEHKNQTWZcfilorux0369*-)
BEHKNQTWZcfilorux0369*-)
BEHKNQTWZcfilorux0369*-)
BEHKNQTWZcfilorux0369*-)
BEHKNQTWZcfilorux0369*-)
CFILORUXadgjmpsvy147!+'=
```

```
CFILORUXadgjmpsvy147!+'=
CFILORUXadgjmpsvy147!+'=
CFILORUXadgjmpsvy147!+'=
CFILORUXadgjmpsvy147!+'=
CFILORUXadgjmpsvy147!+'=
CFILORUXadgjmpsvy147!+'=
CFILORUXadgjmpsvy147!+'=

 byte0   byte1   byte2                 byte70  byte71
AAAAAAAABBBBBBBBCCCCCCCC ............ ))))))))========
```

## Set lights (LL function)

This function turns on or off any combination of the three programmable indicators on the 4748, 9055, or 9068-D01 printer.

| CPRB Field | Content/Description |
|---|---|
| Function code | LL |
| Request DATA length | 3 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 11 |

### Request DATA values

The first three bytes of Request DATA are used to set the corresponding indicator. A ’0’ (X'30') sets the indicator off. Any other character sets it on. The third indicator corresponds to the insert document indicator.

## Open printer (OP function)

This function opens and acquires the 4748 printer server. You must use it before you request any other functions of this server.

The printer is assigned to the application that called it. Subsequent requests from other applications are rejected with a *busy* return code.

To allow the shared use of this server, every application program must release the printer immediately after the execution of the desired task. This is done with the CL function.

| CPRB Field | Content/Description |
|---|---|
| Function code | OP |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |

| CPRB Field | Content/Description |
|---|---|
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

## Read data (RD function)

This function reads the data from the magnetic stripe reader attached to the IBM 9055 Model 1 Passbook Printer or IBM 9068-D01 Multi-Purpose Passbook Printer with REMS. The data conforms to the specified format defined by the DF function. The RD function is always synchronous.

| CPRB Field | Content/Description |
|---|---|
| Function code | RD |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | Sufficient to save the data read |
| Reply PARMLIST length | 26 |
| Replied DATA length | Read data length |
| Replied PARMLIST length | 11 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 1 | 1 | Flag2 setting: 'R' |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 setting from previous operation |
| 7 | 1 | Flag8 setting |

### Reply DATA values

After a successful read, Reply DATA contains the data read from the magnetic stripe.

**Application programming note:** If the passbook is not in the REMS position, the server moves it to the magnetic position specified by the DF function call.

## Write to printer (WR function)

This function writes data from Request DATA to a 4748 printer, or a 9055 Model 1 Passbook Printer with REMS. If Flag2='R', the data is encoded using the magnetic stripe encoder of the 9055 or 9068-D01 printer. The function can also be used to send code to the REMS component.

**Note:** For the 4748 printer, a Flag2='R' request is ignored.

Using this function, your application can:
- Print data
- Encode magnetic stripe data
- Send a control code to the print component

The maximum number of characters that you can send to the REMS component is as follows:

| Record Format | Redundant Records | Data Characters |
|---|---|---|
| IBM 3604/4704 | 1 | 37 |
| ISO/DIN | 1 | 45 |
| IBM 3604/4704 | 0 | 37 |
| ISO/DIN | 0 | 105 |
| **Note:** The IBM 9055 Model 1 Passbook Printer, or IBM 9068-D01 Multi-Purpose Passbook Printer with REMS, forces the first data character to be an X'0A' during an IBM 3604/4704 encode operation. Therefore, the number of characters that can be specified by the user for encoding is one less than shown. | | |

| CPRB Field | Content/Description |
|---|---|
| Function code | WR |
| Request DATA length | Length of data to be written, in bytes |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 11 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 1 | 1 | Flag2 setting. 'R' for encoding magnetic stripe, others for printing |
| 2 | 1 | Flag3 setting. The function executes asynchronously unless Flag3='S'. In asynchronous mode, the server initiates printing but returns control to the application without waiting for printing to complete |
| 3 | 1 | Flag4 setting. If Flag4='G', DATA contains grid attribute data and print character data. Otherwise it contains the print data stream |
| 7 | 1 | Reserved |

**Notes:**

1. When a synchronous call has completed, or when an asynchronous call has been initiated, control returns to your application. The server returns data, return code and other information in the CPRB and Reply PARMLIST.

2. You can check the result of an asynchronous write operation by issuing either a CH call or another WR call.
3. Control codes added by the server for formatting or tracking are not included in the print count.

## Request DATA values

If the data string contains DBCS characters in the user-defined character (UDC) range, this function automatically uses the LD function to down-load the UDC image font to the printer, if it has not already done so. This automatic down-loading function is supported in the DOS T7.0/V, DOS P7.0/V, and OS/2 T3.0 environments.

If Flag4 is not 'G', Request DATA must contain the data to be printed or written on the magnetic stripe. The data includes any control characters imbedded within the text.

If Flag4 is 'G', Request DATA must contain both the grid attribute data and the data to be printed. The format is:

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | n/2 | Grid attribute data<br><br>The format of the attribute data entry is as follows:<br>`Bits 7 6 5 4 3 2 1 0`<br>`  \|◄-- H. grid --►\|◄-- V. grid --►\|`<br>`Bits 7-4: Horizontal Grid Line`<br>`  0000=none`<br>`  0001=single solid line`<br>`  0010=heavy solid line`<br>`  0011=single dotted line`<br>`Bits 3-0: Vertical Grid Line`<br>`  The values are the same as those`<br>`  for the Horizontal Grid Line` |
| n/2 | n/2 | Print character data |

where n is the Request DATA length.

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 setting from the previous operation |
| 7 | 1 | Flag 8 setting |
| 9 | 2 | Print count from this, or the previous, operation |

**Notes:**

1. When writing in formatted mode, the server appends the carriage return and line feed control codes (X'0D0A') to the data stream, unless the last character of the data stream is already a control code.
2. Each byte in the grid attribute data for a WR call with Flag4='G' is directly associated with the corresponding byte in the character buffer. The same rule

applies to DBCS printing. If you specify a vertical grid line to be printed in the middle of a DBCS character, the result is unpredictable.

3. If you specify an odd number as the Request DATA length for a WR call with Flag4='G', the request is rejected as a data length error (return code P2).
4. If you try to print grid data on the first or last line of a form, an end of document exception occurs (return code I7).
5. Writing grid data is supported on a per line basis. Each print line requires a separate WR function with Flag4 set to 'G'. If you specify more than one line on such a function, the attribute data will be ignored for each line except the first, output for which will be unpredictable.

**Application programming notes:**

**Notes:**

1. In formatted mode, if a passbook is present but not in the printer position when printing is requested, the server moves the passbook to the first print position before starting to print. However, your application should use control codes in the Request DATA to specify the position at which you want actual printing to start.
2. Your application must ensure that the code points it passes to the server are valid. This will depend on whether or not translation from BIG 5 to Taiwan PC is required, and also on what the operating system will allow to be registered as a user-font. You may get unexpected results if the code points are not valid.

# Chapter 10. IBM 4770 printer server

This server processes requests for the IBM 4770 Ink Jet Transaction Printer. All functions operate in the **LANDP for OS/2** environment.

When a 4770 printer is attached to a workstation, no other printer can be attached at the same time.

This chapter provides information on how the server handles service requests. It also provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields.

If you need more information about the CPRB fields, see Appendix A, “Connectivity programming request block” on page 669.

*Table 30. Function Codes used in the 4770 Printer Server*

| Function code | Description | Env. | Page |
|---|---|---|---|
| CH | Check printing status | -2-- | 192 |
| CL | Close printer | -2-- | 192 |
| DF | Format parameter load | -2-- | 192 |
| EC | Get error counters | -2-- | 194 |
| OP | Open printer | -2-- | 195 |
| WR | Write to printer | -2-- | 196 |

The first and second columns give the function code and the name of the function. The third column shows the operating environment of the function. “-2--” means that it is available from LANDP for OS/2 servers. The last column refers to the page where you can find the function described.

The 4770 printer server has two modes of operation:
- Formatted mode
- ASCII unformatted mode

To use formatted mode, a DF function call must be issued to set the document characteristics such as page length, line length, writing pitch, and so on. These characteristics are entered during customization, stored in a disk file, and read when the server is loaded.

Control codes allowed in this mode are a subset of the overall control code set that is available for the printers. The correct page format is activated by means of the DF function, which establishes a customized format definition. This format definition can be designated to be used by one or both of the operators.

© Copyright IBM Corp. 1992, 2002

In formatted mode, the 4770 printer server does not support all the control sequences that relate to page media change and page length change. The printer server does not discard these control sequences. However, using them might yield unpredictable results.

In unformatted mode, all the control codes of the printer are available. The server passes the entire data stream transparently to the printer.

After a successful OP function, the printer is available and ready to work in unformatted mode. To enter formatted mode, use the DF function to load the format definition. When working in formatted mode, you can switch to unformatted mode by issuing a DF function call with value 0 in the Request DATA length field. During customization, the working mode is determined; shared or unshared printer mode and shared or unshared formatted page mode. This provides for automatic activation of the printer.

## Using the 4770 printer server

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | 26 | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | 26 | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | PR4770## | Server name |
| **Note:** During customization, a value is defined for ## relating it to the serial or parallel port that the printer is attached to. | | | |

The following fields are variable and are discussed in each function request description:
- Function code
- Request PARMLIST length
- Request DATA length
- Reply PARMLIST length
- Reply DATA length

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 44 | 2 | | Replied PARMLIST length |

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 46 | 2 | | Replied DATA length |

If the request was successful, the *router return code* and the *server return code* are both X'00000000'. In all other cases, refer to the appropriate section in the *LANDP Programming Guide* book to see if you should take any action. The return values in Reply PARMLIST and DATA should be ignored if there is an error.

The following fields are variable and are discussed in each function request description:
- Replied PARMLIST length
- Replied DATA length

**PARMLIST:** The Request PARMLIST and Reply PARMLIST fields for the 4770 printer server are:

| Offset | Length | Content |
|---|---|---|
| 0 | 1 | Flag1 |
| 1 | 1 | Reserved |
| 2 | 1 | Flag3 |
| 3 | 6 | Reserved |
| 9 | 2 | Number of written or encoded characters. It contains the number of characters in the application program's data stream that have been printed or encoded. Control characters are not counted. |

**DATA:** Request DATA and Reply DATA contain data to be sent to, or received from, the server. The use of these areas is explained in the description of each function request.

## 4770 printer server flag descriptions

Flag1 (returned by the server or set by the application program)

**'D'** In the DF function, if this flag is 'D', Request DATA contains a format description. Any other value shows that Request DATA contains a format description name.

**'P'** The return code corresponds to the previous operation.

Flag3 (set by the application program)

**'S'** By default, the WR function is asynchronous. The flag value 'S' makes it synchronous.

## Request reference

This section gives you the programming information required to write application programs, or your own servers, that issue service requests to the 4770 printer server. The function codes are listed in alphabetical order.

## Check printing status (CH function)

This function requests the return code from the last operation on the printer.

Use this function to check the completion of an asynchronous WR function.

| CPRB Field | Content/Description |
|---|---|
| Function code | CH |
| Request DATA length | 0 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 11 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 11 |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 9 | 2 | Print count from previous operation |

## Close printer (CL function)

This function closes the 4770 printer server and releases it for use by another application program. Use this function when your application program is finished with the printer to free it for use by another application program.

| CPRB Field | Content/Description |
|---|---|
| Function code | CL |
| Request DATA length | 0 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 11 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 11 |

## Format parameter load (DF function)

This function starts the 4770 printer server formatting mode and begins page formatting with the customized format defined in Request DATA. If the Request DATA length field is zero, no formatting is done and unformatted mode is set.

This function can only be used for a printer that is attached to a serial port.

| CPRB Field | Content/Description |
|---|---|
| Function code | DF |
| Request DATA length | **0** Unformatted mode<br>**8** Name of format (flag1 not 'D')<br>**13** Format definition (flag1 = 'D') |
| Request PARMLIST length | 2 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 setting |

## Request DATA values

**Flag1 ≠ 'D'** Request DATA must contain the name of the desired format as assigned during customization. The Request DATA length field contains X'0008'.

**Flag1 = 'D'** Request DATA must contain the format definition. In most cases the information provided in the following table should be sufficient.

## Format definition for 4770 printer

| Byte | Journal (roll paper) | Cut-forms |
|---|---|---|
| 0 | X'F2' | X'F1' |
| 1 | Reserved | Page size in lines |
| 2 | Reserved | Attention line number |
| 3 | Line length | Step offset in number of steps |
| 4 | Reserved | Line offset in number of lines |
| 5 | Reserved | Line length |
| 6 | Reserved | Reserved |
| 7 | Reserved | Reserved |
| 8 | Reserved | Reserved |
| 9 | Reserved | Horizontal character offset |
| 10 | Reserved | Reserved |

| Byte | Journal (roll paper) | Cut-forms |
|---|---|---|
| 11 | Device bits:<br>**7:**<br>0=single width<br>1=double width<br>**6, 5:** reserved (set to 0)<br>**4:**<br>0=line length check<br>1=auto new line<br>**3:**<br>0=text spacing mode<br>1=logo spacing mode<br>**2:**<br>0=normal mode<br>1=upside-down mode<br>**1, 0:**<br>font:<br>0=standard font<br>1=large font<br>2=large bold font | Device bits:<br>**7:**<br>0=single width<br>1=double width<br>**6, 5:** reserved (set to 0)<br>**4:**<br>0=line length check<br>1=auto new line<br>**3:**<br>0=text spacing mode<br>1=logo spacing mode<br>**2:**<br>0=normal mode<br>1=upside-down mode<br>**1, 0:**<br>font:<br>0=standard font<br>1=large font<br>2=large bold font |
| 12 | Reserved | Reserved |

## Get error counters (EC function)

This function retrieves error counters, interface setting, printer engineering change (EC) level data, printer ID data, printer personalization data, and redirection setting.

| CPRB Field | Content/Description |
|---|---|
| Function code | EC |
| Request DATA length | 0 |
| Request PARMLIST length | 0 |
| Reply DATA length | 23 |
| Reply PARMLIST length | 11 |
| Replied DATA length | 19 |
| Replied PARMLIST length | 11 |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Error counters. This represents the number of soft errors that have occurred (successfully retried). The counters continue to increment until X'FFFF' is reached. |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 2 | Error counters. This represents the number of hard (irrecoverable) errors that have occurred. The counters continue to increment until X'FFFF' is reached. |
| 4 | 2 | Power on reset (POR) status bytes 1 and 2. These bytes contain the printer feature bytes as presented to the device driver at POR. |
| 6 | 1 | Printer engineering change (EC) level data. This byte contains the EC level of the control read only storage in the printer. |
| 7 | 1 | Printer identification data. This byte contains the identification value for the printer:<br>X'19' for 4770 Ink Jet Transaction Printer |
| 8 | 11 | Not used |
| 19 | 2 | Printer link error counters. This represents the number of soft errors (errors that have been successfully retried) that have occurred. The counters continue to increment until X'FFFF' is reached. |
| 21 | 2 | Printer link error counters. This represents the number of hard (irrecoverable) errors that have occurred. The counters continue to increment until X'FFFF' is reached. |

## Open printer (OP function)

This function opens and acquires the 4770 printer server. You must use it before you request any other functions of this server.

The printer is assigned to the application program that called it. Subsequent requests from other application programs are rejected with a *busy* return code.

To allow the shared use of this server, every application program must release the printer immediately after the execution of the desired task. This is done with the CL function.

| **CPRB Field** | **Content/Description** |
|---|---|
| Function code | OP |
| Request DATA length | 0 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 11 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 11 |

## Write to printer (WR function)

This function writes data from Request DATA to the printer.

| CPRB Field | Content/Description |
|---|---|
| Function code | WR |
| Request DATA length | Length of data to be written, in bytes (maximum 4096) |
| Request PARMLIST length | 3 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 11 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 11 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 1 | Flag3 setting |

**Flag3** This function can be executed either asynchronously or synchronously. It is asynchronous unless you set flag3 = 'S'.

For asynchronous printing, the server initiates printing and returns control to the application program without waiting for printing to complete.

**Notes:**

1. When a synchronous request is complete, or when an asynchronous request has been initiated, control returns to the application program.
2. You can check the result of the asynchronous write operation using either a CH function or another WR function.
3. All control codes added by the server for formatting or tracking are not included in the print count.
4. A difference between the 4770 and the 47x2 printer is that the out-of-paper light may flash when a cut-form print request is pending. If you insert the form, the print request should be satisfied and the data prints. (With the 47x2, for successful printing, the form has to be correctly inserted **before** the WR command is issued.)

### Request DATA values

Request DATA must contain the data to be printed. The data includes any control characters imbedded within the text.

**Note:** When writing in formatted mode, the server appends the carriage return and line feed control codes (X'0D0A') to the data stream, unless the last character of the data stream is already a control character.

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 setting from previous operation |
| 9 | 2 | Print count from this or the previous operation |

# Chapter 11. Printer manager server

The **LANDP for DOS** printer manager server offers a solution to concurrency problems in environments where print requests can come from more than one program at a time. The printer manager server is used only for printers attached to a parallel port.

If the printer manager server is installed in a workstation with one, two, or three parallel-attached printers, the server manages all these printers. Any possible locking situations are also prevented. The operation of the printers is serialized; that is, one print request is completed before another is accepted. Print requests can come from:

- 3287 printer emulator — see Chapter 30, “LANDP 3287 printer emulator API” on page 657.
- 4712, 4722 printer server, operating on a parallel-attached printer — see Chapter 8, “Financial printer server” on page 141.
- Printer servers you have developed (also the sample server PROPRIN).
- Non-LANDP programs, for example the DOS PRINT function or printing directly from an application. Such requests are all considered, by the printer manager server, as coming from a non-LANDP resource.

The presence of the printer manager server in a workstation does not affect the operation of printers attached to a serial port.

This chapter provides information on how the server handles service requests.

This chapter also provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields. If you need more information about the CPRB fields, see Appendix A, “Connectivity programming request block” on page 669.

*Table 31. Function Codes used in the Printer Manager Server*

| Function code | Description | Env. | Page |
|---|---|---|---|
| AC | Acquire a parallel printer | 0--- | 201 |
| RL | Release a parallel printer port | 0--- | 202 |
| | | | |
| You can also use these functions of the local resource manager: | | | |
| GS | Get status | 0--- | 541 |
| SA | Set alternate mode | 0--- | 542 |
| SE | Set exclusive mode | 0--- | 544 |

The first and second columns give the function code and the name of the function. The third column shows the operating environment of the function. “0---” means that it is available from LANDP for DOS servers only. The last column refers to the page where you can find the function described.

© Copyright IBM Corp. 1992, 2002

## Modes of operation

There are two modes of operation of the printer manager server:

**Alternate** The printer manager operates on a “first come, first served” basis. This is the default mode.

When the printer is idle (after initialization, after a timeout, or after the last character has been sent to be printed), the first print request to arrive reserves the printer. It is reserved for that resource until the timeout expires after the last printed character.

**Exclusive** The printer is assigned to one resource. Only print requests coming from this resource are served. All other requests get a “busy” return code. Even if the assigned resource is not using the port, no other resource can use it.

To set a printer in exclusive mode you need the operator interface or the local resource manager.

Print Screen and Ctrl-Print Screen can only be used in alternate mode. In exclusive mode, Ctrl-Print Screen returns a DOS error message. When the printer manager server is in alternate mode, there is also a way to assign one printer to one requester using the functions:

| Function | Meaning |
|---|---|
| **AC** | Acquire a parallel printer. |
| **RL** | Release a parallel printer port. |

After a successful AC function, any request from another resource gets a “busy” return code. Even if the resource that has acquired the printer does not use it, no other request is accepted. An RL is needed to make it accessible by other resources.

## Using the printer manager server

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | 26 | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 26 | 2 | 26 | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | PRTMGR | Server name |

The following fields are variable and are discussed in each function request description:
- Function code
- Request DATA length
- Reply DATA length

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 44 | 2 | | Replied PARMLIST length |
| 46 | 2 | | Replied DATA length |

If the request was successful, the *router return code* and the *server return code* are both X'00000000'. In all other cases, see the appropriate section in the *LANDP Problem Determination* book to see if you should take any action. The return values in Reply PARMLIST and DATA should be ignored if there is an error.

The following fields are variable and are discussed in each function request description:
- Replied PARMLIST length
- Replied DATA length

**PARMLIST:** The first byte of Request PARMLIST is used to store the port number.

## Request reference

This section gives you the programming information required to write application programs or your own servers that issue print requests. The functions are listed in alphabetical order of function code.

### Acquire a parallel printer (AC function)

This function reserves a parallel printer port for the requester. It is reserved until an RL function is requested.

| CPRB Field | Content/Description |
|---|---|
| Function code | AC |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Port number: X'00', X'01', or X'02' |

## Release a parallel printer port (RL function)

This function releases a parallel printer port previously reserved with an AC function.

| CPRB Field | Content/Description |
|---|---|
| Function code | RL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Port number: X'00', X'01', or X'02' |

# Chapter 12. Magnetic stripe reader/encoder server

Two servers provide access to Magnetic Stripe Reader/Encoders and PIN Pad devices. The devices supported are:

- IBM 4717 Magnetic Stripe Reader/Encoder
- IBM 4718 Personal Identification Number Key-pad
- IBM 4777 Magnetic Stripe Reader/Encoder
- IBM 4778 Personal Identification Number Key-pad, including its Magnetic Stripe Reader component

The following table shows which server can be used (✓) with these devices:

| Server | 4717 MSR/E | 4718 PIN Pad | 4777 MSR/E | 4778 MSR | 4778 PIN Pad |
|---|---|---|---|---|---|
| MSR/E | ✓ | | ✓ | ✓ | |
| PIN Pad* | | ✓ | | ✓ | ✓ |

**Notes:**

1. See Chapter 13, “Personal identification number pad server” on page 223.
2. The LANDP for Windows and LANDP for Linux servers do not support the 4717 or 4718 devices.

This chapter provides information on how the magnetic stripe reader/encoder (MSR/E) server handles service requests.

All functions operate in the LANDP for DOS, LANDP for OS/2, LANDP for Windows, and LANDP for Linux environments, except where shown. Some of the functions are not available under **LANDP for DOS** because of restrictions in the device driver supplied with DOS systems. See the *LANDP Introduction and Planning* book for more information.

This chapter also provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields. If you need more information about the CPRB fields, see Appendix A, “Connectivity programming request block” on page 669.

The MSR/E server provides the functions needed to use the features of the IBM 4717 or 4777 MSR/E (Magnetic Stripe Reader/Encoder) device or the MSR component of the IBM 4778 Pin Pad Unit, and to allow for sharing the device among two or more workstations. Using this server you can:

- Open and close
- Arm the device for reading data and obtaining the data read
- Initialize writing of data on the 47xx encoder and checking for completion of the write function
- Dynamically change the parameters for read and write formats

© Copyright IBM Corp. 1992, 2002

- Terminate a pending function
- Obtain device error statistics

The IBM 47x7 model 2 passbook encoder has a single wide encode head that physically covers both tracks 2 and 3. As a result, when you encode on track 2, track 3 is also encoded.

*Table 32. Function Codes used in the MSR/E Server*

| Function code | Description | Env. | Page |
|---|---|---|---|
| AR | Arm the IBM 47xx MSR/E device | 02NL | 206 |
| AT | Arm the IBM 47xx MSR/E device | -2NL | 207 |
| CH | Check the write status | 02NL | 208 |
| CL | Close | 02NL | 209 |
| DV | Load or retrieve device parameters | 02NL | 209 |
| EC | Get error counters and read/encode capability | 02NL | 211 |
| KL | Terminate a pending function | 02NL | 213 |
| OP | Open | 02NL | 213 |
| RD | Read from the IBM 47xx MSR/E | 02NL | 214 |
| WR | Write to IBM 47xx MSR/E device | 02NL | 214 |
| WT | Write to IBM 47xx MSR/E device | -2NL | 215 |

The first and second columns of the table give the function code and the name of the function. The third column shows the operating environment of the function, as explained in “Operating environments” on page xxvi. “02NL” means that it is available from LANDP for DOS, LANDP for OS/2, LANDP for Windows, and LANDP for Linux servers. The last column refers to the page where you can find the function described.

You can share a 47xx MSR/E among workstations connected to the LANDP workgroup using the following functions:
- OP (open)—Reserve a 47xx MSR/E for a specific application.
- CL (close)—Release an acquired 47xx MSR/E for use by other applications.

The 47xx MSR/E has lights that inform you of various conditions (the 4778 MSR component has arrows that point to colored marks):

- Lights during read operations

  **Green light — steady on = Ready to read**
  The light comes on as a result of an AR or AT function. You should now pass a magnetic medium through the slot. After the magnetic medium has passed through the slot, the green light goes off and the yellow light comes on.

  **Yellow light = Wait**
  The light comes on after you have passed the magnetic medium through the reader. It stays on until the validity checks on the data have been completed.

  **Red light = Error detected — Retry**
  If any of the data read from the active tracks fails the validity checks, the

red light comes on. This shows that a “bad read” on at least one of the tracks has occurred. The red light remains on until:
  – A subsequent good read occurs
  – The operation is canceled
  – The 47xx MSR/E is disabled

- Lights during encode operations

  **Flashing green light = Ready to encode**
  When the application issues a WR or WT function, the green light flashes a couple of times per second. You can now pass magnetic media through the encoder.

  **Yellow light = Wait**
  The light comes on after you have passed the magnetic medium through the encode slot. It stays on until after the “read back check” is complete.

  **Red light = Error Detected — Retry**
  The red light comes on to indicate a bad “read back check”. It stays on until:
  – A subsequent good encode occurs
  – The operation is canceled
  – The 47xx MSR/E is disabled

## Using the MSR/E server

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | 26 | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 26 | 2 | 26 | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | MSRE47## | Server name |
| **Note:** The ## in the server name is replaced by the server suffix, as specified in the `lanconf.spc` file. | | | |

The following fields are variable and are discussed in each function request description:
- Function code
- Request DATA length
- Reply DATA length

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 44 | 2 | | Replied PARMLIST length |
| 46 | 2 | | Replied DATA length |

If the request was successful, the *router return code* and the *server return code* are both X'00000000'. In all other cases, see the appropriate section in the *LANDP Problem Determination* book to see if you should take any action. The return values in Reply PARMLIST and Reply DATA should be ignored if there is an error.

The following fields are variable and are discussed in each function request description:
- Replied PARMLIST length
- Replied DATA length

**PARMLIST:** Only one PARMLIST field is used by the MSR/E server:

| Offset | Length | Content |
|---|---|---|
| 0 | 1 | Flag1 |

The flag1 field is used in the functions AR (see page 206), DV (see page 209), and OP (see page 213).

**DATA:** Request DATA and Reply DATA contain data to be sent to or received from the server. The use of these areas is explained in the description of each function request.

## Request reference

The functions supplied with the MSR/E server are synchronous, except for the functions AR, AT, WR, and WT, which are executed asynchronously.

When a synchronous request is complete, or when an asynchronous request is initiated, control returns to the application program.

## Arm the IBM 47xx MSR/E device (AR function)

This asynchronous function prepares the device to read. The green indicator on the device is turned on and control is returned to the application program. The application program should then check if any data has been read by:

- Requesting the local supervisor function WM or AA to wait for asynchronous notification (see “Event notification” on page 215) and then requesting the RD function
- Requesting the RD function repetitively

If you are using the IBM 4778 PIN Pad Unit, you cannot use its magnetic stripe and PIN Pad capabilities simultaneously.

| CPRB Field | Content/Description |
|---|---|
| Function code | AR |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 setting |

Flag1 must be set to 'C', if data is to be read in 4704 compatibility mode. The default is to read the data in non-4704 compatibility mode.

## Arm the IBM 47xx MSR/E device (AT function)

This asynchronous function prepares the device to read—you must define which track to read. It operates in the LANDP for OS/2, LANDP for Windows, and LANDP for Linux environments only.

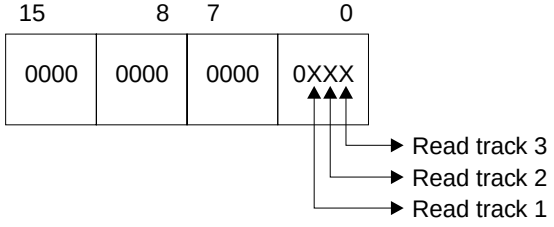
The green light on the device is turned on and control is returned to the application program. The application program then checks whether any data has been read by:

- Requesting the local supervisor function WM to wait for asynchronous notification (see "Event notification" on page 215) and then requesting the RD function (see page 214)
- Requesting the RD function repetitively (see page 214)
- Requesting the local supervisor function AA to wait for asynchronous notification (see "Event notification" on page 215) and then requesting the RD function (see page 214)

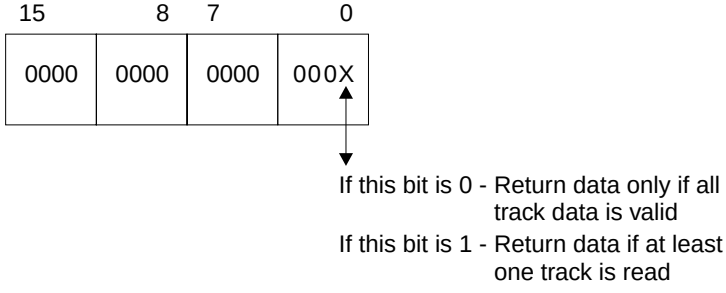
| CPRB Field | Content/Description |
|---|---|
| Function code | AT |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Read tracks |
| 2 | 2 | Multiple-track read operation mode |

Read tracks is a word value identifying which tracks the application program wants to read.




The multiple-track read operation is a word value that defines how the tracks are to be read.




When the 0-bit has a value 1, information is provided to show why other requested tracks could not be read.

## Check the write status (CH function)

This function checks if a write (WR or WT) function previously requested has completed. A return code of zero indicates that the write function has completed. Other return codes indicate that the write function is still in progress (P8), that the write function has an error (the return code for that request is returned), or that the CH request itself is in error.

| CPRB Field | Content/Description |
|---|---|
| Function code | CH |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |

| CPRB Field | Content/Description |
|---|---|
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

## Close (CL function)

This function closes the 47xx MSR/E device driver and releases it for use by another application program.

| CPRB Field | Content/Description |
|---|---|
| Function code | CL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

## Load or retrieve device parameters (DV function)

This function dynamically redefines 47xx MSR/E parameters. It is used when the defaults or current parameters should be altered.

| CPRB Field | Content/Description |
|---|---|
| Function code | DV |
| Request DATA length | 25 |
| Request PARMLIST length | 26 |
| Reply DATA length | 25 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 setting |

If flag1 = 'R', the DV function retrieves the existing device parameters and returns them to the application program in Reply DATA.

| Request and Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Track selection bit definition:<br>**7** Reserved<br>**6** Read track 1<br>**5** Read track 2<br>**4** Read track 3<br>**3** Reserved<br>**2** Encode track 1<br>**1** Encode track 2<br>**0** Encode track 3 |
| 1 | 1 | Track 1 data and longitudinal redundancy check (LRC) parity:<br>**X'00'** Odd data, odd LRC parity<br>**X'01'** Even data, even LRC parity<br>**X'02'** Odd data, even LRC parity<br>**X'03'** Even data, odd LRC parity |
| 2 | 1 | Track 1 bits per character (including the parity bit):<br>**X'05'** 5 bits per character<br>**X'06'** 6 bits per character<br>**X'07'** 7 bits per character |
| 3 | 1 | Track 1 primary start-of-message (PSOM) character |
| 4 | 1 | Track 1 alternate start-of-message (ASOM) character |
| 5 | 1 | Track 1 primary end-of-message (PEOM) character |
| 6 | 1 | Track 1 alternate end-of-message (AEOM) character |
| 7 | 1 | Track 1 format control (encoding only):<br>• If bit 7 of byte 7 is equal to 1, each byte is doubled<br>• If bit 7 is equal to 0, the byte is single<br>• Bits 6-0 are the number of inter-record zero characters |
| 8 | 1 | Track 1 number of leading zero characters |
| 9 | 8 | For track 2: content the same as bytes 1-8 |
| 17 | 8 | For track 3: content the same as bytes 1-8 |

See the *IBM Financial I/O Devices Programming Guide* for a complete explanation.

If you specify values outside these ranges, then the previously loaded values are assumed for all fields. If no previous values are set, the following default values are assumed for all fields:

| Read Defaults | | |
|---|---|---|
| Track 1 Read | Track 2 Read | Track 3 Read |
| Start of message (SOM) = X'05'<br><br>End of message (EOM) = X'1F'<br><br>Bits per character = X'07'<br><br>Odd data parity<br><br>Even LRC parity | Start of message (SOM) = X'0B' or X'0D'<br><br>End of message (EOM) = X'0F' or X'0C'<br><br>Bits per character = X'05'<br><br>Odd data parity<br><br>Even LRC parity | Start of message (SOM) = X'0B' or X'0D'<br><br>End of message (EOM) = X'0F' or X'0C'<br><br>Bits per character = X'05'<br><br>Odd data parity<br><br>Even LRC parity |
| Write Defaults | | |
| Track 1 Write | Track 2 Write | Track 3 Write |
| Records written = single<br><br>Leading zero bits = 70 (10 characters)<br><br>Bits per character = X'07'<br><br>Odd data parity<br><br>Even LRC parity | Records written = double for Model 2<br><br>Records written = single for Model 3<br><br>Leading zero bits = 200 (40 characters) for Model 2<br><br>Leading zero bits = 30 (6 characters) for Model 3<br><br>Inter-record zero bits = 25 (5 characters)<br><br>Bits per character = X'05'<br><br>Odd data parity<br><br>Even LRC parity | Records written = double<br><br>Leading zero bits = 200 (40 characters)<br><br>Inter-record zero bits = 25 (5 characters)<br><br>Bits per character = X'05'<br><br>Odd data parity<br><br>Even LRC parity |

## Get error counters and read/encode capability (EC function)

This function retrieves device error statistics collected since power-on time. The error statistics are shown as 8-bit binary values. Each counter can count to X'FF'. If the counter should be incremented further, the counter goes to X'80' and remains at that value until power off. Reading the counters does not affect their value.

This function also retrieves the read and encode capability for the attached magnetic stripe unit.

It can be issued without opening the server.

| CPRB Field | Content/Description |
|---|---|
| Function code | EC |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 20 means a request for the device error statistics, or 24 means a request for the device error statistics and read and encode capability |
| Reply PARMLIST length | 26 |
| Replied DATA length | 20 or 24 (as above) |
| Replied PARMLIST length | 0 |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Track 1 read errors |
| 2 | 2 | Track 2 read errors |
| 4 | 2 | Track 3 read errors |
| 6 | 2 | Encode read errors (for single track encoding only) |
| 8 | 2 | Read operator cancels |
| 10 | 2 | Encode operator cancels |
| 12 | 2 | All other magnetic stripe errors |
| 14 | 2 | Track 1 encode errors (for multiple-track encode only) |
| 16 | 2 | Track 2 encode errors (for multiple-track encode only) |
| 18 | 2 | Track 3 encode errors (for multiple-track encode only) |
| 20 | 2 | Read capability:<br>**Bits 7-4** Reserved<br>**Bit 3** Asynchronous mode<br>**Bit 2** Track 1<br>**Bit 1** Track 2<br>**Bit 0** Track 3 |
| 22 | 2 | Encode capability:<br>**Bits 7-3** Reserved<br>**Bit 2** Track 1<br>**Bit 1** Track 2<br>**Bit 0** Track 3 |
| **Note:** All even-numbered bytes are filled with X'00' | | |

## Terminate a pending function (KL function)

This function terminates any asynchronous function in process.

| CPRB Field | Content/Description |
|---|---|
| Function code | KL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

**LANDP for DOS only:** The device driver has an option, /K, to designate one of the keyboard keys as *cancel*. This option is not supported because:

- The service request may come from another workstation
- Some MSR/E functions are asynchronous

Use the function KL to cancel instead of the option /K of the device driver.

## Open (OP function)

This function opens the 47xx MSR/E device driver. You must use it before you request any other functions of this server except EC.

The 47xx MSR/E is then reserved for the application program that requested it. Following requests from other application programs are rejected with a *busy* return code.

To allow the shared use of this server, every application program must release the 47xx MSR/E immediately after the execution of the desired task. This is done with the CL function.

In LANDP for OS/2, LANDP for Windows, and LANDP for Linux, OP supports a flag 'K' in the Request PARMLIST that keeps unchanged the device parameters, as loaded by a DV function request. If 'K' is not specified, the parameters are set to their default values. In LANDP for DOS, the device parameters are not modified by the OP function. However, for LANDP for DOS, LANDP for OS/2, LANDP for Windows, and LANDP for Linux, you should use the DV function to set the parameters after each OP if this server is shared between applications that need different DV values. Each time the server is loaded, the default parameters apply.

| CPRB Field | Content/Description |
|---|---|
| Function code | OP |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |

| CPRB Field | Content/Description |
|---|---|
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

## Read from the IBM 47xx MSR/E (RD function)

This function reads data from the 47xx MSR/E buffer into Reply DATA.

| CPRB Field | Content/Description |
|---|---|
| Function code | RD |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | Maximum length of data to be read |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of data actually read |
| Replied PARMLIST length | 0 |

### Reply DATA values

Contains the data read. See “Input data formats” on page 216.

## Write to IBM 47xx MSR/E device (WR function)

This asynchronous function initializes writing to the 47xx MSR/E. The green light flashes and shows that the device is ready to encode, and control is returned to the application program. After that, the application program checks for the completion of the write with repetitive use of the CH function, or by:

- Requesting the local supervisor function WM to wait for asynchronous notification (see “Event notification” on page 215)
- Requesting the local supervisor function AA to wait for asynchronous notification

**Note:** If network-attached laser printer support is required, please read the section entitled “Network-attached laser printer support” in the chapter entitled “Preparing Windows workstations” in the *LANDP Installation and Customization*.

| CPRB Field | Content/Description |
|---|---|
| Function code | WR |
| Request DATA length | Length of data to be written:<br>2 to 255 in LANDP for DOS, LANDP for OS/2, and LANDP for Windows |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |

| CPRB Field | Content/Description |
|---|---|
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

### Request DATA values
Contains the data to be written. See “Output data formats” on page 219.

## Write to IBM 47xx MSR/E device (WT function)
This asynchronous function initializes writing to the MSR/E device. You must define the track to be written on. It operates in the LANDP for OS/2 and LANDP for Windows environments only.

The flashing green light on the device goes on and the application program takes control. The application program then checks for the completion of the write by repetitive use of the CH function (see page 208), or by:

- Requesting the local supervisor function WM to wait for asynchronous notification (see “Event notification”)
- Requesting the local supervisor function AA to wait for asynchronous notification

| CPRB Field | Content/Description |
|---|---|
| Function code | WT |
| Request DATA length | 2 to 512 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | (See above) | Data to be written (see “Output data formats” on page 219) |

## Event notification
When data is available to read from the 47xx MSR/E and the application program has requested a WM function, control is returned to the application program indicating that there is data to be read. The application program receives the Reply PARMLIST which contains:

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | RD |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 8 | MSRE47## |

The application program must request an RD function to obtain the data.

When a WR or WT function has been processed and the application program has requested WM function, control is returned to the application program, so you can check the results of the previous WR or WT function. The application program receives control and Replied PARMLIST contains:

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | WR |
| 2 | 8 | MSRE47## |

The application program must then request a CH function to check the results of the previous WR or WT.

## Input data formats

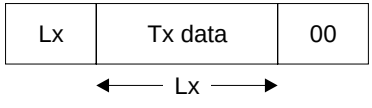
Three input data formats apply, depending on the type of compatibility mode being used and the operating system which your server runs:

- Format for AR function in non-4704 read compatibility mode
- Format for AR function in 4704 read compatibility mode
- Format for AT function is used (for LANDP for OS/2 only)

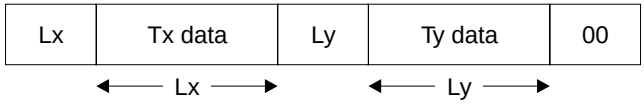
### Non-4704 read compatibility mode

The format of the data passed to the application program for reading one or two tracks, is shown in the following figure:

- Read 1 track.




- Read 2 tracks.




- The T fields contain the data read from the respective track and include the *start of message* (SOM) and *end of message* (EOM).

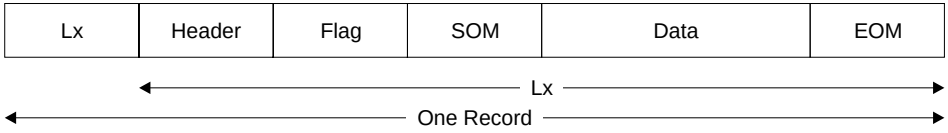
- The L fields are 1-byte fields that define the length of the T fields. The Lx and Ly bytes contain the length of the Tx and Ty data only.
- The X'00' byte defines the end of the inbound data message.

The lower numbered track is always the first data segment in the message. The next lower track is the next data segment.

If two SOMs are defined in the parameter table (PSOM and ASOM), either SOM is accepted as a valid start of message. The same is true for two EOMs.

## 4704 read compatibility mode

The format of the data passed to the application program for tracks 1 and 2 is shown in the following figure:




A record is the input data from one track, as depicted in the previous figure. If two tracks are used, the application program input buffer contains two records.

- The Lx field shows the length of the data, as depicted in the previous figure.
- The header field is a constant: X'0E'.
- The flag byte shows which track the data is from, and whether the data is the first or second and last record in the sequence. The bit definitions for this byte are as follows:

```
Bits 7-4  Reserved
Bits 3,2  01 Last record
          10 First record
          11 Only record
Bits 1,0  00 Track 1 data
          01 Track 2 data
          10 Track 3 data
          11 Reserved
```

A X'00' is appended to the last record, but it is not included in any length field.

Data read from any track without error is always placed in the application program input buffer. If no SOM is detected on a track, the track is considered blank and the following record is placed in the buffer:

| 04 | 0E | Flag | SOM | EOM |
|---|---|---|---|---|

If a SOM is detected but the data contains a parity or LRC error, the track is considered invalid and no record is placed in the buffer.

## Read format if AT function is used (LANDP for OS/2 only)

The format of the data read from the 47xx MSR/E and passed to the application program in its data buffer is:

Single Track

| | | |
|---|---|---|
| Lx | Sx | Datax |

Double Track

| | | | | | |
|---|---|---|---|---|---|
| Lx | Sx | Datax | Ly | Sy | Datay |

where:

- The L-fields represent a 1-byte field defining the length of the succeeding S and data fields. The L-field is not included in the count.

  When the data read from the stripe is valid, the value in the L-field is always 03 or more (SOM and EOM at a minimum, along with the S-field). If the value in the L-field equals 01, then the associated S-field contains the status that shows why the track could not be read.
- The S-fields represent a 1-byte status field defining which track the associated DATA field is from, along with the error status if the requested track did not contain a valid record. The format of the S-field is:

**S-field bit definitions**

| Bit | Meaning |
|---|---|
| 7 | No SOM found, track considered to be blank |
| 6 | SOM found, but either a parity, LRC, or EOM error detected |
| 5 | Reserved |
| 4 | Reserved |
| 3 | Reserved |
| 2 | Data from track 1 |
| 1 | Data from track 2 |
| 0 | Data from track 3 |

- The DATA fields represent the data read from the respective track. The SOM and EOM characters are included in this field, however, the LRC is not. Each magnetic character read from the stripe is returned as a single hexadecimal byte. If the L-field is equal to 01, the DATA field is not present because no data was found.

  If dual SOMs are defined in the parameter table, either SOM is accepted as a start-of-message. The same is true for dual EOMs.

**Note:** If you place any data in the input buffer of the application program, the read operation is considered valid, and the red light remains off. If errors are detected from any track, then the read operation fails and the red light goes on.

## Output data formats

***Single Track Encode:*** A single track encode operation is identified when the application program has set the track selection code (see DV Function) to a value that shows that only one of the three tracks is used for encoding.

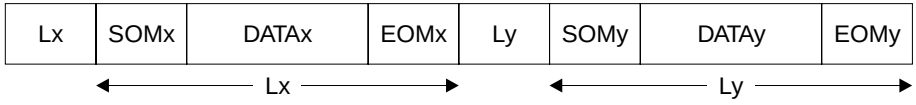
The format of the data passed from the application program must be:




The device driver takes the number of low order bits of each output byte (as specified by the *number of bits per character* parameter), and calculates the correct parity based on these bits. These bits, plus the parity bit, form the character to be encoded. The LRC is automatically calculated and added to the encoded record.

***Two Track Encode:*** A two track encode operation is identified when the application program sets the track selection code (see DV function) to a value that shows that two tracks are used for encoding.

The format of the data passed from the application program must be:




Valid combinations are:

| DATAx | DATAy |
|---|---|
| track 1 | track 2 |
| track 2 | track 3 |

The device driver takes the number of low order bits of each output byte (as specified by the *number of bits per character* parameter), and calculates the correct parity based on these bits. These bits, plus the parity bit, form the character to be encoded.

LRC is automatically calculated and added to the encoded record.

## Output data format used in WT function (LANDP for OS/2)

The magnetic subsystem adds the start-of-message and the end-of-message characters to the application program data stream using the primary-start-of-message (PSOM) character and the primary-end-of-message (PEOM) character from the respective track parameters (see “Load or retrieve device parameters (DV function)” on page 209). The magnetic subsystem also calculates and adds the LRC value to the data stream to be encoded. The format of the encoded data from the application program in its data buffer

is:

Single Track




Double Track




where:

The L-field is a 1-byte field for the respective TRK and DATA fields. The L-field is not included in this length count.

The TRK field identifies the track to which the respective DATA fields are written. Only one track can be selected within each respective TRK field, otherwise an error code is generated. The format of the TRK field is:

**TRK field bit definitions**

| Bit | Meaning |
|---|---|
| 7 | Reserved |
| 6 | Reserved |
| 5 | Reserved |
| 4 | Reserved |
| 3 | Reserved |
| 2 | Encode onto track 1 |
| 1 | Encode onto track 2 |
| 0 | Encode onto track 3 |

The DATA field represents hexadecimal track data to be encoded onto the magnetic media. The magnetic subsystem adds the SOM, EOM, and LRC characters into the data stream before sending it to the 47xx device.

## Migration considerations

The LANDP for OS/2 MSR/E server provides the same functions that are available for the LANDP for DOS and FBSS (DOS) server and a new set of functions to exploit the new capabilities of the OS/2 47xx MSR/E device driver. You may want to change your applications to use these functions. However, if the application must be able to request services from a LANDP for DOS or an FBSS (DOS) 4717 server and from a LANDP for OS/2 47xx server, it has to use the functions that are common.

The OS/2 47xx MSR/E device driver resets device parameters when issuing an open (OP) function. The DOS 47xx MSR/E device driver does not reset these parameters. If

you have applications that request services from the LANDP for DOS or FBSS (DOS) MSR/E server and you want to move this service to the LANDP for OS/2 environment, you have to use the open (OP) function with Flag1='K' to keep the device parameters definition.

Your application programs can access the MSR component of the IBM 4778 Pin Pad Unit by changing from the MSR/E server to the PIN pad server. In this way they can access liquid crystal display (LCD) and MSR capabilities of this device, with minimal change (the server name and some flags). You are recommended to change your applications because, in addition to the new hardware functions, you avoid loading the MSR/E server to perform functions that can be done by the PIN Pad server. You can also use the IBM 4777 and 4778 at the same time.

# Chapter 13. Personal identification number pad server

Two servers provide access to Magnetic Stripe Reader/Encoders and PIN Pad devices. Devices supported are:

- IBM 4717 Magnetic Stripe Reader/Encoder
- IBM 4718 Personal Identification Number Key-pad
- IBM 4777 Magnetic Stripe Reader/Encoder
- IBM 4778 Personal Identification Number Key-pad, including its Magnetic Stripe Reader component

The following table shows which server can be used (✓) with these devices:

| Server | 4717 MSR/E | 4718 PIN Pad | 4777 MSR/E | 4778 MSR | 4778 PIN Pad |
|---|---|---|---|---|---|
| MSR/E* | ✓ | | ✓ | ✓ | |
| PIN Pad | | ✓ | | ✓ | ✓ |

**Notes:**

1. See Chapter 12, “Magnetic stripe reader/encoder server” on page 203.
2. The LANDP for Windows and LANDP for Linux servers do not support the 4717 or 4718 devices.

This chapter provides information on how the personal identification number (PIN) pad server handles service requests. Except where stated, all functions operate in the LANDP for DOS, LANDP for OS/2, LANDP for Windows, and LANDP for Linux environments.

This chapter also provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields. If you need more information about the CPRB fields, see Appendix A, “Connectivity programming request block” on page 669.

The PIN (personal identification number) pad server provides functions to use the features of the IBM 4718 or 4778 PIN Pad device and to allow for sharing the device among two or more workstations. It also accesses the Magnetic Stripe Reader/Encoder component of the 4778. Using this server you can:

- Open and close devices: you can share a 47xx PIN pad among workstations connected to the LANDP workgroup.
- Arm the device for reading data and obtaining the data read. A parameter is used to define where the data is to be read from, and in the case of reading from a PIN pad device, how the data keyed on the pad is treated:
  - Clear (not encrypted) data
  - Master key data
  - 4700 compatibility mode data (LANDP for DOS only)
  - Encrypted data
  - Verify PIN block data
  - Create PIN offset data

© Copyright IBM Corp. 1992, 2002

  - MSR data
- Terminate a pending function
- Read the serial number of a PIN pad device
- Functions are available to use the built-in security processor:
  - Load master key
  - Load keys
  - Load initial chaining values
  - Load PIN verification parameters
- Generate and verify message authentication codes

*Table 33. Function Codes used in the PIN Pad Server*

| Function code | Description | Env. | Page |
|---|---|---|---|
| AR | Arm the PIN pad or MSR for data input | 02NL | 228 |
| AT | Arm the MSR device | -2NL | 234 |
| CL | Close PIN pad or MSR | 02NL | 236 |
| DV | Load or retrieve MSR device parameters | 02NL | 236 |
| EC | Get error counters and MSR read/encode capability | 02NL | 239 |
| GA | Generate message authentication code | 02NL | 240 |
| IV | Load initial chaining value | 02NL | 241 |
| KL | Terminate a pending function | 02NL | 242 |
| LK | Load key | 02NL | 242 |
| LM | Load master key | 02NL | 243 |
| LP | Load PIN verification parameters | 02NL | 244 |
| OP | Open PIN pad | 02NL | 245 |
| RD | Read from the IBM 47xx PIN pad | 02NL | 246 |
| RN | Read serial number | 02NL | 247 |
| VA | Verify message authentication code | 02NL | 248 |
| WD | Write display | 02NL | 249 |

The first and second columns give the function code and the name of the function. The third column shows the operating environment of the function. "02NL" means that it is available from LANDP for DOS, LANDP for OS/2, LANDP for Windows, and LANDP for Linuxservers. The last column refers to the page where you can find the function described.

## Cryptographic function details

This section describes some of the concepts mentioned frequently in this chapter. For more detail, see the *4700 Finance Communication System, Controller Programming, Volume 5, Cryptographic Programming* and the *IBM 4700 Finance I/O Devices Programming Guide.*

**Key variant**

A variant of a cryptographic key is formed when you combine the key with a non-secret quantity. You do this while non-parity bits in the new key differ from the corresponding bits of the original key. The 47xx

PIN pad uses a combination process defined to be the exclusive or logical operation, XOR, of the original key with a defined non-secret quantity. This produces the named variant of the original key. The defined, non-secret quantity is the concatenation of eight single variant definition bytes (16 for double length keys).

The PIN keypad contains a table of variant definition bytes (VDB). This table is arranged as 16 sets of 4 VDBs each. When you use variants as part of a particular function, you must specify a variant table descriptor that points to one of the 16 sets. Variants are defined for only the first six positions of the table that correspond to the actual keypad functions. Since you use only a given variant with designated functions, the use of variants permits you to ensure that a key can be used only for the purpose for which it is intended.

**Triple encryption**

Triple encryption is a cryptographic process where you complete the following steps:

1. Encrypt the 8 bytes of data with the first 8 bytes of a double length key.
2. Decrypt the result with the second 8 bytes of the double length key.
3. Encrypt the result again, using the first 8 bytes of the double length key.

If you use the same 8 bytes for the encryption and decryption steps (for an 8-byte master key), the result is the same as if a single encryption step were performed with a single length (8-byte) key.

**Decimalization table**

The decimalization table is a string of 16 decimal digits, each stored in one byte, that the 47xx PIN pad uses to modify the encrypted validation data.

**Dual entry of the master key**

Dual entry of the master key is a security measure. Two people have to enter their personal part of the key, which ensures that one person alone cannot access the master key. Each key is either eight or 16 bytes long, and has valid parity on every byte. The 47xx PIN pad combines the two components with an XOR (exclusive or) operation, then adjusts the resulting key to have correct parity before storing it.

**PAD character**

The PAD character specifies the digit that is used to pad the PIN.

## Using the PIN pad server

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | 26 | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 26 | 2 | 26 | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | PINP47## | Server name |
| **Note:** The ## in the server name is replaced by the server suffix, as specified in the `lanconf.spc` file. | | | |

The following fields are variable and are discussed in each function request description:
- Function code
- Request DATA length
- Reply DATA length

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 44 | 2 | | Replied PARMLIST length |
| 46 | 2 | | Replied DATA length |

If the request was successful, the *router return code* and the *server return code* are both X'00000000'. In all other cases, see the appropriate section in the *Problem Determination* book to see if you should take any action. The return values in Reply PARMLIST and Reply DATA should be ignored if there is an error.

The following fields are variable and are discussed in each function request description:
- Replied PARMLIST length
- Replied DATA length

**PARMLIST:** The Request PARMLIST and Reply PARMLIST fields used by the PIN pad server are:

| Offset | Length | Content |
|---|---|---|
| 0 | 1 | Flag1 |
| 1 | 1 | Flag2 |
| 2 | 1 | Flag3 |
| 3 | 1 | Flag4 |
| 4 | 1 | Flag5 |
| 5 | 1 | Flag6 |
| 10 | 8 | Key1 (16 hexadecimal digits) |
| 18 | 8 | Key2 (16 hexadecimal digits) |

**DATA:** Request DATA and Reply DATA contain data to be sent to or received from the server. The use of these areas is explained in the description of each function request.

For a complete explanation of the PIN Pad Data Stream, see the *4700 Finance Communication System, Controller Programming, Volume 5, Cryptographic Programming,* and the *IBM 4700 Finance I/O Devices Programming Guide.*

## PIN pad server flag summary

Following are the PIN pad server flags and their meanings:

| | | |
|---|---|---|
| Flag1 | 'B' | Close both the MSR and the PIN pad sessions |
| | 'B' | Open both the MSR and the PIN pad sessions |
| | 'C' | Encrypted and 4700 compatibility PIN pad (LANDP for DOS only) |
| | 'E' | Encrypted PIN pad |
| | 'M' | PIN pad master key |
| | 'N' | Non-encrypted PIN pad. |
| | 'O' | Create PIN offset data. |
| | 'P' | Close the PIN pad session. |
| | 'P' | Open the PIN pad session. |
| | 'R' | Retrieve the MSR parameters. |
| | 'S' | Open the MSR session. |
| | 'S' | MSR data. |
| | 'S' | Close the MSR session. |
| | 'V' | Verify PIN block. |
| | X'00'—X'FF' | Key identifier (hexadecimal) |
| Flag2 | '0'—'6' | Variant table descriptor |
| | '1' | 16-byte key |
| | '2' | 8-byte key |
| | 'C' | Read MSR data in 4704-compatibility mode |

| | | |
|---|---|---|
| | 'K' | MSR session: restore the values set in the last DV request |
| Flag3 | '1' | IBM 4704 PIN format |
| | '2' | ANSI X9.8 PIN format |
| | '3' | IBM 3624 PIN format |
| | 'D' | Dual entry of key |
| | 'E' | Triple encrypted entry |
| | 'N' | Clear entry |
| | 'S' | Single entry of key |
| Flag4 | '1' | Master key format |
| | '2' | Internal key identifier |
| | '3' | Key data field format |
| Flag5 | '0'—'F' | Pad character |
| | '0'—'F' | Length of PIN to be checked |
| | 'N' | Internal ICV, offset data not present |
| | 'Y' | ICV in data/offset data present |
| Flag6 | '0', '2' | Variant used to decrypt ICV |
| | 'E' | Set encrypted mode |
| | 'N' | Do not change the mode |

Any other values are ignored and the server uses the default values. The application program must reset the values of not used flags before requesting a service. If this is not done, an error might occur.

**Attention:** If the operation fails when you are using one of the following functions, the PIN pad clears all keys:
- Enter master key (AR, option 'M') setting encrypted mode (flag6 = 'E')
- Load master key (LM) setting encrypted mode (flag6 = 'E')

A new master key must be provided.

## Request reference

This section gives you the programming information required to write application programs or your own servers that issue service requests to the PIN pad server. The functions are listed in alphabetical order of function code.

When a synchronous request is complete or when an asynchronous request is initiated, control returns to the application program.

### Arm the PIN pad or MSR for data input (AR function)

This function prepares the 47xx PIN pad or MSR for reading. It is always asynchronous. The green light on the device goes on, and control is returned to the application

program. (On the 4778, a downward arrow is displayed when the PIN pad component is armed, and an upward arrow when arming the MSR component. These arrows point to colored marks.)

The application program must then read data from the device buffer with the RD function, or wait for the asynchronous notification. For more information on how LANDP for DOS, LANDP for OS/2, LANDP for Windows, and LANDP for Linux handle the asynchronous notification, see “Event notification” on page 249. Flag1 is used to choose where the data is to be read from, how the data keyed on the PIN pad is treated, and to choose the MSR or PIN pad component of the 4778:

**’N’** Non-encrypted data

**’M’** PIN pad master key data

**’C’** LANDP for DOS: 4700 compatibility mode data (only when the master key is loaded or entered in encrypted mode)

**’E’** Encrypted PIN pad data (only when the master key is loaded or entered in encrypted mode)

**’V’** Verify PIN block data

**’O’** Create PIN offset data

**’S’** MSR data

**Notes:**

1. Using the 47xx PIN pad for reading non-encrypted data prevents its use for encrypted and 4700 compatibility data, until all Data Encryption Standard (DES) keys are reloaded in encrypted mode.
2. If you are using the IBM 4778 PIN Pad Unit, you cannot use its magnetic stripe and PIN Pad capabilities simultaneously.

## Function AR (Non-encrypted PIN pad data option)

The data input keyed on the pad is passed to the application program without encryption.

| CPRB Field | Content/Description |
|---|---|
| Function code | AR |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 = 'N' |

**Note:** This function disables the later use of encrypted data and 4700 compatibility data (LANDP for DOS only) until the master key and other needed keys are loaded in encrypted mode.

## Function AR (Enter master key option)

The master key is used to encrypt other keys or for encrypting and decrypting data.

This option lets you manually enter the master key into the 47xx PIN pad using the 47xx PIN keypad. You can choose any of the following four ways to enter the master key:

- 8-byte master key—single entry
- 8-byte master key—dual entry
- 16-byte master key—single entry
- 16-byte master key—dual entry

The PIN pad light (or the downward pointing arrow) goes on to show the start of the master key entry and remains on until the operation is completed. If you press an invalid key, or enter a byte with bad parity, the operation stops at that point. Otherwise, it ends automatically when the last digit of the master key has been entered.

If an 8-byte master key is entered at the 47xx, it is duplicated as the second 8 bytes of the double-length master key in the 47xx. If a dual entry option is selected, the two entries are XORed together to produce the master key.

You must enter the master key in 3-3-2 format. This format is explained in the *IBM 4700 Financial I/O Devices Programming Guide.*

| CPRB Field | Content/Description |
|---|---|
| Function code | AR |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 = 'M' |
| 1 | 1 | Flag2 Type of key<br>**'1'** 16 bytes (48 keystrokes)<br>**'2'** 8 bytes (24 keystrokes) |
| 2 | 1 | Flag3 Method of entry<br>**'S'** Single entry<br>**'D'** Dual entry (XOR the two parts) |
| 5 | 1 | Flag6 Mode selection<br>**'E'** Set encrypted mode<br>**'N'** Do not change the mode |

## Function AR (Encrypted and 4700 compatibility option)

The data input keyed on the pad is encrypted and passed to the application program in the same format as the one passed by the 4704 encrypted PIN pad.

This function operates in the **LANDP for DOS** environment only.

| CPRB Field | Content/Description |
|---|---|
| Function code | AR |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 = 'C' |

## Function AR (Encrypt PIN pad option)

The PIN entered on the pad is encrypted using the key and format shown by the calling data.

| CPRB Field | Content/Description |
|---|---|
| Function code | AR |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |

| CPRB Field | Content/Description |
|---|---|
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 = 'E' |
| 1 | 1 | Flag2 Variant table descriptor<br>**'0'** Variants not used<br>**'3'** Fixed variant 3 used |
| 2 | 1 | Flag3 PIN block format<br>**'1'** IBM 4704<br>**'2'** ANSI X9.8<br>**'3'** IBM 3624 |
| 3 | 1 | Flag4 Key format<br>**'1'** Master key<br>**'2'** Key identifier<br>**'3'** 8-byte key data field |
| 4 | 1 | Flag5 Pad character, 0–F in character format |
| 10 | 8 | • If flag4 = '3', it contains the key encrypted under the appropriate variant of the master key in hexadecimal<br>• If flag4 = '2', it contains a 1-byte identifier, see function LK on page 242 |

***Request DATA values:*** When arming in ANSI X9.8 format, the first six bytes of Request DATA contain a 12-digit identification number, for example a personal account number (PAN). The digits are stored as 4-bit binary numbers.

Other formats do not need any PAN.

## Function AR (Verify PIN block option)

PIN entries are verified using the PIN offset and validation data read from a magnetic stripe card. The PIN pad encrypts and compares the data passed with the function according to the 3624 algorithm.

| CPRB Field | Content/Description |
|---|---|
| Function code | AR |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 = 'V' |
| 1 | 1 | Flag2 Variant table descriptor<br>**'0'** Variants not used<br>**'4'** Fixed variant 4 used |
| 3 | 1 | Flag4 Key format<br>**'1'** Master key<br>**'2'** Key identifier<br>**'3'** 8-byte key data field |
| 4 | 1 | Flag5 Offset<br>**'Y'** Offset data present<br>**'N'** Offset data not present |
| 10 | 8 | • If flag4 = '3', it contains the encrypted key 1 in hexadecimal<br>• If flag4 = '2', it contains a 1-byte identifier, see function LK on page 242 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Verification data read from the magnetic card |
| 8 | 8 | Offset data read from the magnetic card (if required) |

## Function AR (Create PIN offset data option)

This function generates a PIN offset parameter to be used in verifying PINs with the 3624 algorithm. The PIN pad computes the offset by cryptographic combination of the validation data entered by the application program with the PIN entered using the pad.

| CPRB Field | Content/Description |
|---|---|
| Function code | AR |
| Request DATA length | 8 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 = 'O' (letter O) |
| 1 | 1 | Flag2 Variant table descriptor<br>**'0'** Variants not used<br>**'4'** Fixed variant 4 used |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 1 | Flag3 is not used |
| 3 | 1 | Flag4 Key format<br>**'1'** Master key<br>**'2'** Key identifier<br>**'3'** 8-byte key data field |
| 4 | 1 | Flag5 is not used |
| 10 | 8 | • If flag4 = '3', it contains the key encrypted under the appropriate variant of the master key in hexadecimal<br>• If flag4 = '2', it contains a 1-byte identifier, see function LK on page 242 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Validation data to be used in creating the offset |

## Function AR (MSR data option)

This function arms the magnetic stripe reader of the IBM 4778 PIN Pad Unit.

| CPRB Field | Content/Description |
|---|---|
| Function code | AR |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 = 'S' |
| 1 | 1 | Flag2 Compatibility mode<br>**'C'** Read data in 4704 compatibility mode<br>Default is to read data in non-4704 compatibility mode |

## Arm the MSR device (AT function)

This function is used to access the MSR capabilities of the IBM 4778 PIN Pad Unit. Its interface is the same as that for the MSR/E server, except for the server name

(PINP47##). It prepares the device to read—you must define which track to read. It operates in the LANDP for OS/2, LANDP for Windows, and LANDP for Linux environments only.

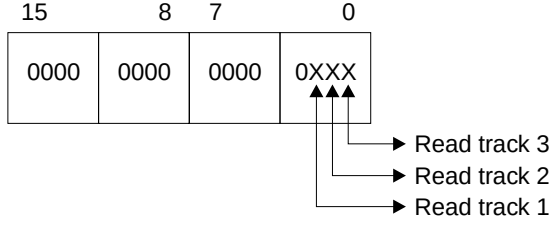
The green light on the device (or the upward pointing arrow) is turned on and control is returned to the application program. The application program then checks whether any data has been read by:

- Requesting the local supervisor function WM to wait for asynchronous notification (see “Event notification” on page 249) and then requesting the RD function (see page 246).
- Requesting the RD function repetitively (see page 246).
- Requesting the local supervisor function AA to wait for asynchronous notification (see “Event notification” on page 249) and then requesting the RD function (see page 246).

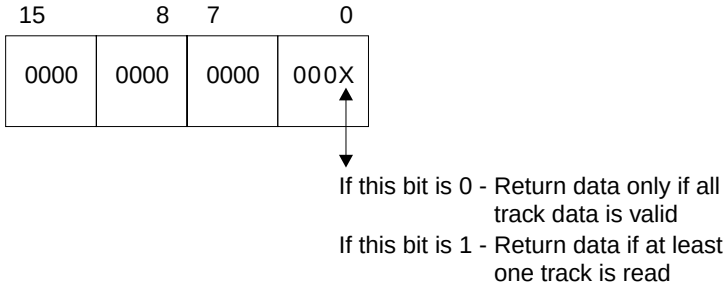
| CPRB Field | Content/Description |
|---|---|
| Function code | AT |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Read tracks |
| 2 | 2 | Multiple-track read operation mode |

Read tracks is a word value identifying which tracks the application program wants to read.



The multiple-track read operation is a word value that defines how the tracks are to be read.




When the 0-bit has a value 1, information is provided to show why other requested tracks could not be read.

## Close PIN pad or MSR (CL function)

This function closes and releases the server for use by another application program. Use this function, when your application program is finished with the 47xx PIN pad, so that another application program can use it.

| CPRB Field | Content/Description |
|---|---|
| Function code | CL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1. Session.<br>**'B'** Close both the MSR and the PIN pad sessions.<br>**'P'** Close the PIN pad session.<br>**'S'** Close the MSR session.<br>Default is to close the PIN pad session |

## Load or retrieve MSR device parameters (DV function)

This function is used to access the MSR capabilities of the IBM 4778 PIN Pad Unit. Its interface is the same as that for the MSR/E server, except for the server name (PINP47##). It dynamically redefines 4778 MSR parameters. It is used when the defaults or current parameters should be altered.

| CPRB Field | Content/Description |
|---|---|
| Function code | DV |
| Request DATA length | 25 or 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 or 25 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 setting |

If flag1 = 'R', the DV function retrieves the existing device parameters and returns them to the application program in Reply DATA.

| Request and Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Track selection bit definition:<br>**7** Reserved<br>**6** Read track 1<br>**5** Read track 2<br>**4** Read track 3<br>**3** Reserved<br>**2** Encode track 1<br>**1** Encode track 2<br>**0** Encode track 3 |
| 1 | 1 | Track 1 data and longitudinal redundancy check (LRC) parity:<br>**X'00'** Odd data, odd LRC parity<br>**X'01'** Even data, even LRC parity<br>**X'02'** Odd data, even LRC parity<br>**X'03'** Even data, odd LRC parity |
| 2 | 1 | Track 1 bits per character (including the parity bit):<br>**X'05'** 5 bits per character<br>**X'06'** 6 bits per character<br>**X'07'** 7 bits per character |
| 3 | 1 | Track 1 primary start-of-message (PSOM) character |
| 4 | 1 | Track 1 alternate start-of-message (ASOM) character |
| 5 | 1 | Track 1 primary end-of-message (PEOM) character |
| 6 | 1 | Track 1 alternate end-of-message (AEOM) character |
| 7 | 1 | Track 1 format control (encoding only):<br>• If bit 7 is equal to 1, each byte is doubled<br>• If bit 7 is equal to 0, the byte is single<br>• Bits 6-0 are the number of inter-record zero characters |

| Request and Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 8 | 1 | Track 1 number of leading zero characters |
| 9 | 8 | For track 2: content the same as bytes 1-8 |
| 17 | 8 | For track 3: content the same as bytes 1-8 |

Refer to the *IBM Financial I/O Devices Programming Guide* for a complete explanation.

If you specify values outside these ranges, then the previously loaded values are assumed for all fields. If no previous values are set, the following default values are assumed for all fields:

| Read Defaults | | |
|---|---|---|
| Track 1 Read | Track 2 Read | Track 3 Read |
| Start of message (SOM) = X'05'<br><br>End of message (EOM) = X'1F'<br><br>Bits per character = X'07'<br><br>Odd data parity<br><br>Even LRC parity | Start of message (SOM) = X'0B' or X'0D'<br><br>End of message (EOM) = X'0F' or X'0C'<br><br>Bits per character = X'05'<br><br>Odd data parity<br><br>Even LRC parity | Start of message (SOM) = X'0B' or X'0D'<br><br>End of message (EOM) = X'0F' or X'0C'<br><br>Bits per character = X'05'<br><br>Odd data parity<br><br>Even LRC parity |
| Write Defaults | | |
| Track 1 Write | Track 2 Write | Track 3 Write |
| Records written = single<br><br>Leading zero bits = 70 (10 characters)<br><br>Bits per character = X'07'<br><br>Odd data parity<br><br>Even LRC parity | Records written = double for Model 2<br><br>Records written = single for Model 3<br><br>Leading zero bits = 200 (40 characters) for Model 2<br><br>Leading zero bits = 30 (6 characters) for Model 3<br><br>Inter-record zero bits = 25 (5 characters)<br><br>Bits per character = X'05'<br><br>Odd data parity<br><br>Even LRC parity | Records written = double<br><br>Leading zero bits = 200 (40 characters)<br><br>Inter-record zero bits = 25 (5 characters)<br><br>Bits per character = X'05'<br><br>Odd data parity<br><br>Even LRC parity |

## Get error counters and MSR read/encode capability (EC function)

This function is used to access the MSR capabilities of the IBM 4778 PIN Pad Unit. Its interface is the same as that for the MSR/E server, except for the server name (PINP47##). It retrieves device error statistics collected since power-on time. The error statistics are shown as 8-bit binary values. Each counter can count to X'FF'. If the counter should be incremented further, the counter goes to X'80' and remains at that value until power off. Reading the counters does not affect their value.

Also, this function retrieves the read and encode capability for the attached magnetic stripe unit.

| CPRB Field | Content/Description |
|---|---|
| Function code | EC |
| Request DATA length | 0 |
| Request PARMLIST length | |
| Reply DATA length | 20 means a request for the device error statistics, or 24 means a request for the device error statistics and read and encode capability |
| Reply PARMLIST length | 26 |
| Replied DATA length | 20 or 24 (as above) |
| Replied PARMLIST length | 0 |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Track 1 read errors |
| 2 | 2 | Track 2 read errors |
| 4 | 2 | Track 3 read errors |
| 6 | 2 | Encode read errors (for single track encoding only) |
| 8 | 2 | Read operator cancels |
| 10 | 2 | Encode operator cancels |
| 12 | 2 | All other magnetic stripe errors |
| 14 | 2 | Track 1 encode errors (for multiple-track encode only) |
| 16 | 2 | Track 2 encode errors (for multiple-track encode only) |
| 18 | 2 | Track 3 encode errors (for multiple-track encode only) |
| 20 | 2 | Read capability:<br>**Bits 7-4** Reserved<br>**Bit 3** Asynchronous mode<br>**Bit 2** Track 1<br>**Bit 1** Track 2<br>**Bit 0** Track 3 |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 22 | 2 | Encode capability:<br>**Bits 7-3** Reserved<br>**Bit 2** Track 1<br>**Bit 1** Track 2<br>**Bit 0** Track 3 |
| **Note:** All even-numbered bytes are filled with X'00' | | |

## Generate message authentication code (GA function)

This function is used to generate a message authentication code (MAC) on a data string of up to 1KB in length. A MAC can be used to ensure data integrity when a message is transmitted from one node to another through an unprotected communication link. The MAC is generated at the sending node and sent along with the message. At the receiving end, the MAC is verified to ensure that it is the same as that transmitted by the sending node. If the MAC does not verify, it can be assumed that some of the data was either intentionally or unintentionally modified. The data must be presented as a multiple of eight bytes. No padding or element extraction is provided.

| CPRB Field | Content/Description |
|---|---|
| Function code | GA |
| Request DATA length | 0 to 1024 |
| Request PARMLIST length | 26 |
| Reply DATA length | 8 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 8 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Not used |
| 1 | 1 | Flag2 Variant table descriptor<br>**'0'** Variants not used<br>**'5'** For fixed variant 05 |
| 2 | 1 | Not used |
| 3 | 1 | Flag4 Key format<br>**'1'** Master key<br>**'2'** Key identifier<br>**'3'** 8-byte key data field |
| 4 | 1 | Flag5 Initial chaining value format<br>**'Y'** Initial chaining value in data<br>**'N'** Internal ICV |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 5 | 1 | Flag6 Variants used to decrypt ICV<br>**'0'** Variants not used<br>**'2'** Fixed variant 02 |
| 10 | 8 | • Key1 encrypted under the appropriate variant of the master key (if flag4 = '3'), in hexadecimal<br>• If flag4 = '2', this contains a 1-byte identifier, see function LK on page 242 |
| 18 | 8 | ICV encrypted under the appropriate variant of the master key (if flag5 = 'Y') |

### Request DATA values

Data to generate a MAC. The length must be a multiple of eight bytes.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 8-byte MAC |
| 0 | 4 | For application programs requiring only four bytes, the application program selects the first four to be retained |

**Note:** This function may take up to 60 seconds for execution. To avoid errors while requesting this function, you must specify sufficient LAN timeout during customization. Refer to the *LANDP Installation and Customization* book for more details.

## Load initial chaining value (IV function)

Use this function to store an initial chaining value (ICV) in the PIN pad security processor. The ICV can be any 8-byte value. It is passed from the workstation, triple encrypted under a variant of the master key. Failure to load the triple encrypted ICV results in an unexpected ICV in the 47xx. An unexpected ICV occurs because the ICV is decrypted at the 47xx using the master key.

| CPRB Field | Content/Description |
|---|---|
| Function code | IV |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 1 | 1 | Flag2 Variant table descriptor<br>**'0'** Variants not used<br>**'2'** Fixed variant 02 |
| 18 | 8 | Key2 Initial chaining value |

## Terminate a pending function (KL function)

This function disarms the PIN pad or the MSR component of the 4778, and the light-emitting diode (LED) of the 4718 or the arrow of the 4778 goes off.

| CPRB Field | Content/Description |
|---|---|
| Function code | KL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

**LANDP for DOS only:** The device driver has an option, /K, to designate one of the keyboard keys as *cancel*. This option is not supported in LANDP for DOS because the service request may come from another workstation. Use the function KL to cancel instead of the option /K of the device driver.

## Load key (LK function)

This function is used to load keys into the non-volatile memory of the PIN pad security processor. The keys are passed from the workstation to the PIN pad, triple encrypted under an appropriate variant of the master key. The keys are decrypted to check for valid parity but are stored in encrypted form for later use.

The 47xx PIN pad can store 256 keys and the master key.

When you load a key, you assign a key identifier to it (from X'00' to X'FF'). You must supply this key identifier any time you want to use a particular key.

| CPRB Field | Content/Description |
|---|---|
| Function code | LK |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 8 |
| Reply PARMLIST length | 26 |

| CPRB Field | Content/Description |
|---|---|
| Replied DATA length | 8 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1. Key identifier X'00' to X'FF' |
| 1 | 1 | Flag2 Variant table descriptor<br>**'0'** Variants not used<br>**'3'** Fixed variant 03<br>**'4'** Fixed variant 04<br>**'5'** Fixed variant 05<br>**'6'** Fixed variant 06 |
| 10 | 8 | Key1 |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Triple encryption of the device serial number with the loaded key |

## Load master key (LM function)

This function is used to load a new master key into the 47xx PIN pad. The master key is used for encrypting other keys, or for encrypting and decrypting data. You can load the master key using any of the options:

- 8-byte master key (non-encrypted)
- 8-byte master key (encrypted)
- 16-byte master key (non-encrypted)
- 16-byte master key (encrypted)

If you load an 8-byte master key into the 47xx, it is duplicated as the second eight bytes of the double length master key. If you load the master key in encrypted form, it is decrypted by the 47xx PIN pad using the resident master key.

If the new master key has correct parity, the old key is replaced and the triple encryption of the serial number with the new master key is returned to the application program. If the new master key has incorrect parity, an error code is returned.

| CPRB Field | Content/Description |
|---|---|
| Function code | LM |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 8 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 8 |

| CPRB Field | Content/Description |
|---|---|
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 is not used |
| 1 | 1 | Flag2 Type of key<br>**'1'** 16-byte key<br>**'2'** 8-byte key |
| 2 | 1 | Flag3 Method of entry<br>**'N'** Non-encrypted<br>**'E'** Triple encrypted under previous key |
| 5 | 1 | Flag6 Mode selection<br>**'E'** Set encrypted mode<br>**'N'** Do not change the mode |
| 10 | 8 | New master key (first 8 bytes) |
| 18 | 8 | New master key (last 8 bytes) |

**Note:** If flag6 = 'E', you cannot choose flag3 = 'E', because setting encrypted mode means that you lose all previous keys. This combination is not allowed and the server issues the return code P4, even though the flag values are valid.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Triple encryption of the device serial number with the loaded key |

## Load PIN verification parameters (LP function)

This function is used to load the parameters needed by the PIN pad to either verify the PIN or create PIN offset data. The parameters loaded are stored in non-volatile memory and are used until the function is repeated.

| CPRB Field | Content/Description |
|---|---|
| Function code | LP |
| Request DATA length | 16 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 4 | 1 | Flag5. Length of PIN to be checked. '0' to 'F' in character form. ('0' means check 1 byte, '1' means check 2 bytes, and so on.) |

### Request DATA values

The first 16 bytes of Request DATA contain a 16-byte decimalization table.

## Open PIN pad (OP function)

This function opens the 47xx PIN pad device driver or the MSR device driver. If the PIN pad component is open, it informs the application program if the 47xx PIN pad is in encryption mode. You must use it before you request any other functions of this server except EC.

The 47xx PIN pad or MSR is assigned to the application program that requested it. Subsequent requests from other application programs are rejected with a *busy* return code.

To allow the shared use of this server, every application program must release it, with the CL function, immediately after the execution of the desired task.

| CPRB Field | Content/Description |
|---|---|
| Function code | OP |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 1 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1. Component selection<br>**'B'** Open both the MSR and PIN pad sessions<br>**'P'** Open the PIN pad session<br>**'S'** Open the MSR session<br>Default is to open the PIN pad session |
| 1 | 1 | Flag2 MSR session<br>**'K'** Restore the values set in the last DV request<br>Default is not to open the MSR session |

In LANDP for OS/2, OP supports a flag 'K' in the Request PARMLIST that keeps unchanged the device parameters, as loaded by a DV function request. If 'K' is not specified, the parameters are set to their default values. In LANDP for DOS, the device

parameters are not modified by the OP function. However, for LANDP for DOS, LANDP for OS/2, LANDP for Windows, and LANDP for Linux, you should use the DV function to set the parameters after each OP if this server is shared between applications that need different DV values.

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1 setting:<br>**'E'** encrypted mode<br>**'N'** non-encrypted mode |

## Read from the IBM 47xx PIN pad (RD function)

This function reads information into Reply DATA from the 47xx PIN pad or MSR buffer. Keying in your PIN is normally completed by pressing the END key on the pad. The application program can also cancel the previous AR function by requesting a KL function.

The data obtained from this function depends on the type of the previous AR function.

The data read from the IBM 4778 is from the last component (MSR or PIN pad) armed.

| CPRB Field | Content/Description |
|---|---|
| Function code | RD |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | Length of data to be read |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of data actually read |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1. Data selection<br>**'P'** Read the PIN pad data<br>**'S'** Read the MSR data<br>Default is to read the PIN pad data |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1. Type of data:<br>**'N'** Non-encrypted PIN pad data<br>**'C'** Encrypted and 4700 compatibility mode data (only available in LANDP for DOS)<br>**'E'** Encrypted PIN pad data<br>**'M'** Master key PIN pad data<br>**'V'** Verify PIN block data<br>**'O'** Create PIN offset data<br>**'S'** MSR data |

| Reply DATA Values | | |
|---|---|---|
| **Type** | **Offset** | **Content** |
| N | 0—31 | Non-encrypted data stream |
| C | 0—23 | Encrypted and 4700 Compatibility |
| E | 0—7 | Encrypted PIN pad |
| M | 0—7 | Triple encryption of the device serial number with the loaded key |
| V | | No data returned |
| O | 0—7 | Offset data. Up to 16 BCD digits padded with X'F' if less than 16 digits checked |
| S | 0—n | MSR data. See “Input data formats” on page 216 for further information. |

## Read serial number (RN function)

This function is used to read the serial number stored internally in the PIN pad.

| CPRB Field | Content/Description |
|---|---|
| Function code | RN |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 8 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 8 |
| Replied PARMLIST length | 0 |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Machine type (X'4718' or X'4778') |
| 2 | 1 | Plant of control |
| 3 | 4 | Serial number |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 7 | 1 | Serial set flag |

## Verify message authentication code (VA function)

This function is used to verify a message authentication code (MAC). The MAC is the last four or eight bytes of the data passed. If only four bytes are passed, they are compared with the four leftmost bytes of the MAC internally computed on the preceding data bytes.

| CPRB Field | Content/Description |
|---|---|
| Function code | VA |
| Request DATA length | 12 to 1024 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Not used |
| 1 | 1 | Flag2 Variant table descriptor<br>**'0'** Variants not used<br>**'6'** For fixed variant 06 |
| 2 | 1 | Not used |
| 3 | 1 | Flag4 Key format<br>**'1'** Master key<br>**'2'** Key identifier<br>**'3'** 8-byte key data field |
| 4 | 1 | Flag5 Initial chaining value format<br>**'Y'** Initial chaining value in data<br>**'N'** Internal ICV |
| 5 | 1 | Flag6 Variants used to decrypt ICV<br>**'0'** Variants not used<br>**'2'** Fixed variant 02 |
| 10 | 8 | • Key1 encrypted under the appropriate variant of the master key (if flag4 = '3'), in hexadecimal<br>• If flag4 = '2', it contains a 1-byte identifier, see function LK on page 242 |
| 18 | 8 | ICV encrypted under the appropriate variant of the master key (if flag5 = 'Y') |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 to 1024 | Message with MAC to be verified |

## Write display (WD function)

This function writes data to the liquid crystal display (LCD) on the IBM 4778 PIN Pad Unit. (The LCD is a single-line 16-character display.) The last character is overwritten when the device is armed. In LANDP for OS/2 workstations, closing the PIN pad session causes the display to be cleared.

| CPRB Field | Content/Description |
|---|---|
| Function code | WD |
| Request DATA length | Length of data to be written (0 to 16) |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

### Request DATA values

Contains the data to be written.

## Event notification

When data is available from the 47xx PIN pad (including the 4778 MSR component) and the application program is idle, after having requested the AA or WM function, control is returned to the application program indicating that there is data to be read. The application program receives the Reply PARMLIST, which contains:

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | RD or RS (for the IBM 4778 MSR component) |
| 2 | 8 | PINP47##, where ## is replaced by the session ID |

The application program must then issue an RD function request to obtain the data.

## Migration considerations

The “data encrypted in 4700 compatibility” option is not provided in LANDP for OS/2, because this function is not available in the OS/2 47xx device driver.

The PIN pad server now includes all the functions of the MSR/E server. If you want to migrate applications from the MSR/E server to the PIN pad server, change the server

name in the CPRB, and check the settings of flags 1 and 2 in the Request PARMLIST field. The advantages of making this change are:

- To gain access to the new LCD functions supported in the PIN pad server
- To avoid having to load both servers, if you are using both sets of functions
- To be able to use the IBM 4777 and IBM 4778 devices at the same time

# Part 4. Data management servers

The LANDP family offers several data management servers to satisfy the need to store, retrieve, and update data handled in a branch office.

This part of the book has the following chapters:

**Chapter 14, “Shared-file server” on page 255**

The shared-file server provides database support for a LANDP workgroup. It contains shared files that can be accessed using the *direct*, *sequential*, *indexed*, *indexed sequential*, or *direct indexed* access method.

The shared-file server ensures data integrity. It locks variable length records (up to 26 KB long) when they are retrieved for update. It allows a simultaneous read-only access and logs transactions for recovery purposes. Multiple shared-file servers can be run in a LANDP workgroup. The shared-file server requires a hard disk.

This chapter describes the functions provided by the shared-file server for servicing requests originating in application programs or other servers. You can also find programming notes that you should consider when designing your application programs.

The chapter also describes the shared file distributor and replicator servers which use the same API.

**Chapter 15, “Query server” on page 301**

The query server provides support for application programs to access SQL databases. It extends the functions of IBM Database Server for OS/2 Warp, Version 4.0 and of the OS/400 databases.

The LANDP workgroup can contain multiple query servers each of them requiring a hard disk for data storage. The query server offers two interfaces for the application program:

- Shared-file server interface: This interface provides compatibility with the shared-file server program interface.
- SQL dynamic interface: This interface provides access to a database using standard SQL syntax.

The SQL dynamic interface has important characteristics that offer several advantages:

© Copyright IBM Corp. 1992, 2002

- Application programs running under LANDP for DOS in the same LANDP workgroup can also use the SQL dynamic interface. This allows portability of LANDP application programs.
- A utility program is provided to convert the format of the shared files to the SQL table structure.
- Other—already available—application programs can continue to access the database system, apart from LANDP for OS/2.
- Data integrity is provided by the database system, offering transactional integrity and roll-back functions. LANDP for OS/2 provides an additional logging process to ensure data recovery from a backup copy.

This chapter presents the functions provided by the query server for servicing requests originating in application programs or other servers. You can also find programming notes that you should consider when designing your application programs.

**Chapter 16, “ODBC query server” on page 353**

The ODBC query server, which is currently supported only on the LANDP for Windows platform, provides access to a number of Relational Database Management Systems (RDBMs) and provides database services to application programs that use SQL (Structured Query Language).

This chapter outlines the main objectives of the ODBC query server, describes its modes of operation, and gives details of the control blocks and other data structures that you need to use.

**Chapter 17, “Electronic journal server” on page 379**

The electronic journal server provides a convenient way to store transaction data of a LANDP workgroup during a specified time period. Data can be added, retrieved, updated, and deleted. At the specified end of the journalizing period, the data set is saved and the original space is released for reuse.

This chapter describes the functions provided by the electronic journal server for servicing requests originating in application programs or other servers. You can also find programming notes that you should consider when designing your application programs.

**Chapter 18, “Store-for-forwarding server” on page 405**

The store-for-forwarding server helps to improve the

availability of your LANDP workgroup. It stores transaction data originated in the LANDP workgroup that cannot be sent to the host because, for example, a temporary failing host connection.

You can also use the store-for-forwarding server when the host connection is running properly. It allows you to send stored records at once or when certain conditions are met.

This chapter presents the functions provided by the store-for-forwarding server for servicing requests originating in application programs or other servers. You can also find programming notes that you should consider when designing your application programs.

There is also a forwarding server which complements the store-for-forwarding server. It retrieves the data stored by that server and communicates session status. To send the data, the forwarding server requests services from the SNA communication server. The data transmission is performed using the SNA/SDLC, SNA/Token-Ring, or SNA/X.25 communication server.

Depending on the customization parameters you choose the forwarding server transmits designated data sets, either initiated by an application program or automatically at periodical intervals.

The forwarding server can send data, at the same time as the store-for-forwarding server is adding data.

# Chapter 14. Shared-file server

Using the shared-file server, application programs can access records in shared files. Many application programs can read the same file, but only one can lock and change a record. Shared files can be accessed in five ways:

- Direct access
- Sequential access
- Indexed access
- Indexed sequential access
- Direct indexed access

If you have existing applications that use the shared-file server and you want to use the query server, see the “Migrating from the shared-file server to the LANDP for OS/2 query server.” section in the “Clients and servers” chapter of the *LANDP Programming Guide*.

*Table 34. Function Codes used in the Shared-File Server*

| Function code | Description | Env. | Page |
|---|---|---|---|
| BT | Begin transaction | 02NL | 264 |
| CB | Close batch | 02NL | 264 |
| CO | Close on-line | 02NL | 265 |
| CP | Checkpoint | 02NL | 265 |
| CS | Close session | 02NL | 266 |
| DL | Delete record | 02NL | 266 |
| ET | End transaction | 02NL | 267 |
| EX | Request exclusive use | 02NL | 268 |
| FN | Fetch next record | 02NL | 268 |
| FP | Fetch previous record | 02NL | 269 |
| FU | Fetch unique record | 02NL | 271 |
| GF | Grant | 02NL | 272 |
| GN | Get next record | 02NL | 273 |
| GP | Get previous record | 02NL | 274 |
| GU | Get unique record | 02NL | 275 |
| HL | Habilitate (enable) logging | 02NL | 276 |
| HN | Hold next record | 02NL | 276 |
| HP | Hold previous record | 02NL | 277 |
| HU | Hold unique record | 02NL | 278 |
| ID | Initialize DBD | 02NL | 280 |
| IL | Inhibit logging | 02NL | 281 |
| IP | Initialize pointers | 02NL | 281 |
| IS | Insert record | 02NL | 282 |
| KN | Keep next record | 02NL | 283 |
| KP | Keep previous record | 02NL | 284 |
| KU | Keep unique record | 02NL | 284 |
| OB | Open batch | 02NL | 286 |
| OO | Open on-line | 02NL | 286 |
| OS | Open session | 02NL | 287 |

© Copyright IBM Corp. 1992, 2002

*Table 34. Function Codes used in the Shared-File Server (continued)*

| Function code | Description | Env. | Page |
|---|---|---|---|
| QP | Query PCB | 02NL | 287 |
| RB | Rollback | 02NL | 288 |
| RF | Revoke | 02NL | 289 |
| RH | Read header | 02NL | 290 |
| RP | Replace record | 02NL | 292 |
| SR | Statistic request | 02NL | 293 |
| TS | Test status | 02NL | 294 |
| ZD | Erase shared-file data | 02NL | 295 |

The first and second columns give the function code and the name of the function. The third column shows the operating environment of the function, as explained in "Operating environments" on page xxvi. "02NL" means that it is available from LANDP for DOS, LANDP for OS/2, LANDP for Windows, and LANDP for Linux servers. The last column refers to the page where you can find the function described.

## Shared-file server structure

There can be many shared-file servers installed in a LANDP workgroup, but only one in any one particular workstation. A detailed structure of each shared-file server is defined during customization.

## Shared-file description

Each shared-file server can contain several shared files. The characteristics of the individual shared files are defined at customization in a table called the shared-file *database description* (DBD) file.

The DBD file record contains:

- The record length (maximum of 26624 bytes)
- File name (maximum of eight characters)
- Number of keys:
  - Up to 15 used for the indexed and indexed sequential access methods
- Record split size (see "Variable length records" on page 260)
- Flag to determine if splitting of logical records allowed or not (see "Variable length records" on page 260)
- Key characteristics (only for the indexed access method):
  - Key name
  - Key starting offset
  - Key length, in the range 1 through 255 bytes
  - Key type (unique or not, can be changed or not)
  - Key null or not. Keys and segments of keys can be made null. See the related PCB key information in the description of "Read header (RH function)" on page 290 for more information about null keys.

- Key segmentation. A key is segmented if it is formed of parts from different locations in a record. Up to 32 segments can form a key. Each segment of a key has these attributes:
  - The kind of segment: data (containing valid user data), and indicator (showing whether the next segment is valid, and whether the global key is in the index file—such segments do not form part of the key)
  - The length and offset
  - The collation type
  - The null value, if it can be made null
- Key modifiable or nonmodifiable
- Key must be unique or nonunique
- Key collated or uncollated
- Key DBCS or SBCS With DBCS keys, every character occupies two bytes. DBCS characters are treated as such. SBCS characters are preceded by X'00' and the second byte of the key is the SBCS character itself. Therefore, when you sort DBCS keys, SBCS characters come first.

For more information about how to define key characteristics, see the *LANDP Installation and Customization* book.

Every defined DBD results in one data file. One *logical record* consists of one or more *physical records* which result when the logical record is split into physical records, see “Variable length records” on page 260 for details. Every key defined results in one index file. The number of files that the shared-file server can manage is limited only by the amount of available disk space.

To improve the performance, the server can allocate buffers in memory for the index during loading time. Up to 484 index pages can be allocated.

## Program control block

The table of *program control blocks* (PCBs) is also defined during customization. It is a file with the following structure:

```
PCB name 1   Associated DBD name   Key name
PCB name 2   Associated DBD name   Key name
PCB name 3   Associated DBD name   Key name
  .            .                     .
  .            .                     .
```

The “Key name” field is not used for the direct and sequential access methods. Requesting services from the shared-file server requires that you specify the PCB name, *except for the following functions*:

BT, CB, CO, CP, CS, ET, GF, HL, IL, IP, OB, OO, OS, RB, RF, SR, and TS.

## Initialization of the shared-file server structure

All DBD definitions reside in one file that is loaded into storage when the server is loaded. The content is compared with duplicate information contained in record 0 of each shared file.

All the PCBs are also stored in one file. The DBD file and the PCB file are collectively known as a shared-file profile. The shared-file server supports only one profile each time it is started. There can be a .SEQ file that defines the collating sequence for the indexed fields.

## Shared-file distributor server

For extended database management, the **LANDP for OS/2** shared-file distributor server manages data that is located in different machines as a single database. Each part of the database is managed by shared-file servers in different machines.

This server provides for data integrity at the transaction level, error recovery, and the management of deadlocks.

Integrity and consistency of the data are provided through a “two-phase commit” process that is transparent to the application. As a first stage, all involved shared files are asked to be prepared to commit data. When a positive response is received from all of them, the data is committed. This process takes place when the shared-file distributor server receives requests to close a session (CB and CO), to checkpoint a transaction (CP), to end a transaction (ET), and to erase shared-file data (ZD).

The shared-file distributor server is able to operate if at least one of the shared-file servers is up and running. This means that it is able to work on a partial database. If an attempt is made to read or write on a part that is not active, an error code is returned to the application, which has the responsibility of deciding if the entire transaction is to take place or not. In this way, maximum flexibility is given to the application program.

Where a distributed database is used, all write operations must be done through the shared-file distributor server. However, read operations can be done through each shared-file server. When you design your applications, you should ensure that different shared files contain disjoint information. Program control blocks (PCBs) must not be duplicated among the shared files that are used by a shared-file distributor server.

The shared-file distributor server has the same interface as the shared-file server, allowing applications to treat the server as a shared-file server that manages a single database. Requests for functions such as grant (GF) are passed to all shared files that are loaded. Others like rollback (RB) are passed to all shared files that are involved in the transaction. Others like get (GN, GP, and GU), hold, and keep are issued to the shared file that has the requested PCB.

The server runs in LANDP for OS/2, and can manage shared-file servers that operate in LANDP for OS/2 .

## Shared-file replicator server

For the high availability of data, the **LANDP for OS/2** shared-file replicator server maintains exact copies of the same database in different machines. Each of these databases is managed by a shared-file server. Program control blocks (PCBs) must be duplicated among the shared files that are used by a shared-file replicator server.

The shared-file replicator server can operate if at least one up-to-date database is running. If an error happens to one of the databases while work progresses, this database is discarded and operations continue with the rest. An error notification is passed to the application program only when all databases are in error or not loaded.

If a database that is not up-to-date is loaded, the shared-file replicator server automatically performs the synchronization.

The shared-file replicator server has the same interface as the shared-file server, allowing applications to treat the server as a shared-file server that manages a simple database. Requests to habilitate (enable) and inhibit logging (HL and IL) have no effect when made to the shared-file replicator server.

All write operations must be done through the shared-file replicator server. However, read operations can be done through any of the shared-file servers. Read operations that take place through the shared-file replicator server result in the updating of the file pointers for all the shared files.

The server runs in LANDP for OS/2, and can manage shared-file servers that operate in LANDP for OS/2.

## Log files

The shared-file server uses the log files, LOG.DAT and LOG2.DAT, to store information about the changes made to the data files. All the functions that involve changes (deletion, update, or insertion) to the data files are stored as log records. All records related to one transaction are chained, so that the shared-file server can identify the functions and the sequence in which they were performed for each single transaction.

The log files ensure integrity at the transaction level:

- At server loading time, the shared-file server scans the log files for any open transactions that were neither committed nor rolled back in the previous working session. If the shared-file server finds any of these transactions, an automatic back-out of these transactions takes place.
- At server run time, it is possible to rollback any transaction.

The log files also enable forward recovery. If data files are lost or damaged, it is possible to recover the data by using a backup of the data files and the log files that include all the changes done to the data files since that backup. The shared-file server reads the log files records and performs the logged functions again.

For further information concerning the various types of log mechanism and the use of log files, see the *LANDP Servers and System Management* book.

## Variable length records

Variable length records allow you to use disk space in an optimal way. The data record length (the *logical* record length) can be split into several *physical* records.

For example, a logical record with a maximum length of 1024 bytes can be split into physical records of 128 bytes each. Then a record of this type whose *actual* length is 200 bytes occupies two physical record areas of only 256 bytes, and not 1024 bytes.

The capability of using variable length records obviously leads to a much more efficient use of secondary storage. Each DBD record belonging to the DBD profile has to be updated to include the two fields that define the *split size* of each logical record (which is decided by you), and the flag which shows whether a logical record is split-typed or not.

## Shared-file server operating modes

The shared-file server can operate in three modes: closed, on-line, and batch. Depending on the mode of the server, the request syntax for the shared-file server varies. Some functions are not allowed, depending on the mode of the server.

**Closed** The shared files and log file are closed. The only functions allowed are GF (grant), TS (test status), and SR (statistic request).

**On-line** On-line processing lets you lock at the record or shared-file level. This status implies:

- OO (open on-line) and RF (revoke) are always accepted.
- OB (open batch), CB (close batch), and GF (grant) are never accepted.
- CP (checkpoint), EX (request exclusive use), IS (insert record), HN (hold next record), HP (hold previous record), HU (hold unique record), KN (keep next record), KP (keep previous record), KU (keep unique record), and RB (rollback) are accepted only after a BT (begin transaction) function.
- ZD (erase shared-file data) is accepted only after an EX (request exclusive use) function against the referenced DBD.
- DL (delete record) and RP (replace record) are accepted after a hold function (HN, HP, or HU), or a keep function (KN, KP, or KU).
- Other functions can be requested but only after an OO (open on-line) function.

**Batch** The batch mode implies locking at the shared-file level. Here:

- OB (open batch) and RF (revoke) are always accepted.
- OO (open on-line), BT (begin transaction), ET (end transaction), and GF (grant) are never accepted.

- For the remaining functions, the workstation must issue OB (open batch) first.
  - CB (close batch), CP (checkpoint), RF (revoke), and RB (rollback) are always accepted.
  - EX (request exclusive use) is accepted if the DBD through which the request is made (through any of its PCBs) is not being used exclusively by any of the workstations, or when it is being used by the requesting workstation.
  - HN, HP, HU (hold functions), DL (delete record), IS (insert record), KN, KP, KU (keep functions), and RP (replace record) are accepted only when the DBD is blocked by the requesting terminal that has used function EX (request exclusive use).

Exceptions are the TS (test status) and SR (statistic request) control functions which can be requested by any workstation in any mode. Use the TS function to check the server status, see page 294.

## File locking in on-line mode

For major updates file locking is available when working in on-line mode. If the EX function is used, after opening a transaction with the BT function, the shared file is locked (on the DBD level). Unlocking is made by requesting an ET function. A workstation disconnect message has the same effect. The CP function does not unlock the shared file, it releases only record level locks.

The error code BL is issued if an EX, HU, HN, HP, or IS function request is requested by another workstation when the shared file is locked.

File locking in on-line mode should only be used when absolutely needed. It increases the contention in the LANDP workgroup.

## Using the shared-file server

This section provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields. If you need more information about the CPRB fields, see Appendix A, “Connectivity programming request block” on page 669.

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | 26 | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 26 | 2 | 26 | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | SHFILE##<br>EHCSFD##<br>EHCSFR## | Server name (shared-file server, shared-file distributor server, or shared-file replicator server) |

The following fields are variable and are discussed in each function request description:
- Function code
- Request DATA length
- Reply DATA length

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 44 | 2 | | Replied PARMLIST length |
| 46 | 2 | | Replied DATA length |

If the request was successful, the *router return code* and the *server return code* are both X'00000000'. If the server return code is nonzero see the appropriate section in the *LANDP Problem Determination* book to see if you should take any action.

The following fields are variable and are discussed in each function request description:
- Replied PARMLIST length
- Replied DATA length

**PARMLIST:** The Request PARMLIST and Reply PARMLIST fields for the shared-file server are:

| Offset | Length | Content |
|---|---|---|
| 0 | 2 | Comparison Operator: use one of the following codes to specify how the key or record number is to be used:<br>**'EQ'** Search for the key or record number specified<br>**'GE'** Search for the first key or record number that is greater than or equal to the one specified<br>**'GT'** Search for the first key or record number that is greater than the one specified<br>**'NE'** Search for the first key or record number that is not equal to the one specified<br>**'LE'** Search for the first key or record number that is less than or equal to the one specified<br>**'LT'** Search for the first key or record number that is less than the one specified<br>**'PR'** Search for the record returned by the last Hold, Get or Keep request, with the same PCB (see note) |
| 2 | 8 | PCB name |
| 10 | 2 | Session identifier |
| 18 | 4 | Record number (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |
| **Note:** The 'PR' operator retrieves the last record accessed by the application. If a record is retrieved using the PCB without holding, then another process can modify or delete the same record. If this occurs and if the application program then requests a Get, Hold, or Keep function with 'PR' operator using the same PCB, a GE (record not found) code is returned. The record must then be re-read to determine if it has been changed. Also note that when an IP (initialize pointers), ID (initialize DBD), or ET (end transaction) function is requested, a FU (fetch unique record), GU (get unique record), HU (hold unique record), or KU (keep unique record) function with 'PR' operator also results in a GE return code. | | |

Not all fields are needed for all functions. The use of these fields is explained in the description of each function request.

**DATA:** Request DATA and Reply DATA contain data to be sent to or received from the server. The use of these areas is explained in the description of each function request.

**Note:** The client program must provide sufficient space in Reply DATA to receive correctly. It is recommended that you allocate the maximum size of 26624 bytes (X'6800') for your DATA areas.

The session identifier for the main session is always X'0000'. If you operate multiple sessions make sure that you always specify the appropriate session identifier when requesting services. For further information on operating multiple sessions, see “Open session (OS function)” on page 287.

## Request reference

These are the functions that can be used with the shared-file server. They are listed in alphabetical order of function code.

## Begin transaction (BT function)

A shared-file transaction is a set of related shared-file functions. Use the BT function before you begin a transaction. You can cancel a transaction with the rollback (RB) function any time during the transaction, until you issue either an end transaction (ET) or checkpoint (CP) function. The ET function completes the execution of the transaction and confirms that the changes have been made. The CP function only confirms the changes. This function can only be used in on-line mode.

If the request is issued for a shared-file distributor server, the request is issued to the shared-file servers when they are needed. That is, a BT is issued by the shared-file distributor server when a function requires the use of one of the shared-file servers.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | BT |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

## Close batch (CB function)

This function ends a batch session started with open batch (OB). It commits all changes and releases the shared file.

If the request is issued for a shared-file distributor server, the request is issued to all shared-file servers that have batch access for the session. If there are open transactions, the two-phase commit process is used to close them.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

**Note:** Remember to issue a CB function request to close the batch session before you issue an RF (revoke) function request to close the shared-file server.

| CPRB Field | Content/Description |
|---|---|
| Function code | CB |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

## Close on-line (CO function)

This function ends an on-line session started with open on-line (OO). It commits all changes if there is an open transaction for the session, and releases the shared file.

If the request is issued for a shared-file distributor server, the request is issued to all shared-file servers that have online access for the session. If there are changes to be committed, the two-phase commit process is used.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | CO |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

## Checkpoint (CP function)

This function validates all the operations performed in the present transaction. It frees all records that are being held because of changes, and the deleted record space is freed and is added to the chain of free records for later use. The transaction remains open and the workstation can continue using shared-file functions until an ET (end transaction) function is requested. Also see the RF (revoke) function.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

If the request is issued for a shared-file distributor server, the request is issued to all shared-file servers involved in the transaction. The two-phase commit process is used.

| CPRB Field | Content/Description |
|---|---|
| Function code | CP |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

## Close session (CS function)

This function closes a session previously opened with an OS (open session) function request. Specify the session to be closed in the Request PARMLIST area. It can be requested at any time but if it is requested and a transaction is open, an automatic rollback is performed before the session is closed.

If the request is issued for a shared-file distributor server, the request is issued to all shared-file servers having this session opened. If necessary, changes are rolled back for all of them.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | CS |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

## Delete record (DL function)

This function deletes the current record based on the PCB name. You must first reserve the record for exclusive use with a hold function request (HN, HP, or HU) or a keep function request (KN, KP, KU).

The space reserved for this record is returned to the free chain of records when the transaction is validated with a CB (close batch), CP (checkpoint), or ET (end transaction) function request.

If the request is issued for a shared-file distributor server, the request is issued to the shared-file server that has the record, as identified by the PCB name.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | DL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 256 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Not used |
| 2 | 8 | PCB name |

## End transaction (ET function)

This function ends the transaction started by a BT (begin transaction) function request. The ET function:

- Validates all completed work
- Frees all records that are being held because of changes
- Puts the deleted record(s) in the chain of free records for later use
- Resets the application program file pointers
- Ends the transaction

You must issue a BT function request if you want to begin another transaction.

This function is valid only in on-line mode.

**Note:** Remember to issue an ET function request to close the on-line transaction before you close the shared-file server with RF (revoke).

If the request is issued for a shared-file distributor server, the request is issued to all shared-file servers that are involved in the transaction. The two-phase commit process is used.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | ET |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 256 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

## Request exclusive use (EX function)

This function requests exclusive use of a DBD. The DBD is released at the end of the transaction or batch session.

If the request is issued for a shared-file distributor server, the request is issued to the shared-file server that has the DBD, as identified by the PCB name.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | EX |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Not used |
| 2 | 8 | PCB name used to access the DBD |

## Fetch next record (FN function)

This function reads the next record into Reply DATA.

The function is the same as GN, and also unlocks the record if it has not been modified. If the record has been modified, it is not unlocked, even if it belongs to a transaction that does not log movements.

This function reads the first record in the shared file:

- If the shared-file addressing pointers do not point to any specific record in the file, because there have not been any read/write activities
- If, after an initialize pointers IP (or initialize DBD, ID for the same DBD), ET (end transaction), or RB (rollback) were requested
- If you received a GB return code, having reached the end of the shared file, the next FN function reads the first record in the shared file.

If the request is issued for a shared-file distributor server, the request is issued to the shared-file server that has the record, as identified by the PCB name.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | FN |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | The expected length of the record to be read |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of record read |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Not used |
| 2 | 8 | PCB name |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 to 26624 | Next data record of shared file |

## Fetch previous record (FP function)

This function reads the previous record into Reply DATA.

The function is the same as GP, and also unlocks the record if it has not been modified. If the record has been modified, it is not unlocked, even if it belongs to a transaction that does not log movements.

This function reads the last record in the shared file:

- If the shared-file addressing pointers do not point to any specific record in the file, because there have not been any read/write activities
- If, after a reset pointer IP (or initialize DBD, ID, for the same DBD), ET (end transaction), or RB (rollback) were requested
- If you received a GB return code, having reached the beginning of the shared file, the next FP function reads the last record in the shared file.

If the request is issued for a shared-file distributor server, the request is issued to the shared-file server that has the record, as identified by the PCB name.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | FP |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | The expected length of the record to be read |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of record read |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Not used |
| 2 | 8 | PCB name |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 to 26624 | Previous data record of shared file |

## Fetch unique record (FU function)

This function reads a record according to the key stored in Request DATA, or the record number in Request PARMLIST. The return code is GE if the selected record does not exist.

The function is the same as GU, and also unlocks the record if it has not been modified. If the record has been modified, it is not unlocked, even if it belongs to a transaction that does not log movements.

If the request is issued for a shared-file distributor server, the request is issued to the shared-file server that has the record, as identified by the PCB name.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | FU |
| Request DATA length | **0** Direct indexed access<br>**4** Direct access<br>**Key length:** Indexed sequential access |
| Request PARMLIST length | 26 |
| Reply DATA length | Expected length of record to be read |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of record read |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Comparison operator (see page 263) |
| 2 | 8 | PCB name |
| 18 | 4 | Record number (for direct indexed access) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 or<br>4 or<br>key length | Nothing for direct indexed access<br>Record number for direct search access<br>Record key for index sequential search access |

For direct indexed access, the returned record might not have any key through this PCB. In that case, the current instance of the PCB is not modified, and the addressing pointer keeps pointing to the same record, if any, in the file as it was before the call.

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 to 26624 | Data record of shared file |

**Note:** To maintain compatibility with IBM Financial Branch System Services, a Request DATA length value of 2 is also accepted for direct access. Record number is not byte-reversed (COBOL format) in that case.

## Grant (GF function)

This function opens the shared-file server as follows:

- Opens the log file and tests for a change in the log file
- Loads log file header information
- Opens the shared files

At the start of the daily operations, the appropriate workstation should issue this function request once for the entire LANDP workgroup.

If the request is issued for a shared-file distributor server or a shared-file replicator server, the request is issued to all associated shared-file servers that are loaded. If an associated shared-file server is loaded later, the request is automatically made to it.

If a shared-file distributor or replicator server is used, the GF request cannot be made to an associated shared file, but must be done through the distributor or replicator server.

| CPRB Field | Content/Description |
|---|---|
| Function code | GF |
| Request DATA length | 1 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | 'O' (X'4F') = Shared file used in on-line mode<br>'B' (X'42') = Shared file used in batch mode |

## Get next record (GN function)

This function reads the next record into Reply DATA. This function reads the first record in the shared file:

- If the shared-file addressing pointers do not point to any specific record in the file, because there have not been any read/write activities
- If, after an initialize pointers IP (or initialize DBD, ID, for the same DBD), ET (end transaction), or RB (rollback) were requested
- If you received a GB return code, having reached the end of the shared file, the next GN function reads the first record in the shared file

If the request is issued for a shared-file distributor server, the request is issued to the shared-file server that has the record, as identified by the PCB name.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | GN |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | The expected length of the record to be read |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of record read |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Not used |
| 2 | 8 | PCB name |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 to 26624 | Next data record of shared file |

## Get previous record (GP function)

This function reads the previous record into Reply DATA. This function reads the last record in the shared file:

- If the shared-file addressing pointers do not point to any specific record in the file, because there have not been any read/write activities
- If, after a reset pointer IP (or initialize DBD, ID, for the same DBD), ET (end transaction), or RB (rollback) were requested
- If you received a GB return code, having reached the beginning of the shared file, the next GP function reads the last record in the shared file.

If the request is issued for a shared-file distributor server, the request is issued to the shared-file server that has the record, as identified by the PCB name.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | GP |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | The expected length of the record to be read |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of record read |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Not used |
| 2 | 8 | PCB name |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 to 26624 | Previous data record of shared file |

## Get unique record (GU function)

This function reads a record according to the key stored in Request DATA, or the record number in Request PARMLIST. The return code is GE if the selected record does not exist.

If the request is issued for a shared-file distributor server, the request is issued to the shared-file server that has the record, as identified by the PCB name.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | GU |
| Request DATA length | **0** Direct indexed access<br>**4** Direct access<br>**Key length:** Indexed sequential access |
| Request PARMLIST length | 26 |
| Reply DATA length | Expected length of record to be read |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of record read |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Comparison operator (see page 263) |
| 2 | 8 | PCB name |
| 18 | 4 | Record number (for direct indexed access) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 or<br>4or<br>key length | Nothing for direct indexed access<br>Record number for direct search access<br>Record key for index sequential search access |

For direct indexed access, the returned record might not have any key through this PCB. In that case, the current instance of the PCB is not modified, and the addressing pointer keeps pointing to the same record, if any, in the file as it was before the call.

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 to 26624 | Data record of shared file |

**Note:** To maintain compatibility with IBM Financial Branch System Services, a Request DATA length value of 2 is also accepted for direct access. Record number is not byte-reversed (COBOL format) in that case.

## Habilitate (enable) logging (HL function)

This function allows the logging of transactions made with the shared-file server. HL complements the IL function, see “Inhibit logging (IL function)” on page 281. The HL function is used only after logging has been disabled by the IL function. The HL function can be requested on an open shared file (using GF), when no transactions are pending (through OO and BT, or by OB). The RB function cannot be requested while the logging of transactions has been disabled.

If the request is issued for a shared-file distributor server the request is issued to all associated shared-file servers that have this session open. If the session is later opened for an associated shared-file server, the request is automatically made to it.

If the request is issued for a shared-file replicator server, the request is ignored.

| CPRB Field | Content/Description |
|---|---|
| Function code | HL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

## Hold next record (HN function)

This function works in the same way as GN (get next record), but obtains the exclusive use of the record. It frees the exclusive use of the last record called by the PCB, except when the last record was changed. If the accessed record was changed, it is not released until the transaction has ended.

If the selected record is held by a different PCB for the same user, the record is obtained and the hold is released.

If you receive the return code GB, you have reached the end of the shared file. A GET function that uses a PCB frees the record that was held by a previous HOLD function with the same PCB. The next HN function reads the first record in the shared file.

If the request is issued for a shared-file distributor server, the request is issued to the shared-file server that has the record, as identified by the PCB name.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | HN |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | Expected length of record to be read |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of record read |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Not used |
| 2 | 8 | PCB name |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 to 26224 | Next data record of shared file |

## Hold previous record (HP function)

This function works in the same way as GP (get previous record) but obtains the exclusive use of the record. It releases the exclusive use of the last record requested by the PCB, if the last record has not been changed. Otherwise, the last record is not released until the transaction has ended.

If the selected record is being held by a different PCB for the same application program, the hold is released and the record is obtained.

A GET function that uses a PCB frees the record that was held by a previous HOLD function with the same PCB.

If you receive the return code GB, you have reached the first record in the shared file. The next HP function reads the last record in the shared file.

If the request is issued for a shared-file distributor server, the request is issued to the shared-file server that has the record, as identified by the PCB name.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | HP |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | Expected length of record to be read |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of record read |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Not used |
| 2 | 8 | PCB name |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 to 26624 | Previous data record of shared file |

## Hold unique record (HU function)

This function obtains exclusive use of a specific record, according to the key stored in Request DATA, or the record number in Request PARMLIST. It frees the exclusive use

of the last record requested by the PCB, except when the last record has been changed. If the accessed record has been changed, it is not released until the transaction has ended.

If the selected record is held by a different PCB for the same application program, the record is obtained and the hold is released.

A GET function that uses a PCB frees the record that was held by a previous HOLD function with the same PCB.

An ID (initialize DBD) or IP (initialize pointers) function also releases the record if it has not been changed.

If the request is issued for a shared-file distributor server, the request is issued to the shared-file server that has the record, as identified by the PCB name.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | HU |
| Request DATA length | **0** Direct indexed access<br>**4** Direct access<br>**Key length:** Indexed sequential access |
| Request PARMLIST length | 26 |
| Reply DATA length | Expected length of record to be read |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of record read |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Comparison operator (see page 263) |
| 2 | 8 | PCB name |
| 18 | 4 | Record number (for direct indexed access) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 or<br>4 or<br>key length | Nothing for direct indexed access<br>Record number for direct access<br>Record key for indexed sequential access |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 to 26624 | Data record of shared file |

**Note:** To maintain compatibility with IBM Financial Branch System Services, a Request DATA length value of 2 is also accepted for direct access. Record number is not byte-reversed (COBOL format).

For direct indexed access, the selected record might not have any key through this PCB. In that case, the current instance of the PCB is not modified, and the addressing pointer keeps pointing to the same record, if any, in the file as it was before the call.

## Initialize DBD (ID function)

This function initializes a DBD, resetting each PCB pointer for the DBD. The parameter used is PCB Name.

If the request is issued for a shared-file distributor server, the request is issued to the shared-file server that is identified through the PCB name.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | ID |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Not used |
| 2 | 8 | PCB name of the DBD to be initialized |

## Inhibit logging (IL function)

This function stops logging the transactions made with the shared-file server. It is useful for the initial storing of records in shared files, as the throughput is increased. It should be used with caution since the rollback function cannot be used while logging is inhibited, and future forward recovery may not work if data files contain data both logged and not logged.

Before using the IL function request, you must open the shared file with the GF (grant) function request. There should be no pending transactions.

If the request is issued for a shared-file distributor server the request is issued to all associated shared-file servers that have this session open. If the session is later opened for an associated shared-file server, the request is automatically made to it.

If the request is issued for a shared-file replicator server, the request is ignored.

| CPRB Field | Content/Description |
|---|---|
| Function code | IL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

## Initialize pointers (IP function)

This function resets the shared-file addressing pointers to the locations they have when the shared-file server is loaded. Requesting FN, GN, HN, or KN (fetch, get, hold, or keep next record) retrieves the first record of the file. Requesting FP, GP, HP, or KP (fetch, get, hold, or keep previous record) retrieves the last record of the file.

If the request is issued for a shared-file distributor server, then the request is issued to all associated shared-file servers that have this session open.

| CPRB Field | Content/Description |
|---|---|
| Function code | IP |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

## Insert record (IS function)

This function inserts the record from Request DATA into the file specified through the PCB. The record is inserted in the first available place in the file. If there is no available place, the inserted record is appended to the end of the file. The record is locked from other users until you issue a CP (checkpoint) or ET (end transaction) function request to validate the record.

The record pointer in the PCB for this function is set to the position of the inserted record. If you use an FN, GN, HN, or KN (fetch, get, hold, or keep next record) function request after this IS function request, you receive the record that follows this inserted record. If you use an FP, GP, HP, or KP (fetch, get, hold, or keep previous record) function after the IS function request, you receive the record that precedes this inserted record.

If the request is issued for a shared-file distributor server, the request is issued to the shared-file server that is to have the record, as identified by the PCB name.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | IS |
| Request DATA length | Length of record |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Not used |
| 2 | 8 | PCB name |

### Request DATA values

Store the record to be inserted in Request DATA.

**Note:** When you insert a record, the server stores it in the first record in the chain of free records. This chain is in record sequence in an empty shared file. But when a record is deleted and validated, with CB (close batch), CP (checkpoint), or ET (end transaction), its free space is placed in the first position in the chain of free records. You should use sequential insertion only when you initially store records in the shared file.

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

## Keep next record (KN function)

This function works in the same way as HN (hold next record). It maintains the lock on the record for the duration of the transaction until a CP (checkpoint), ET (end transaction), or RB (rollback) function request is made. This function keeps the record held in read mode.

If the request is issued for a shared-file distributor server, the request is issued to the shared-file server that has the record, as identified by the PCB name.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | KN |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | Expected length of record to be read |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of record read |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Not used |
| 2 | 8 | PCB name |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 to 26624 | Next data record of shared file |

## Keep previous record (KP function)

This function works in the same way as HP (hold previous record). It maintains the lock on the record for the duration of the transaction until a CP (checkpoint), ET (end transaction), or RB (rollback) function request is made. This function keeps the record held in read mode.

If the request is issued for a shared-file distributor server, the request is issued to the shared-file server that has the record, as identified by the PCB name.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | KP |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | Expected length of record to be read |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of record read |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Not used |
| 2 | 8 | PCB name |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 to 26624 | Previous data record of shared file |

## Keep unique record (KU function)

This function obtains exclusive use of a specific record, according to the key stored in Request DATA, or the record number in Request PARMLIST, in the same way as HU (hold unique record). It maintains the lock on the record for the duration of the transaction until a CP (checkpoint), ET (end transaction), or RB (rollback) function request is made. This function keeps the record held in read mode.

If the request is issued for a shared-file distributor server, the request is issued to the shared-file server that has the record, as identified by the PCB name.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | KU |
| Request DATA length | **0** Direct indexed access<br>**4** Direct access<br>**Key length:** Indexed sequential access |
| Request PARMLIST length | 26 |
| Reply DATA length | Expected length of record to be read |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of record read |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Comparison operator (see page 263) |
| 2 | 8 | PCB name |
| 18 | 4 | Record number (for direct indexed access) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 or<br>4 or<br>key length | Nothing for direct indexed access<br>Record number for direct access<br>Record key for index sequential access |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 to 26624 | Data record of shared file |

For direct indexed access, the selected record might not have any key through this PCB. In that case, the current instance of the PCB is not modified, and the addressing pointer keeps pointing to the same record, if any, in the file as it was before the call.

## Open batch (OB function)

This function opens a batch session that is characterized by locking at the shared-file level. You must first issue a GF (grant) function request with 'B' (X'42') in Request DATA.

If the request is issued for a shared-file distributor server, the request is issued to associated shared-file servers as needed.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers that are loaded.

| CPRB Field | Content/Description |
|---|---|
| Function code | OB |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

## Open on-line (OO function)

This function opens an on-line session that is characterized by locking at the record or file level. You must first issue a GF (grant) function with 'O' (X'4F') in Request DATA.

If the request is issued for a shared-file distributor server, the request is issued to associated shared-file servers as needed.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers that are loaded.

| CPRB Field | Content/Description |
|---|---|
| Function code | OO |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

## Open session (OS function)

This function can be used at any time to establish an additional working session.

The OS function is primarily useful for environments where two or more application programs (in OS/2, Windows or Linux, threads) run in the same machine (in OS/2, Windows, or Linux, process) on a time slice basis. Using the OS function request, an application program can achieve “database independence” from other application programs. Its shared-file server requests are seen as coming from an “independent logical workstation”.

As the result of a successful OS function request, a session identifier is returned in Reply PARMLIST. This session identifier must be provided in all requests using the additional session. If the normal session should be used, the field “session identifier” in Request PARMLIST must contain blanks or nulls.

**Notes:**

1. The use of additional sessions must be reflected in the /S parameter when loading the shared-file server.
2. Application programs running under different process IDs do not need to issue OS and CS (close session) function requests explicitly.

If the request is issued for a shared-file distributor server, the request is issued to an associated shared-file server when needed.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | OS |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier |

## Query PCB (QP function)

This function can be used to return the names of the PCBs.

If the request is issued for a shared-file distributor server or a shared-file replicator server, the information is obtained from internal tables. However, in the case of the

shared-file distributor server, only the PCBs corresponding to the shared files currently loaded are returned.

| CPRB Field | Content/Description |
|---|---|
| Function code | QP |
| Request DATA length | 0 or 8 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 to 26624 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 to 26624 |
| Replied PARMLIST length | 26 |

| Request DATA Values if Request DATA length is 8 | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | PCB name |

When a PCB name is passed to this function, the server replies with the names of the PCBs after the one named in Request DATA.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | PCB name 1 |
| And so on for each PCB name | | |

The suggested way of using this function is as follows:

```
Issue QP without a PCB name, with Request DATA length = 0
         to get the first PCB
While return code = MI (more information pending)
  Issue QP with last PCB name received,
         Request DATA length = 8
```

## Rollback (RB function)

This function cancels all changes made since the last ET (end transaction) or CP (checkpoint) function request, and closes the active transaction. If you are using batch mode, a CB (close batch) function is performed automatically when you use RB.

**Note:** If you have large transactions that could be cancelled by the rollback function, you might need to change the LAN timeout value. This is done during customization by specifying a longer period for the LAN timeout value.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

If the request is issued for a shared-file distributor server, the request is issued to all shared-file servers involved in the transaction.

| CPRB Field | Content/Description |
|---|---|
| Function code | RB |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

## Revoke (RF function)

This function does a forced or non-forced close of the shared file server.

When the request is received:

- If there is no open transaction, the shared files are closed immediately.
- If there are open transactions, no further OB (open batch) or BT (begin transaction) requests are accepted.

  If the revoking is *non-forced*, when all workstations have completed their processing, using ET (end transaction), CB (close batch), or RB (rollback), the shared files are closed.

  If revoking is *forced*, the shared-file server will automatically rollback every open transaction and the shared files are closed. This type of revoking acts as if each workstation with an open transaction had completed its work by requesting an RB function.

Before you request the RF function to close the server, it is recommended that you request CB or ET to close any off-line or on-line transaction.

When the close begins, the server:

- Closes all the shared files.
- Closes the LOG.DAT file (and LOG2.DAT if used) and updates the log file headers.
- If the /E option was specified when the server was loaded, the server creates a file containing the total number of operations for each workstation in the configuration. The file consists of records having the same structure as those described in the SR (statistic request) function request. There is one record for each workstation in the network. The file is called **mmddhhmn.EST**, where:
  **mm** Month.
  **dd** Day of the month.
  **hh** Closing hour.
  **mn** Closing minute.

If the request is issued for a shared-file distributor server or a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | RF |
| Request DATA length | 0 or 1 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

When Request DATA length is X'0000' a normal close is performed.

| Request DATA Values when Request DATA length is 1 | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | 'N' (X'4E') or ' ' (X'20'): Normal close (*non-forced*) of shared-file server<br><br>'F' (X'46'): Forced close of shared-file server |

## Read header (RH function)

This function provides information about the characteristics of a file associated with a PCB name. You must open the file with an OO (open on-line) or OB (open batch) function request.

If the request is issued for a shared-file distributor server, the request is issued to the shared-file server that has the PCB name defined.

If the request is issued for a shared-file replicator server, the request is issued to one of the associated shared-file servers.

**Note:** In batch mode, the used record count is not updated until an ET record is issued. Therefore, if the shared file is opened using the OB function, an RH request returns (at offset 10 in the reply data values) the number of valid physical records used in the data file associated with the PCB ***prior*** to the current transaction block.

| CPRB Field | Content/Description |
|---|---|
| Function code | RH |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | Up to 26624 |
| Reply PARMLIST length | 26 |
| Replied DATA length | Up to 26624 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 8 | PCB name |

| Replay DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| **Related PCB information** | | |
| 0 | 4 | Physical record positioning of the PCB pointer. After IPL, or IP (initialize pointers), ID (initialize DBD), or ET (end transaction) function, this value is 0. |
| 4 | 2 | Number of updated index pages that are kept in memory for this PCB. This value can change at any moment, as pages are refreshed asynchronously. |
| 6 | 4 | Number of pre-formatted physical records present in the data file. It is the size of the file. |
| 10 | 4 | Number of used records in the data file associated with the PCB. It is the logical file size, in other words the number of valid physical records in the data file. |
| **DBD general information** | | |
| 14 | 2 | DBD number (based on 0) |
| 16 | 2 | Total number of DBDs |
| 18 | 2 | Record length |
| 20 | 2 | Record split size (see “Variable length records” on page 260) |
| 22 | 8 | DBD name |
| 30 | 64 | Full path and file name |
| 94 | 1 | Number of indexes for the DBD |
| **Related PCB key information** | | |
| 95 | 1 | Number of segments |
| 96 | 1 | Key class:<br>**0** Is unique and cannot be changed<br>**1** Need not be unique and can be changed<br>**2** Is unique and can be changed<br>**3** Need not be unique and cannot be changed |

| Replay DATA Values | | | |
|---|---|---|---|
| **Offset** | **Length** | **Content** | |
| 97 | 1 | Type (see note):<br>**0** data segment<br>**1** indicator segment<br>**2** locally nullable<br>**3** globally nullable | First segment |
| 98 | 2 | Length | |
| 100 | 2 | Offset | |
| 102 | 1 | Collating value:<br>**1** Yes<br>**2** No<br>**3** DBCS<br>**Fx** Use PROFILE.SEx for collation | |
| 103 | 1 | Null value (see note) | |
| 104 | 1 | Reserved | |
| 105 | 1 | Type | Second segment |
| 106 | 2 | Length | |
| 108 | 2 | Offset | |
| 110 | 1 | Collating value | |
| 111 | 1 | Null value | |
| 112 | 1 | Reserved | |
| And so on for further segments. | | | |
| **Note:** When any segment whose type is “locally nullable” is filled with its null value, the key is discarded.<br><br>When all segments whose type is “globally nullable” are filled with their null values, the key is discarded.<br><br>The integers are stored as binary numbers in the requester’s native computer format. | | | |

## Replace record (RP function)

This function replaces the last record held with this PCB. Before using this function, you must hold the record with a hold function request (HN, HP, or HU) or a keep function request (KN, KP, or KU). You cannot modify a key that cannot be changed by requesting an RP function.

If the request is issued for a shared-file distributor server, the request is issued to the shared-file server that has the record, as identified by the PCB name.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | RP |
| Request DATA length | Length of record read |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Not used |
| 2 | 8 | PCB name |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | The record to be replaced in the shared file |

## Statistic request (SR function)

This function requests information about the functions sent by a workstation.

If the request is issued for a shared-file distributor server or a shared-file replicator server, the information is obtained from internal tables.

| CPRB Field | Content/Description |
|---|---|
| Function code | SR |
| Request DATA length | 0 or 2 |
| Request PARMLIST length | 26 |
| Reply DATA length | At least 24 |
| Reply PARMLIST length | 26 |
| Replied DATA length | At least 24 |
| Replied PARMLIST length | 26 |

| Request DATA Values when Request DATA length is 2 | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Code which identifies the workstation |

If the Request DATA length is zero, statistics for all workstations are returned in Reply DATA. In that case, the following information is preceded by the two-byte workstation

ID, and the entire structure is repeated for each workstation.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Number of erroneous BT function requests |
| 2 | 2 | Number of error-free BT function requests |
| 4 | 2 | Number of erroneous IS function requests |
| 6 | 2 | Number of error-free IS function requests |
| 8 | 2 | Number of erroneous DL function requests |
| 10 | 2 | Number of error-free DL function requests |
| 12 | 2 | Number of erroneous RP function requests |
| 14 | 2 | Number of error-free RP function requests |
| 16 | 2 | Number of erroneous RB function requests |
| 18 | 2 | Number of error-free RB function requests |
| 20 | 2 | Sum of erroneous FN, FU, FP, GN, GU, GP, HN, HU, HP, KN, KU, and KP function requests |
| 22 | 2 | Sum of error-free FN, FU, FP, GN, GU, GP, HN, HU, HP, KN, KU, and KP function requests |

**Notes:**

1. The integers are stored as binary numbers in the requester's native computer format.
2. Functions that return an AD, NS, TL, or NT error code do not update the statistical count.

## Test status (TS function)

This function tests whether the shared file is closed or open.

If the request is issued for a shared-file distributor server or a shared-file replicator server, the information is obtained from internal tables.

| CPRB Field | Content/Description |
|---|---|
| Function code | TS |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 to 4096 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 or 2 |
| Replied PARMLIST length | 26 |

The return code can be:

**X'00000000'** Shared file closed: there are no open transactions

**X'01004E43' (NC)** Shared file not closed: there may be open transactions

**X'01004F50' (OP)** Shared file not initialized: its initialization is deferred until the OPENLOG utility has been run. See the *LANDP Servers and System Management* book for more information. If the TS request is for a shared-file replicator server or a shared-file distributor server, the initialization process is started by this replicator or distributor server.

If you receive an NC return code, the Replied DATA includes a list of opened transactions in every workstation of the LANDP workgroup. The Replied DATA length must have a size of four bytes for each possible workstation.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Workstation id |
| 2 | 2 | Process id |
| The previous pattern is repeated for every open transaction until the list of open transactions is completed or the specified length is reached. | | |

## Erase shared-file data (ZD function)

This function is used to erase shared-file data, identified by a database description (DBD). You must issue an EX (request exclusive use) function request before requesting a ZD function. An automatic checkpoint is issued after this request.

If the request is issued for a shared-file distributor server, the request is issued to the shared-file server that has the record, as identified by the PCB name.

If the request is issued for a shared-file replicator server, the request is issued to all associated shared-file servers.

| CPRB Field | Content/Description |
|---|---|
| Function code | ZD |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

The PCB name in Request PARMLIST can be any PCB “belonging” to the relevant DBD.

**Note:** This function must be used with caution, because no rollback function is available.

## Examples of shared-file transactions

Following are example transaction sequences that you might find helpful when you design your program.

### Online transaction to search for information

```
[GF]                Grant
[OO]                Open on-line
GU, GN, or GP       Get record
...
[RF]                Revoke
[TS]                Test status
```

### Online transaction to update one shared file

```
[GF]                Grant
[OO]                Open online
GU, GN, or GP       Get record
...
BT                  Begin an updating transaction
HU, HN, or HP       Exclusive use of one record
RP                  Replace one record
...
IS                  Insert one record
...
CP                  Checkpoint
IS                  Insert one record
...
ET                  End of transaction
[RF]                Revoke
[TS]                Test status
```

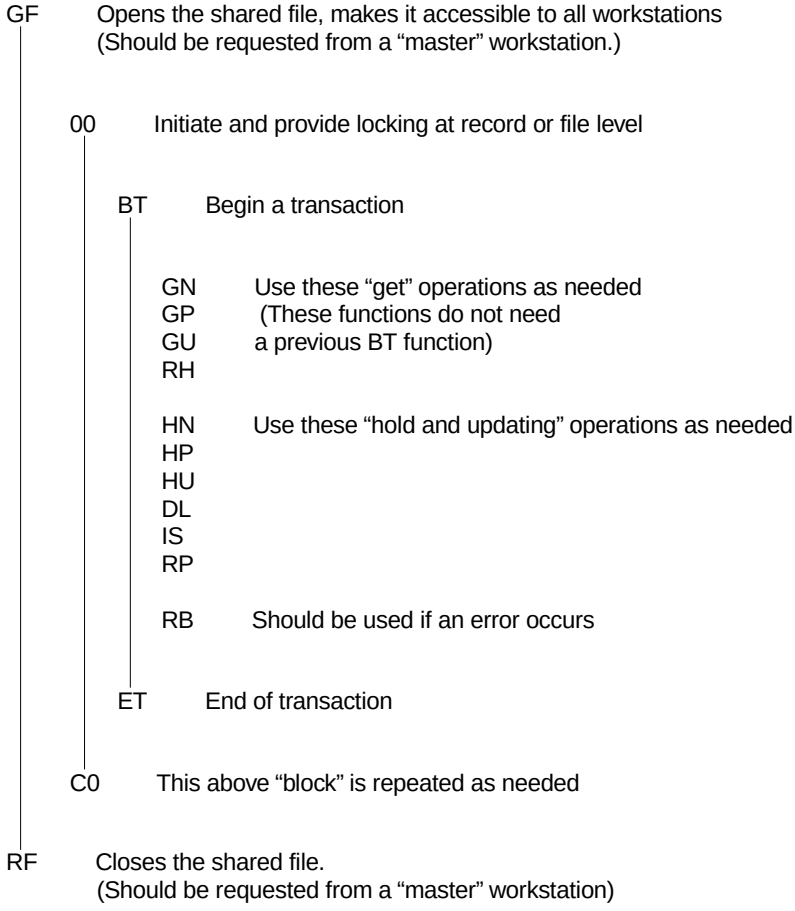
**Note:** Remember to use an ET or CB function before the end of your transactions. An RF function does not close the shared-file server until all the current transactions are closed (see explanations of RF function and “Shared-file server operating modes” on page 260).

### Offline transaction

```
[GF]                Grant
[OB]                Open batch
EX                  Exclusive use of one shared file
HU, HN, or HP       Hold record
RP                  Replace one record
...
IS                  Insert one record
...
```

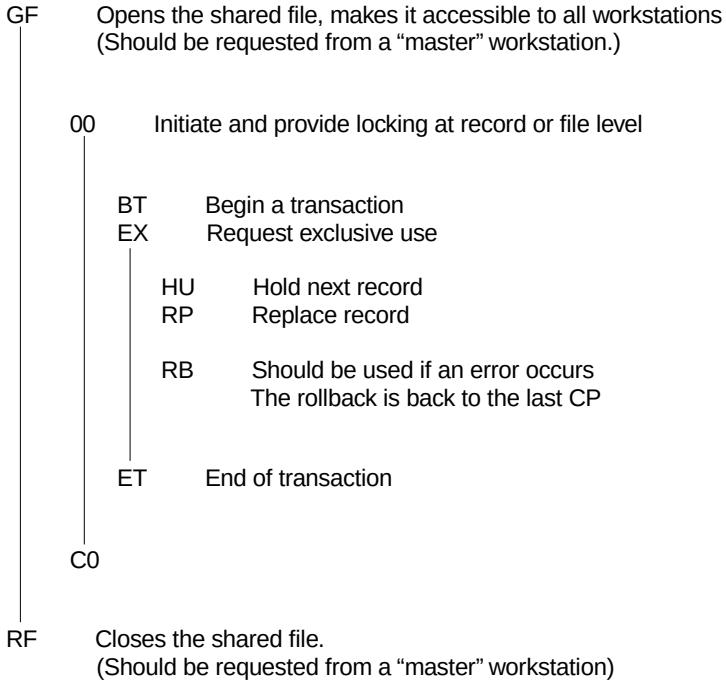
```
CP                  Checkpoint
IS                  Insert one record
...
CB                  Close batch
[RF]                Revoke
[TS]                Test status
```

## Typical use of the shared-file server in on-line mode

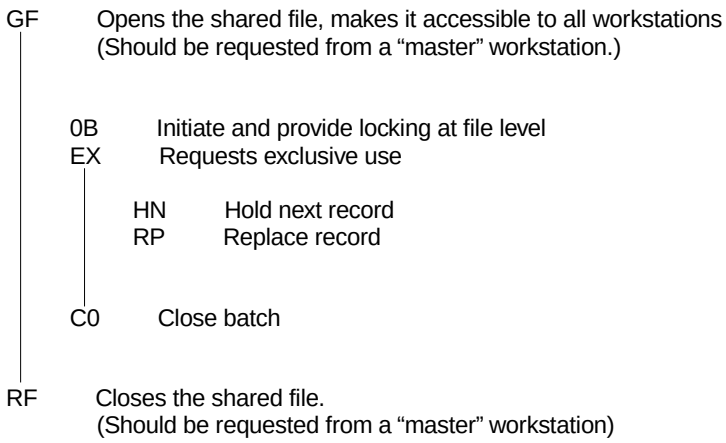


## Major changes at file level

One alternative is:



A second alternative is:



# Chapter 15. Query server

This chapter describes the query server and its available services for the computers in the LANDP workgroup.

In **LANDP for OS/2** the query server provides access to the IBM Database Server for OS/2 Warp, Version 4. It supports Distributed Relational Database Architecture (DRDA) connections to remote DB2 databases and private protocol connections to remote DB2/2 databases.

The query server provides database services, both to existing application programs using the shared-file server (with slight modifications) and to new application programs that extensively use structured query language (SQL).

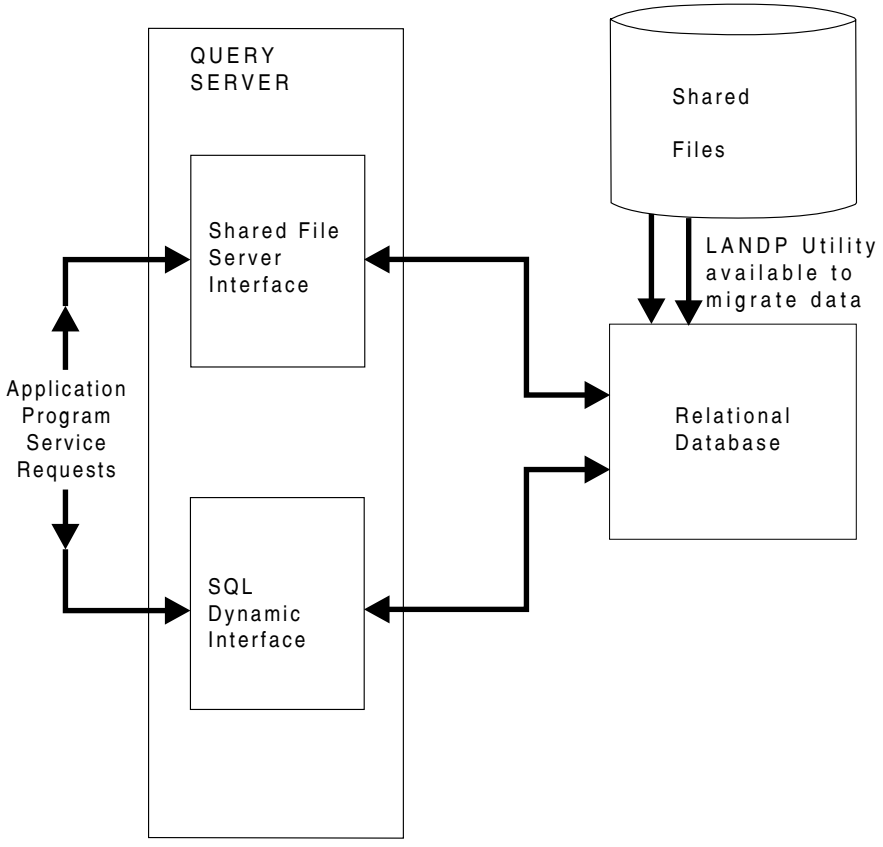



*Figure 4. Query Server Components and their Interaction with Applications and Databases*

© Copyright IBM Corp. 1992, 2002

The main objectives of this server are:
- Data integrity
- Shared multiple-station access
- Compatibility with the shared-file server
- Full SQL functions
- Common interface
- Application program portability

If you have existing applications that use the shared-file server and you want to use the query server, see the “Migrating from the shared-file server to the LANDP for DOS query server.” section in the “Clients and servers” chapter of the *LANDP Programming Guide*.

## Modes of operation

When a LANDP query server is installed and customized, it has three modes of operation:
- Not-logged-on mode
- Logged-in to query mode
- Logged-in to shared-file mode

### Not-logged-on mode

When the server is in its not-logged-on mode, some functions can be requested to enter and leave the logged-in modes and perform other duties like opening and closing additional sessions, see “Using the query server in not-logged-on mode” on page 312.

These functions can be requested without requiring you to work in any other mode and some of them can be requested from any mode. Two modes of logged-in operation are possible and four server functions are available to switch between them (OO, CO, OQ, and CQ) This is the initial mode of the server after successfully loading.

### Shared-file mode

In this mode, the server supports the shared-file function requests, allowing data accesses dealing with database descriptions (DBDs), program control blocks (PCBs), and transactional programming techniques. It transforms these requests and passes them to the database manager. The server then receives responses and maps the data and information to the normal fields of the server-requester programming interface.

This interface is almost compatible with the shared-file server functions. Most shared-file server functions are available. For more information refer to “Compatibility with the shared-file server” on page 329.

### Query mode

In this mode, the application program is able to send SQL statements through the SQ function and receive SQL responses through the LANDP network. Additional functions to fetch, insert, delete, and update a row are also available.

## SQL table structured files

There are several important characteristics of the SQL dynamic interface and the associated SQL table structured files:

- Applications running in a LANDP for DOS workstation, in the same LANDP workgroup, can also use the SQL dynamic interface. This also allows applications to be designed as portable applications.
- Utility programs are provided to convert files from shared-file format to SQL table structured files.
- The query manager of IBM DB/2 can be used to access data in LANDP for OS/2 files.
- Queries are defined as a part of the common API request. No preprocessing step is necessary.
- Data integrity is provided through transactional integrity and rollback functions. A logging process is added in the LANDP for OS/2 query server to ensure recovery from a backup copy of the files and associated log files.

  Forward recovery is also provided for this purpose in LANDP for OS/2.

  **Note:** LANDP for OS/2's logging process only records database activity that takes place through LANDP. To ensure that the log can be used for recovery, all database activity must be requested via LANDP.

The following example of a sequence of operations illustrates how the operation modes are entered:

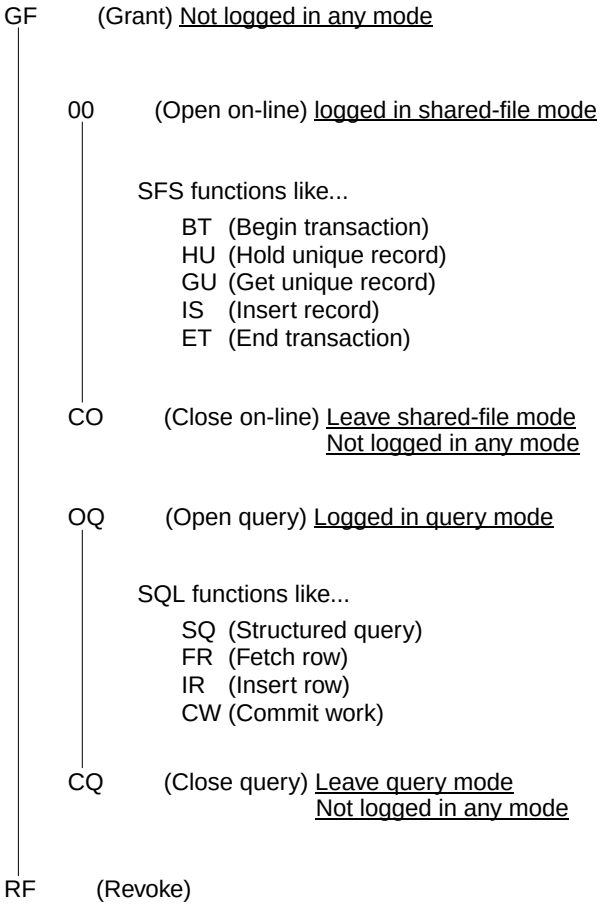



*Table 35. Function Codes used in the Query Server*

| Function code | Description | Env. | Page |
|---|---|---|---|
| **Functions available in any mode:** | | | |
| CS | Close session | -2-- | 312 |
| HL | Habilitate (enable) logging | -2-- | 313 |
| IL | Inhibit logging | -2-- | 314 |
| OS | Open session | -2-- | 315 |
| TS | Test status | -2-- | 316 |
| **Functions available in not-logged-on mode:** | | | |
| GF | Grant | -2-- | 312 |
| OO | Open on-line | -2-- | 314 |

*Table 35. Function Codes used in the Query Server (continued)*

| Function code | Description | Env. | Page |
|---|---|---|---|
| OQ | Open query mode | -2-- | 315 |
| RF | Revoke | -2-- | 316 |
| **Functions available in query mode:** | | | |
| CQ | Close query mode | -2-- | 319 |
| CW | Commit work | -2-- | 319 |
| DQ | Describe query | -2-- | 320 |
| DR | Delete row | -2-- | 321 |
| DT | Describe table | -2-- | 321 |
| EQ | End query | -2-- | 322 |
| FR | Fetch row | -2-- | 323 |
| IR | Insert row | -2-- | 323 |
| RR | Replace row | -2-- | 324 |
| RW | Rollback work | -2-- | 325 |
| SP | Set parameters | -2-- | 325 |
| SQ | Structured query | -2-- | 326 |
| **Functions available in shared-file mode** | | | |
| BT | Begin transaction | -2-- | 331 |
| CO | Close on-line | -2-- | 331 |
| CP | Checkpoint | -2-- | 332 |
| DD | Define DBD | -2-- | 332 |
| DI | Define index | -2-- | 333 |
| DL | Delete record | -2-- | 334 |
| DP | Define PCB | -2-- | 334 |
| ED | Erase DBD | -2-- | 335 |
| EI | Erase index | -2-- | 336 |
| EP | Erase PCB | -2-- | 336 |
| ET | End transaction | -2-- | 337 |
| EX | Request exclusive use | -2-- | 337 |
| GN | Get next record | -2-- | 338 |
| GP | Get previous record | -2-- | 339 |
| GU | Get unique record | -2-- | 340 |
| HN | Hold next record | -2-- | 341 |
| HP | Hold previous record | -2-- | 342 |
| HU | Hold unique record | -2-- | 342 |
| ID | Initialize DBD | -2-- | 343 |
| IP | Initialize pointers | -2-- | 344 |

*Table 35. Function Codes used in the Query Server (continued)*

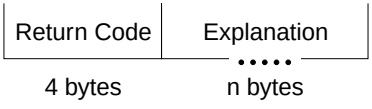
| Function code | Description | Env. | Page |
|---|---|---|---|
| IS | Insert record | -2-- | 344 |
| KN | Keep next record | -2-- | 345 |
| KP | Keep previous record | -2-- | 346 |
| KU | Keep unique record | -2-- | 347 |
| RB | Rollback | -2-- | 348 |
| RP | Replace record | -2-- | 348 |
| SD | Statistics on DBD | -2-- | 349 |
| SR | Statistic request | -2-- | 350 |
| ZD | Erase shared-file data | -2-- | 351 |

The first and second columns give the function code and the name of the function. The third column shows the operating environment of the function, as explained in "Operating environments" on page xxvi. "-2--" means that it is available from LANDP for OS/2 servers. The last column refers to the page where you can find the function described.

## Return code and explanation handling

If the query server reports a return code of EP, QE, or EE, then it provides the 4-byte field containing the error code from the OS/2 Database Manager, in Reply DATA, with the corresponding explanation concatenated to it. The length of Reply DATA determines whether the entire explanation can be provided, or only that part that fits into the remaining bytes.

For the return code EP, QE, or EE, Reply DATA is as follows:




If the length fields are left blank in the Reply DATA and PARMLIST areas, the length is the same as specified in the CPRB when the request is made.

*Log Access Failure:* If there is a log file access failure, the first four bytes are X'8FFFFFFF', followed by '\LOGPATH\EHCSQLOG.DAT'. It is recommended that you perform a ROLLBACK, since the function requested from the database manager has failed to be logged.

## I/O structure blocks

This section describes the I/O blocks passed to or returned from the server. The Column Name, Column Type, and Field Length fields are repeated for each DB2 column descriptor being transmitted.

## DBD definition

The first I/O block is the DBD definition.

| DBD Definition | | | | |
|---|---|---|---|---|
| **Field Name** | **Offset** | **Size** | **Type** | **Content** |
| DBD name | 0 | 20 | Char | Name of the DBD |
| Reserved | 20 | 2 | Integer | This must be set to 0 |
| Numoffields | 22 | 2 | Integer | Number of column descriptors (column name and type) that follow |
| | | | | |
| Column Name | 24xN | 20 | Char | Name of the column |
| Column Type | 20+24xN | 2 | Integer | Type of the table column that is described here |
| Field Length | 22+24xN | 2 | Integer | Length of the table column that is described (not including the space for null-indicators) |
| **Note:** This is used in request (DD). | | | | |

You must follow the ***Intel reversed format*** for the fields containing hexadecimal data. when putting data into the request data area of the CPRB for the DD function.

For example:

- Database name, YOUR_DATABASE.
- Two columns:

  **CUSTOMER_NAME** 30 bytes ASCII characters

  **CUSTOMER_NUM** 4 byte integer.
- Request Data Area values for DD function:

  **Offset 0 - 19** 'YOUR DATABASE '.

  **Offset 20 - 21** X'00' X'00'.

  **Offset 22 - 23** X'02' X'00', two column descriptors.

  **Offset 24 - 43** 'CUSTOMER_NAME'.

  **Offset 44 - 45** X'C4' X'01', column type of fixed length ASCII string.

  **Offset 46 - 47** X'1E' X'00', field length of 30 bytes.

  **Offset 48 - 67** 'CUSTOMER_NUM'.

  **Offset 68 - 69** X'F0' X'01', column type of 4 byte integer.

  **Offset 70 - 71** X'04' X'00', field length of 4 bytes.

## PCB definition

The next I/O block is used in defining the PCBs.

| PCB Definition | | | | |
|---|---|---|---|---|
| **Field Name** | **Offset** | **Size** | **Type** | **Content** |
| DBD name | 0 | 20 | Char | Name of the DBD |
| Column name | 20 | 20 | Char | Name of the column to be used for data access |
| **Note:** This is used in request (DP). | | | | |

## Short descriptor

The next I/O block defines the Short Descriptor. The Field Type and Field Length fields are repeated for each field descriptor being transmitted.

| Short Descriptor Block | | | | |
|---|---|---|---|---|
| **Field Name** | **Offset** | **Size** | **Type** | **Content** |
| Queryhandle | 0 | 2 | Integer | Query handle under which current function has been performed |
| Numoffields | 2 | 2 | Integer | Number of field descriptors (field type and length) that follow |
| | | | | |
| Field Type | 4xN | 2 | Integer | Type of the table field that is described here |
| Field Length | 2+(4xN) | 2 | Integer | Length of the table field that is described (not including the space for null-indicators) |
| | | | | |
| **Note:** This is used in reply (SP, EQ, FR, DT, IR, DR, RR). | | | | |

**Note:** The field length does not include the two additional bytes needed for the indicator variable if null values are allowed. It also does not include the half bytes needed for the used length in the 'Varchar' field types (448, 449, 456, 457, 476, 477).

## Long descriptor

The next block defines the Long Descriptor. The Name Length and Field Name fields are repeated for each field descriptor being transmitted.

| Long Descriptor Block | | | | |
|---|---|---|---|---|
| **Field Name** | **Offset** | **Size** | **Type** | **Content** |
| Name Length | 32xN | 2 | Integer | Number of characters that are valid in the following column name |
| Field Name | 2+(32N) | 30 | Char | Name of the described column |
| | | | | |
| **Note:** This is used in reply (DT, DQ). | | | | |

## Pre-Fetch block

This block describes the mechanism by which application programs can pre-fetch parameters to a SELECT statement before it is sent. The double lines in the table show that several parameters can be issued in the same call. The Parameter Value field is repeated for each field descriptor being transmitted.

| Pre-fetch Parameter Block | | | |
|---|---|---|---|
| **Field Name** | **Size** | **Type** | **Content** |
| | | | |
| Fieldtype (1) | 2 | Integer | Type of the parameter marker ('?') described here |
| Fieldlength (2) | 2 | Integer | Length of the parameter marker described (not including the space for null-indicators) |
| Parameter value | (2) | (1) | Parameter Value |
| | | | |
| **Note:** This is used in request (SP). | | | |

## Using the query server

This section provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields. If you need more information about the CPRB fields, see Appendix A, “Connectivity programming request block” on page 669.

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | 26 | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | | Request DATA length |
| 22 | 4 | Address | Request DATA address |

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 26 | 2 | 26 | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | EHCSQL## | Server name |
| **Note:** The value of ## is defined during customization and relates to the particular query server with which the application program communicates. | | | |

The following fields are variable and are discussed in each function request description:
- Function code
- Request DATA length
- Reply DATA length

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 44 | 2 | | Replied PARMLIST length |
| 46 | 2 | | Replied DATA length |

If the request was successful, the *router return code* and the *server return code* are both X'00000000'. In all other cases, see the appropriate section in the *LANDP Problem Determination* book, to see if you should take any action. The return values in Reply PARMLIST and Reply DATA should be ignored if there is an error.

The following fields are variable and are discussed in each function request description:
- Replied PARMLIST length
- Replied DATA length

**PARMLIST:** Request PARMLIST and Reply PARMLIST fields are described for each function. Many functions share a common format for the Request PARMLIST area, which is given here for convenience. Not all fields are required for all functions, and some variants exist.

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Comparison Operator (used in shared-file mode)<br><br>Specify one of the following:<br><br>**'EQ'** (Equal to) Search for the key or record number specified<br><br>**'GT'** (Greater than) Search for the first key or record number that is greater than the one specified<br><br>**'GE'** (Greater than or equal to) Search for the first key or record number that is greater than or equal to the key specified<br><br>**'NE'** (Not equal) Search for the first key or record number that is not equal to the one specified<br><br>**'LT'** (Less than) Search for the first key or record number that is less than the one specified<br><br>**'LE'** (Less than or equal) Search for the first key or record number that is less than or equal to the one specified<br><br>**'PR'** Search for the record returned by the last Hold, Get or Keep request, with the same PCB. (See note.) (LANDP for OS/2 only) |
| 0 | 2 | Query handle (used in query mode) |
| 0 | 8 | User ID (used in not-logged-on mode) |
| 2 | 8 | PCB name (used in shared-file mode) |
| 2 | 1 | Connect option or hold parameter (used in query and shared-file modes) |
| 10 | 2 | Session identifier (used in all modes) |
| **Note:** The 'PR' operator retrieves the last record accessed by the application program. If a record is retrieved using the PCB without holding, then another process can modify or delete the same record. If this occurs and if the application program then requests a Get, Hold, or Keep function with 'PR' operator using the same PCB, a GE (record not found) code is returned. The record must then be re-read to determine if it has been changed. Also note that when an IP, ID, or ET function is requested, a GU, HU, or KU function with 'PR' operator also results in a GE return code. | | |

**DATA:** Request DATA and Reply DATA contain data to be sent to or received from the server. The use of these areas is explained in the description of each function request.

## Using the query server in not-logged-on mode

Besides the functions that can be requested when in query mode and shared-file mode, other functions are supplied to enter and leave the query and shared-file modes, and to perform the following additional tasks:

- To start and end the server services (GF, RF), to enter and leave shared-file mode (OO, CO), and to enter and leave query mode (OQ, CQ).
- To request and close an additional working session (OS, CS). The query server provides a multiple session capability which allows the application program to have more than one session open at a time. Each session has special attributes and a separate commit unit. Thus, an application program can work simultaneously in shared-file and query modes (one mode per session).
- To start and stop logging for forward recovery purposes (HL, IL).
- To provide workstation-based statistics (SR) and server status feedback (TS).

## Request reference in not-logged-on mode or in any mode

These are the functions available in not-logged-on mode. This section also includes functions that can be issued in any mode. (See Table 35 on page 304 to differentiate between the two types.) They are listed in alphabetical order of function code.

### Close session (CS function)

This function closes a session previously opened with an OS function request. It can be used at any time with an opened session identifier. If the requesting session has an uncommitted opened transaction, a ROLLBACK is forced. All changes made to the database since the last COMMIT operation are lost. This function closes all open database files and releases any DBDs that are used exclusively.

| CPRB Field | Content/Description |
|---|---|
| Function code | CQ |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier to be closed |

### Grant (GF function)

This function starts the server operation. The server rejects any request (except the session managing requests OS, CS, SR, and TS) until this function is requested. This

gives the database administrator (DBA) some control over the server. Optionally, for compatibility reasons, this function accepts one parameter byte in Request DATA with value '0'. It must be requested only once by any of the served workstations.

| CPRB Field | Content/Description |
|---|---|
| Function code | GF |
| Request DATA length | 0 or 1 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | '0' |

## Habilitate (enable) logging (HL function)

This function is used to resume server logging for forward recovery.

Sometimes, it may be desirable to switch off logging to improve performance by requesting an inhibit log (IL) function. Then an HL function request resumes transaction logging. No transactions must be open in this session of the workstation that requests an HL function. This function is accepted but ignored, if logging is performed using the OS/2 ES facility, instead of the one provided with LANDP for OS/2.

| CPRB Field | Content/Description |
|---|---|
| Function code | HL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

## Inhibit logging (IL function)

This function is used to switch off transaction logging for forward recovery. It operates in the **LANDP for OS/2** environment.

When the command is requested, the logging of transactions is switched off for the calling session. *This function can prevent the correct working of forward recovery*. Generally, it is not advisable to switch off logging for workstations that use tables that other workstations are using at the same time. No transactions must be open in this session of the workstation that requests an IL function. This function is accepted but ignored, if logging is performed using the OS/2 ES facility, instead of the one provided with LANDP for OS/2.

| CPRB Field | Content/Description |
|---|---|
| Function code | IL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

## Open on-line (OO function)

This function logs in an application program to the LANDP query server in shared-file mode. By requesting this function, the requesting application program is allowed to perform any of the shared-file mode specific function requests, as explained on page “Using the query server in shared-file mode” on page 328.

| CPRB Field | Content/Description |
|---|---|
| Function code | OO |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |

| CPRB Field | Content/Description |
|---|---|
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

## Open query mode (OQ function)

This function logs in an application program to the query server in query mode. The requesting application program is allowed to perform any of this mode's specific function requests.

| CPRB Field | Content/Description |
|---|---|
| Function code | OQ |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

## Open session (OS function)

This function opens an additional working session for an application program. By requesting this function, the application program requests a session identifier that is used in following operations requested for the new session. Any function can be requested from the additional session in the same way as in the primary session. Request the desired function with the session identifier in Request PARMLIST offset X'000A'. The session identifier must always be specified when you close the session with a CS function request.

An additional session is like starting another application program from the same workstation. It has separate attributes (including working mode) and a separate commit unit (transaction integrity). By using this facility, an application program can work simultaneously in shared-file mode and in query mode (one mode per session).

| CPRB Field | Content/Description |
|---|---|
| Function code | OS |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | New session identifier |

## Revoke (RF function)

When this function is requested, the server waits until all the opened transactions are closed and it prevents new transactions from starting. It is intended for the DBA program to control server operations and must be performed only once by any of the served workstations.

| CPRB Field | Content/Description |
|---|---|
| Function code | RF |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

## Test status (TS function)

This function determines the server status. The status can be open and servicing, or closed. The return code for open is X'0100 xxxx' and for close is X'0000 0000'. The function can be used to check the status for the database management system (DBMS) where servicing is not allowed (for example, backup, and restore, and so on).

In LANDP for OS/2 the return code X'0100 xxxx' can be returned if the DBMS open has been deferred to a later time and the OPENDB2 utility has not been run.

| CPRB Field | Content/Description |
|---|---|
| Function code | TS |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |

| CPRB Field | Content/Description |
|---|---|
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Server return code | 256 if open, 0 if closed |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

## Using the query server in query mode

You enter this mode with an open query (OQ) function request and end it with a close query (CQ) function request. The main function in this mode is the structured query (SQ) see “Structured query (SQ function)” on page 326, which allows the application programmer to perform any SQL statement. If the performed statement is a SELECT statement, the application program obtains a query handle together with the first row of the result table that allows the application program to fetch the rest of the data using the fetch row (FR) function request.

When all the rows have been fetched (read), the query handle is closed without application program intervention and the same query handle is used for following SELECT statements. This introduces the concept of an “Active Query”, which is a SELECT statement that has been sent to the database before all the resultant rows have been fetched. You can close the handle (closing the Active Query) by reading the last row of the result table or by requesting the EQ function. You can also use the close query function (CQ) that leaves query mode and closes the opened query. A COMMIT function (CW) and a ROLLBACK function (RW) also close an open query handle.

This mode also offers a set of functions that allow the updating (RR), deleting (DR), and inserting (IR) of rows with respect to the current row of a given handle. These functions are especially useful because they help the application programmer to avoid building up complicated insert, update, or delete SQL statements.

Other functions supplied ensure general data-integrity handling for committing or rolling back the transaction (CW, RW). You can also obtain the complete description of the fields in a specified table (DT) or SELECT statement (DQ).

There is an additional function, (SP), that allows application programs to “pre-fetch” parameters to a SELECT statement before it is sent. The SELECT statement then can contain as many ’?’ parameter markers as parameters have been pre-fetched.

## Notes on the query server in query mode

- All fetch cursors are used without hold. That is, resources are returned to the system whenever a COMMIT point is reached.
- The maximum row length is 4096 bytes.
- The maximum supported statement length is 4000 bytes.
- The maximum supported number of columns per table is 128.

If you make transactions without using the server, it causes errors when you try forward recovery, because the transactions have not been logged in the forward recovery log file.

## Supported data types

The supported data types are:

| Data Types Allowed in Structured Query | | | | |
|---|---|---|---|---|
| *Field type (binary)* | *Field type (HEX)* | *Type* | *Field length* | *Support* |
| 384/385 | 0180/0181 | Date ASCII record | 10 | Fully supported |
| 388/389 | 0184/0185 | Time ASCII record | 8 | LANDP for OS/2 |
| 392/393 | 0188/0189 | Time stamp ASCII record | 26 (see notes 2 and 3) | Fully supported |
| 448/449 | 01C0/01C1 | Varying char string (2-byte length) ASCII record | n | LANDP for OS/2 |
| 452/453 | 01C4/01C5 | Fixed length string ASCII record | n | Fully supported |
| 460/461 | 01CC/01CD | Null ended varying | Length string | Supported from migration only |
| 464/465 | 01D0/01D1 | Varying-length graphic string (2-byte length) | 2*n | LANDP for OS/2 only |
| 468/469 | 01D4/01D5 | Fixed-length graphic string | 2*n | LANDP for OS/2 only |
| 476/477 | 01DC/01DD | Varying char string (1-byte length) | n | Supported from migration only |
| 480/481 | 01E0/01E1 | Float IEEE FP | LANDP for OS/2: 8 | Fully supported |
| 484/485 | 01E4/01E5 | Decimal | n (see note 1) Byte 1 = precision byte 2 = scale | Fully supported |
| 496/497 | 01F0/01F1 | 4-byte integer | 4 | Fully supported |
| 500/501 | 01F4/01F5 | 2 or 4-byte integer | 2 | Fully supported |

The first of the two field type values means that the column does not have a null indicator and does not allow null values. The second of the two field type values means that the column has a null indicator and does allow nulls. If the field type allows null values, an additional indicator variable of 2 bytes is added to each field as an indicator for null variables. However, the length of the indicator variable is not reflected in the length field. The field length value specifies the maximum length of the referred column (the space that must be reserved or allocated in the application program to hold the value).

## Request reference for query server in query mode

These are the functions available in query mode. They are listed in alphabetical order of function code.

### Close query mode (CQ function)

This function logs off an application program from the query server in query mode. After requesting this function, the requesting application program returns to the not-logged mode. If the requesting session has an uncommitted opened transaction, a COMMIT function is performed. Open query handles are closed.

| CPRB Field | Content/Description |
|---|---|
| Function code | CQ |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

### Commit work (CW function)

This function terminates a transaction and commits the database changes made during that transaction. It has the same effects as requesting SQ with the sentence 'COMMIT WORK' in Request DATA. The query handles are closed.

| CPRB Field | Content/Description |
|---|---|
| Function code | CW |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 or (to obtain error information) ≥ 4 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0<br>or<br>Length of error code + error message |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values for LANDP for OS/2 | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

## Describe query (DQ function)

This function obtains information about a given query. It is used to get feedback about the result columns of an SQL SELECT statement. The SQL statement is not performed. The descriptor is returned in the Reply PARMLIST area. A specific value, the null handle (X'FFFF'), is returned in the short descriptor block of the Reply PARMLIST. This can be used along with the IR function request to insert new rows into the table. The names of the query columns are returned in Reply DATA in the long-descriptor-block format.

| CPRB Field | Content/Description |
|---|---|
| Function code | DQ |
| Request DATA length | Length of the SELECT statement |
| Request PARMLIST length | 26 |
| Reply DATA length | ≥ 1 or (to obtain error information) ≥ 4 |
| Reply PARMLIST length | 26 |
| Replied DATA length | Long descriptor block length |
| Replied PARMLIST length | Length of descriptor |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | SELECT statement |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Short descriptor block |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Long descriptor block or error information |

## Delete row (DR function)

This function deletes the row at which the current query handle points.

| CPRB Field | Content/Description |
|---|---|
| Function code | DR |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 or (to obtain error information) ≥ 4 |
| Reply PARMLIST length | 26 |
| Replied PARMLIST length | Length of descriptor |
| Replied DATA length | 0 |
| Replied PARMLIST length | Length of descriptor |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Query handle returned by the SQ function |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Short descriptor block |

## Describe table (DT function)

This function obtains information about a table. Send the table name listed in Request DATA. The same information is returned as in SQ and FR function requests plus the names of the respective fields. These field names are in Reply DATA in the long-descriptor-block format. A specific value, the null handle (X'FFFF'), is returned in the short descriptor block of the Reply PARMLIST area. This can be used along with the IR function to insert new rows into the table.

| CPRB Field | Content/Description |
|---|---|
| Function code | DT |
| Request DATA length | Length of table name |
| Request PARMLIST length | 26 |
| Reply DATA length | ≥ 1 or (to obtain error information) ≥ 4 |
| Reply PARMLIST length | 26 |
| Replied DATA length | Long descriptor block length |
| Replied PARMLIST length | Length of descriptor |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Table name |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Short descriptor block |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Long descriptor block or error information |

## End query (EQ function)

Using this function is one way to close an opened query handle without reading the remaining rows. If you request an FR function and the return code is ED (end of data), the query handle is also closed. The CQ, CW, and RW function requests also close all the query handles.

| CPRB Field | Content/Description |
|---|---|
| Function code | EQ |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 or (to obtain error information) ≥ 4 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0<br>or<br>Length of error code + error message |
| Replied PARMLIST length | Length of descriptor |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Query handle returned by SQ |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Short descriptor block |

## Fetch row (FR function)

This function obtains the next row of a query sent with the SQ (Structured Query) function. If the return code is ED, the query handle will be closed.

| CPRB Field | Content/Description |
|---|---|
| Function code | FR |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | ≥ 1 or (to obtain error information) ≥ 4 |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of data fetched |
| Replied PARMLIST length | Length of descriptor |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Query handle returned by SQ |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Short descriptor block |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Data fetched or error information |

## Insert row (IR function)

This function inserts a new row into the table implicitly specified by the query handle. The short descriptor is also returned in Reply PARMLIST.

| CPRB Field | Content/Description |
|---|---|
| Function code | IR |
| Request DATA length | Length of row |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 or (to obtain error information) ≥ 4 |

| CPRB Field | Content/Description |
|---|---|
| Reply PARMLIST length | 26 |
| Replied DATA length | 0<br>or<br>Length of error code + error message |
| Replied PARMLIST length | Length of descriptor |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Query handle returned by SQ or null handle returned from DT or DQ |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Row to insert |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Short descriptor block |

## Replace row (RR function)

This function replaces the row to which the current query handle points, with the row supplied in Request DATA.

The query handle identifying the table to be updated must have been opened with the FOR UPDATE OF clause.

| CPRB Field | Content/Description |
|---|---|
| Function code | RR |
| Request DATA length | Length of row |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 or (to obtain error information) ≥ 4 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 or<br>Length of error code + error message |
| Replied PARMLIST length | Length of descriptor |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Query handle returned by SQ |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | New row to replace old row |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Short descriptor block |

## Rollback work (RW function)

This function is used to roll back work. It terminates a transaction and cancels the database changes that were made during that transaction. It has the same effect as requesting SQ with the sentence 'ROLLBACK WORK' in Request DATA. The query handles are closed.

| CPRB Field | Content/Description |
|---|---|
| Function code | RW |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 or (to obtain error information) ≥ 4 |
| Replied DATA length | 0 or<br>Length of error code + error message |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values for LANDP for OS/2 | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

## Set parameters (SP function)

This function sets the parameters to be used with the next SQ function. Parameters are passed in the Request DATA in the format of *pre-fetch parameters*, described on page"Pre-Fetch block" on page 309.

You can send one or more parameters within the same SP function request. Following SP requests stack the parameters in a parameter area. The parameters are used in the same order as they are sent to the server. They are deleted when an SQ or a CQ function is requested.

| CPRB Field | Content/Description |
|---|---|
| Function code | SP |
| Request DATA length | Length of query parameters to set |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Pre-fetch parameters block |

## Structured query (SQ function)

This function sends and executes the SQL sentence found in Request DATA. If the sentence is performed successfully, the return code is zero.

All SQL statements requested through the SQ function can only contain valid language elements for the specific SQL.

If the sentence is a SELECT statement, the function returns the first row of the selected query result. The data returned in Reply PARMLIST is a descriptor of the row thus obtained. If the result of the query is null, (no rows), the return code is ED. The format of the descriptor is a short descriptor block. The first two bytes of the short descriptor block represent the query handle, a number that identifies this SELECT statement as an “opened query”. Following rows can be obtained by requesting the FR function using the query handle returned. The query handle must be used for retrieving and updating the rest of the data with the function requests FR, RR, and IR.

If you are going to use the RR function request at the returned identifier, the query handle identifying the table to be updated must have been opened with the FOR UPDATE OF clause for each column to be updated.

If the field type allows null values, an additional indicator variable of two bytes is added before each field as an indicator for null values. If this indicator contains the value -1, the field contains a null value. If this indicator contains the value 0, the field contains a value. This length, however, is not reflected in the length field.

The field length value specifies the maximum length of the referenced column. This is the space that must be reserved or allocated in the application program to hold the value.

If the sentence is not a SELECT statement, no data or parameters are returned. You can issue ANY standard SQL statement, even ROLLBACK or COMMIT. Both of these, however, are detected by the server and it requests either a RW or CW function. If the sentence results in an error, the first four bytes of Reply DATA contain the SQL return code. If enough space is defined in Reply DATA length, the SQL error message is appended after the SQL return code.

The only exception is if there is a log file access failure, see page 306.

| CPRB Field | Content/Description |
|---|---|
| Function code | SQ |
| Request DATA length | Length of query |
| Request PARMLIST length | 26 |
| Reply DATA length | ≥ 1 or (to obtain error information) ≥ 4 |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of data fetched |
| Replied PARMLIST length | Length of descriptor |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Query handle |
| 10 | 2 | Session identifier (X'0000' is the main session) |
| **Note:** The query handle is a number which is used by the query server to identify the query. | | |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Query statement |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Short descriptor block |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Data fetched or error information |

## Using the query server in shared-file mode

The shared-file mode of the query server allows a subset of the shared-file functions of LANDP to access data dealing with DBDs and PCBs and transactional programming techniques. The server internally transforms these requests and passes them to the database system, receiving the response and mapping data and information to CPRB fields. The shared-file mode is entered with the open on-line OO function request and exited with the close on-line CO function request. The shared-file mode allows the application program to use the same application programmer interface that is supported with the shared-file server of LANDP, with some extensions and some restrictions.

## Concepts

In LANDP, each shared-file server can contain several shared files. The characteristics of the individual shared files are defined at customization in a table called the shared-file *database description* (DBD). A DBD is a data file with records of the same length and contains:

- Record length
- File name
- Number of keys
- Key characteristics

A *key* is a part of a DBD record used to access data and it has an associated index file for fast data access.

A *program control block* (PCB) is an “access method” that relates a DBD with a key in a unique way and contains a pointer to the data file. This means that it is possible to search for a data record using a PCB, and from this starting point, to continue searching for another record, using the key of the PCB.

Each DBD, PCB, or key has an associated name.

## Shared-File mode in LANDP for OS/2

The table structure in shared-file emulation mode is as simple as possible. A catalog table called EHCSQLTB is maintained by the server. It contains the name of all the PCBs defined, and also the name of the DBDs and related fields for the PCBs. DBD names are, in fact, table names. Field names are table column names. When the server is loaded, or if an OO function is requested and the database was previously changed, this table is read into main memory, checking the existence of the related tables and fields. A description of the table fields is also stored in memory for performance reasons. This table can be seen as an “extension” to the system catalogs table. This way, it is possible for an application program to define a PCB by accessing this catalog. The description of this table is as follows:

| **EHCSQLTB for LANDP for OS/2** | | |
|---|---|---|
| **Field name** | **Field type** | **Content** |
| PCBNAME | Char (8) not null | PCB name |
| TABLENAME | Char (20) not null | Table name (DBD name) |
| FIELDNAME | Char (20) not null | Field name of this PCB |

These tables contain the names of all PCBs defined, and also their DBDs and field names. The *tablename* field specifies a table that must exist in the corresponding SQL database.

The tables created through shared-file DDL functions (DD) and those generated by the migration utility programs, have an additional column, EHCSQLCL. The server adds and maintains the fields in this column, which it uses for sequential or direct access. You will see the field when you retrieve data from one of these tables using query mode. You must specify EHCSQLCL as the column field-name for any PCB intended for sequential access. This EHCSQLCL field is transparent to the user working in shared-file mode, but it must be maintained when working in query mode.

Using the SQL server, the concepts as described in “Concepts” on page 328 are migrated to the SQL language. That is, a DBD name is a table name. A key is a column name, and the associated index file is now an index created with the SQL statement “CREATE INDEX” over the specified field. A PCB can be thought of as an “opened cursor” to access a table.

## Data definition and data manipulation

Functions are provided to allow data manipulation and also data definition. The data manipulation functions, which the query server supports are:

- Get and hold functions (GU, HU, KU, GN, HN, KN, GP, HP, KP)
- Insert, replace or delete updating functions (IS, RP, DL, ZD)
- The flow and integrity control functions (BT, ET, CP, ID, IP, EX, RB)

The data definition functions, which the query server supports are:

- Define or erase a DBD (DD, ED)
- Define or erase a new index (DI, EI)
- Define or erase a PCB (DP, EP)

## Compatibility with the shared-file server

The query server, in shared-file mode, is generally compatible with the shared file server. However some differences exist.

### Notes for shared-file mode of LANDP for OS/2 query server

Although the servers have been designed to be highly compatible there may be some functional differences between the query server and the shared-file server. The main differences are:

- Application programs may need to provide some support for new return codes. These are programming errors that only need to be displayed.
- The “sequential access” method and the record number assignment do not work in the same way, and the record numbers that belong to erased records may no longer be reused as they may be in the shared-file server.
- The meaning of the error codes for the EP return code has been changed. It now reflects SQL errors instead of primary errors. The only exception is if there is a log file access failure. For further information, see “Return code and explanation

handling” on page 306. The values returned in Reply DATA are now:

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 4 | SQL error code (long integer) |
| 4 | | SQL error message (ASCII string) |

- The index ordering sequence depends on the declared data type, as specified when requesting the DD (define DBD) function. Whether indexes can accept duplicates is now specified in the DI (define index) function.
- The response times vary from those obtained with the shared-file server.
- Some function processing has been added to this mode, including all the DDL functions. If you code these functions or operators in your program, you will not be able to migrate your application program to the DOS environment without some redesign.
- If you access tables created with the function DD using the QUERY mode, you find a column EHCSQLCL added to the data tables. You must take care with this column, because you can destroy the data scheme by altering the “system” table, EHCSQLTB.

## Performance considerations under LANDP for OS/2

The performance behavior of the server is different from the old shared-file server, especially if you specified “fast cursor” when you loaded the program parameters. Here, requesting a get function (GU, HU, KU, GN, HN, KN, GP, HP, KP), may take several minutes because SQL DBMS may decide to reorder the result table in memory rather than accessing the index. Following retrievals of the resulting records are, however, much faster. If you don’t want this happening, do not load the server using this option. The index path data search is forced, simulating the shared-file response time behavior. The second issue where response times are different, is in table locking or record locking. With the query server the RL return code is never received sooner than the timeout value number of seconds, because the timeout has to expire before this return code is issued.

Tuning of the timeout value is very important. The correct value should always be smaller than the customized time to search for deadlocks in the chosen database. If the timeout is higher than this, DBMS may decide to roll back a workstation. This could cause a compatibility issue for application programs that have no supporting code to cover the situation because they were from an earlier version. Also, the timeout value should be big enough to allow normal SQL operation. Otherwise DBMS may be interrupted even if no locking conflicts are being issued. The default value is 15 seconds. It is recommended to increase this “time for search deadlocks” of the DBMS for the database which is to be accessed for the query server up to the maximum to allow the server control deadlocks without DBMS interferences.

For performance tuning of the server, see the *LANDP Servers and System Management* book.

Fields that contain null values cannot be associated with PCBs when working in shared-file mode. If you define null values for fields that are associated with PCBs, *the results will be unpredictable*.

## Request reference in shared-file mode

These are the functions provided by the query server when it is operating in shared-file mode. They are listed in alphabetical order of function code.

### Begin transaction (BT function)

The BT function begins a new transaction. A transaction consists of several related changes of the database that appear as a single change to the user. This function can only be requested if no transaction has been started in the same working session of the same LANDP application. Beginning a transaction allows the update of data by using insert IS, replace RP, or delete record DL function requests. You can cancel a transaction with the rollback RB function anytime during the transaction, until you request either an end transaction ET or a checkpoint CP function. The ET function completes the execution of the transaction and confirms that the changes have been made. The CP function only confirms the changes. Close online CO or close session CS function requests also end the current opened transaction, whereby CO forces a COMMIT operation and CS forces a ROLLBACK operation.

| CPRB Field | Content/Description |
|---|---|
| Function code | BT |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

### Close on-line (CO function)

This function logs off an application program from the shared-file mode of the LANDP query server. After requesting CO, the application returns to the not-logged mode. If the requesting session has an uncommitted opened transaction, a COMMIT is performed. All open database files for this session are closed and any DBDs that are used exclusively are released.

| CPRB Field | Content/Description |
|---|---|
| Function code | CO |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

## Checkpoint (CP function)

This function validates all operations in the current transaction and begins a new "logical" transaction. The effect is the same as requesting an end-transaction ET function and a begin-transaction BT function in a single request. Record locks are freed and DBDs used exclusively are released.

| CPRB Field | Content/Description |
|---|---|
| Function code | CP |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

## Define DBD (DD function)

This function defines a new DBD by supplying the DBD definition. This is an alternative way of issuing a 'CREATE TABLE' statement directly to the DBMS.

A column is added among those specified in Request DATA. This column, EHCSQLCL, is transparent to the application programs, if the server is used in shared-file mode. It is used to store the record number of the inserted records.

| CPRB Field | Content/Description |
|---|---|
| Function code | DD |
| Request DATA length | Length of the DBD definition block |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | DBD definition block |

For details of the DBD definition block, see “DBD definition” on page 307.

## Define index (DI function)

This function defines a new index to access data through a specified PCB. It is an alternative way of issuing a ’CREATE INDEX’ statement directly to the DBMS. A character, passed in Request DATA, specifies if the data can be duplicated (with repeated keys) or is unique (without repeated keys). Another character is passed to specify whether the index must be sorted in ascending or descending order. The DBD table is indexed through the related field name as specified in the define PCB (DP) function request. This function creates an index, which is sorted by the specified field and is available for accessing data. The index is used to determine the record to be obtained using a subsequent GN, GP, HN, HP, KN, or KP function.

The name of this index is composed of the DBD name (up to 8 characters) plus the field name (up to 8 characters) and the character ’A’ for ascending or ’D’ for descending.

| CPRB Field | Content/Description |
|---|---|
| Function code | DI |
| Request DATA length | 2 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |

| CPRB Field | Content/Description |
|---|---|
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 8 | PCB name for which to create a new index |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | **'U' or 'u'** Unique<br>**'D' or 'd'** Duplicated |
| 1 | 1 | **'+'** Ascending<br>**'–'** Descending |

## Delete record (DL function)

This function erases the record pointed to by a given PCB. This function request requires an open transaction and a record must have been obtained through HU, KU, HN, KN, HP, or KP.

| CPRB Field | Content/Description |
|---|---|
| Function code | DL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 8 | PCB name from which to delete record |
| 10 | 2 | Session identifier (X'0000' is the main session) |

## Define PCB (DP function)

This function defines a new PCB by supplying the PCB name, the associated DBD and the key field name. The referred DBD with the key field must already exist in the database. Otherwise an error is returned. An EX function should be previously requested against the related DBD. For a detailed description of the PCB definition

block, see “PCB definition” on page 308.

| CPRB Field | Content/Description |
|---|---|
| Function code | DP |
| Request DATA length | Size of the PCB definition block |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 8 | PCB name to define |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | ≥40 | PCB definition block |

## Erase DBD (ED function)

This function erases a DBD definition. It is an alternative way of issuing ’DROP TABLE’ directly to the DBMS. All related PCBs must be deleted before requesting an ED.

| CPRB Field | Content/Description |
|---|---|
| Function code | ED |
| Request DATA length | 20 (length of the database name to be erased) |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | DBD name to erase. Length must be greater than or equal to that of the DBD name |

## Erase index (EI function)

This function erases an index that was created with the DI function request. Request DATA must specify the same attributes that were used to define the index. An EX function request is previously required.

| CPRB Field | Content/Description |
|---|---|
| Function code | EI |
| Request DATA length | 2 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 8 | Name of PCB with the index that is to be erased |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | **'U' or 'u'** Unique<br>**'D' or 'd'** Duplicated |
| 1 | 1 | **'+'** Ascending<br>**'–'** Descending |

## Erase PCB (EP function)

This function erases a PCB definition. Pass the name of the PCB, which you want to erase, in Request PARMLIST. Lock the table with a request exclusive use EX function, then request an EP function.

Deleting the PCB is not effective until an end transaction ET or checkpoint CP is requested. Before one of these functions has been requested, the PCB definition may be restored by the rollback RB function.

| CPRB Field | Content/Description |
|---|---|
| Function code | EP |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 8 | PCB name to erase |
| 10 | 2 | Session identifier (X'0000' is the main session) |

## End transaction (ET function)

This function ends the current transaction and commits the changes to the database on the disk. If the system hangs, is shut down, or a CS function is requested before the function completes its operation, the changes made to the database is rolled back. That is, the database appears as if the changes have never been made to the files. If a CO function is requested, a COMMIT operation is performed. This function frees locked records and releases DBDs used exclusively.

| CPRB Field | Content/Description |
|---|---|
| Function code | ET |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

## Request exclusive use (EX function)

This function requests the exclusive use of a PCB and its related DBD. It requests the exclusive use of that PCB and other PCBs that relate to the same DBD. You must request this function before you use the:

- Erase PCB definition (EP) function request

- Define PCB (DP) function request
- Erase index (EI) function request
- Erase shared-file data (ZD) function request

| CPRB Field | Content/Description |
|---|---|
| Function code | EX |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 8 | PCB name of the DBD for which you want exclusive use |
| 10 | 2 | Session identifier (X'0000' is the main session) |

## Get next record (GN function)

This function reads the next record into the Reply DATA area (without locking the record). It obtains the record located immediately after the previous record read for the same PCB, based on the defined index associated with the PCB (see the DI function). If this function is used immediately after the database server has been loaded or initialized, it reads the first record of the shared file. If there are no more records in the file following, the function returns a return code of GB. After getting the return code GB, the next GN function request reads the first record of the file. This function can be requested without a transaction being opened.

| CPRB Field | Content/Description |
|---|---|
| Function code | GN |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | ≥ Expected length of resulting record |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of resulting record |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 8 | PCB name for which to obtain record |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Record read |

## Get previous record (GP function)

This function obtains the record located immediately before the previous record without locking it, read for the same PCB based on the defined index associated with the PCB (see the DI function). If this function is used immediately after the database server has been loaded or initialized, it reads the last record of the shared file (that is, it wraps around). If GP or HP function request with the same PCB has read the first record in the file, the GP function request returns a return code of GB. After getting the return code GB, the next GP function request reads the last record of the file. This function can be requested without an open transaction.

| CPRB Field | Content/Description |
|---|---|
| Function code | GP |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | ≥ Expected length of resulting record |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of resulting record |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 8 | PCB name from which to obtain record |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Record read |

## Get unique record (GU function)

This function obtains a record using a given PCB without locking it. The record is obtained using the search criteria specified in the field “Comparison Operator” of Request PARMLIST. The data passed in Request PARMLIST must be of the same field type and field length as specified in the DBD definition, see DD and DP function requests. The function returns GE if no matching record is found. If a PCB name is specified that has been defined with the key field blank, a record number for direct access has to be specified in Request DATA.

This function can be requested even without an open transaction.

| CPRB Field | Content/Description |
|---|---|
| Function code | GU |
| Request DATA length | Key length to search |
| Request PARMLIST length | 26 |
| Reply DATA length | ≥ Expected length of resulting record |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of resulting record |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Comparison Operator (see page 311) |
| 2 | 8 | PCB name from which to obtain record |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Key value for which to search, or record number |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Record read |

## Hold next record (HN function)

This function obtains the next record into the Reply DATA area.

The next record is the record located immediately after the previous record read for the same PCB based on the defined index associated with the PCB (see the DI function). The function returns GB return code if there are no more records in the file following. After getting the return code GB, the next HN function request reads the first record of the file. If this function is used immediately after the database server has been loaded or initialized, it reads the first record of the file.

This function needs an open transaction.

| CPRB Field | Content/Description |
|---|---|
| Function code | HN |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | ≥ Expected length of resulting record |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of resulting record |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 8 | PCB name from which to obtain record |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Record read |

## Hold previous record (HP function)

This function obtains the record located immediately before the previous record, read for the same PCB based on the defined index associated with the PCB (see the DI function). If this function is used immediately after the database server has been loaded or initialized, it reads the last record of the shared file because it wraps around. The function returns the GB return code if no previous record is found. After getting the return code GB, another HP function request reads the last record of the file.

This function needs an open transaction.

| CPRB Field | Content/Description |
|---|---|
| Function code | HP |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | ≥ Expected length of resulting record |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of record read |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 8 | PCB name from which to obtain record (8 characters) |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Record read |

## Hold unique record (HU function)

This function obtains a record using a given PCB. The record is obtained using the search criteria specified in the field “Comparison Operator” of Request PARMLIST The data passed in Request PARMLIST must be of the same field type and field length as specified in the DBD definition, see the DD and DP function requests. The code GE is returned if the search criteria do not find a matching record. If a PCB name is specified that has been defined with the key field blank, a record number for direct access has to be specified in Request DATA.

This function needs an open transaction.

| CPRB Field | Content/Description |
|---|---|
| Function code | HU |
| Request DATA length | Key length to search |
| Request PARMLIST length | 26 |
| Reply DATA length | ≥ Expected length of resulting record |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of resulting record |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Comparison Operator (see page 311) |
| 2 | 8 | PCB name from which to obtain record |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Key value for which to search, or record number |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Record read |

## Initialize DBD (ID function)

This function initializes a DBD, resetting each PCB pointer for the DBD. It operates in the **LANDP for OS/2** environment.

The parameter used is PCB Name.

| CPRB Field | Content/Description |
|---|---|
| Function code | ID |
| Request DATA length | 0 |

| CPRB Field | Content/Description |
|---|---|
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Not used |
| 2 | 8 | PCB name of the DBD to be initialized |

## Initialize pointers (IP function)

This function sets all PCB record pointers to null, that is, to the initial state. If a GN or HN function is requested after this, the first record of the database file is obtained. If a GP or HP function is requested after IP, the last record of the database file is obtained.

| CPRB Field | Content/Description |
|---|---|
| Function code | IP |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

## Insert record (IS function)

This function inserts a record into the file specified by the PCB name. The inserted record must match in length and data type with the DBD to which the PCB is related.

This function needs an open transaction.

| CPRB Field | Content/Description |
|---|---|
| Function code | IS |
| Request DATA length | Length of record to be inserted |
| Request PARMLIST length | 26 |

| CPRB Field | Content/Description |
|---|---|
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 8 | PCB name through which to insert record |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Record to insert |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

## Keep next record (KN function)

This function obtains the next record into the Reply DATA area. It operates in the LANDP for OS/2 environment.

The next record is the record located immediately after the previous record read for the same PCB based on the defined index associated with the PCB (see the DI function). The function returns GB return code if there are no more records in the file following. After getting the return code GB, the next KN function request reads the first record of the file. If this function is used immediately after the database server has been loaded or initialized, it reads the first record of the file.

This function needs an open transaction.

| CPRB Field | Content/Description |
|---|---|
| Function code | KN |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | ≥ Expected length of resulting record |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of resulting record |

| CPRB Field | Content/Description |
|---|---|
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 8 | PCB name from which to obtain record |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Record read |

## Keep previous record (KP function)

This function obtains the record located immediately before the previous record, read for the same PCB based on the defined index associated with the PCB (see the DI function). It operates in the LANDP for OS/2 environment. If this function is used immediately after the database server has been loaded or initialized, it reads the last record of the shared file (that is, it wraps around). The function returns the GB return code if no previous record is found. After getting the return code GB, another KP function request reads the last record of the file.

This function needs an open transaction.

| CPRB Field | Content/Description |
|---|---|
| Function code | KP |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | ≥ Expected length of resulting record |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of record read |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 8 | PCB name from which to obtain record (8 characters) |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Record read |

## Keep unique record (KU function)

This function obtains a record using a given PCB. It operates in the LANDP for OS/2 environment. The record is obtained using the search criteria specified in the field “Comparison Operator” of Request PARMLIST. The data passed in Request PARMLIST must be of the same field type and field length as specified in the DBD definition, see the DD and DP function requests. The code GE is returned if the search criteria do not find a matching record. If a PCB name is specified that has been defined with the key field blank, a record number for direct access has to be specified in Request DATA.

The function needs an open transaction.

| CPRB Field | Content/Description |
|---|---|
| Function code | KU |
| Request DATA length | Key length to search |
| Request PARMLIST length | 26 |
| Reply DATA length | ≥ Expected length of resulting record |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of resulting record |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Comparison Operator (see page 311) |
| 2 | 8 | PCB name from which to obtain record |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Key value for which to search, or record number |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 4 | Record number in PC format (word-reversed and byte-reversed) |
| 22 | 4 | Record number, not reversed |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Record read |

## Rollback (RB function)

This function ends the current transaction by undoing all the changes to the database since the last ET or CP function request. The opened transaction appears to the database as if it had never been requested. This request is useful to maintain data integrity when a problem appears in the transaction processing. This function frees record locks and releases DBDs used exclusively.

This function needs an open transaction.

| CPRB Field | Content/Description |
|---|---|
| Function code | RB |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

## Replace record (RP function)

This function replaces the current record pointed to by a PCB with the one supplied in the Request DATA area.

This function needs an open transaction and a record must have been obtained through HU, KU, HN, KN, HP, or KP.

| CPRB Field | Content/Description |
|---|---|
| Function code | RP |
| Request DATA length | Length of record to be replaced |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 8 | PCB name from which to replace record |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | New record with which to replace an old one |

## Statistics on DBD (SD function)

This function is used to produce statistics on a given DBD. It provides information that can be used when performance improvement is required. If the DBD is in exclusive mode (an EX function was requested), the statistics are run in change mode. Otherwise they are run in reference mode, which allows other application programs to access the data and update the table. Request this function when the table definitions have changed or when the data in the table has been extensively changed.

| CPRB Field | Content/Description |
|---|---|
| Function code | SD |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 8 | PCB name of related DBD |
| 10 | 2 | Session identifier (X'0000' is the main session) |

## Statistic request (SR function)

This function requests statistics information for a given workstation. A workstation identifier (only two characters are significant) is required in Request DATA.

| CPRB Field | Content/Description |
|---|---|
| Function code | SR |
| Request DATA length | 2 |
| Request PARMLIST length | 26 |
| Reply DATA length | 24 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 24 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Workstation identifier |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Number of unsuccessful BT function requests |
| 2 | 2 | Number of successful BT function requests |
| 4 | 2 | Number of unsuccessful IS function requests |
| 6 | 2 | Number of successful IS function requests |
| 8 | 2 | Number of unsuccessful DL function requests |
| 10 | 2 | Number of successful DL function requests |
| 12 | 2 | Number of unsuccessful RP function requests |
| 14 | 2 | Number of successful RP function requests |
| 16 | 2 | Number of unsuccessful RB function requests |
| 18 | 2 | Number of successful RB function requests |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 20 | 2 | Number of unsuccessful Get, Hold, and Keep function requests |
| 22 | 2 | Number of unsuccessful Get, Hold, and Keep function requests |

LANDP for OS/2: Functions that return an AD, ND, NT, or TL error code do not update the statistical counters.

## Erase shared-file data (ZD function)

This function erases the contents of a given DBD identified by a PCB name. An EX function request must have been requested before this function request can be requested. The PCB and DBD definitions remain defined.

You can perform a ROLLBACK with this function request.

| CPRB Field | Content/Description |
|---|---|
| Function code | ZD |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 8 | PCB name of the DBD from which to erase contents |
| 10 | 2 | Session identifier (X'0000' is the main session) |

## Migrating data

Utility programs are supplied that convert SBCS data and data definitions written for *shared-file databases* from LANDP for DOS or LANDP for Windows into a form suitable for LANDP for OS/2 databases. See the *LANDP Servers and System Management* book for more about migrating data.

# Chapter 16. ODBC query server

This chapter describes the ODBC query server and its available services for the workstations in the LANDP workgroup. The ODBC query server:

- Is currently supported only on the LANDP for Windows platform.
- Provides access to a number of RDBMs (Relational Database Management Systems).
- Is functionally equivalent to the SQL mode of the LANDP for OS/2 query server.
- Provides database services to application programs that use SQL.
- Does not currently support the shared-file interface.

The ODBC query server provides database services to application programs that use structured query language (SQL). The ODBC query server does not support application programs that use the shared-file mode of operation.

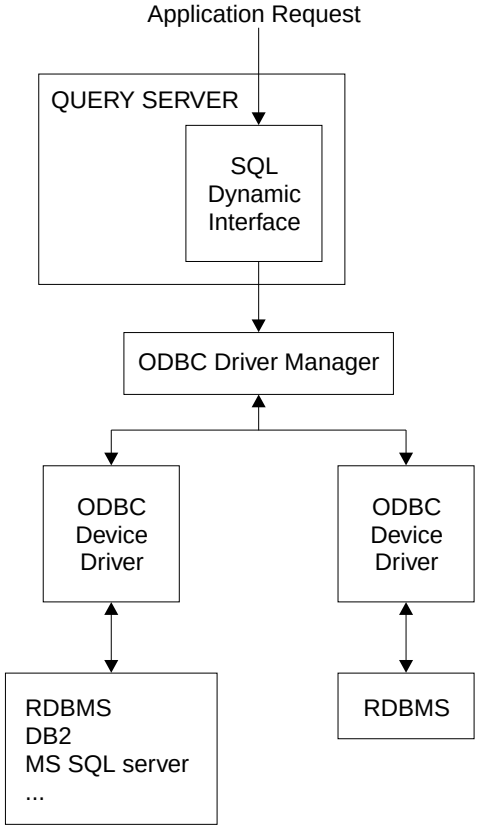



*Figure 5. ODBC query server Components and their Interaction with Applications and Databases*

© Copyright IBM Corp. 1992, 2002

The main objectives of this server are:
- Data integrity
- Shared multiple-station access
- Full SQL functions
- Common interface
- Application program portability
- Multiple-database access

Migration from LANDP for OS/2 query server to LANDP for Windows query server is aided by the `openodb` application that issues the extended version of the GF function.

## Implications of using ODBC as an access path to the RDBMS

The ODBC query server uses the ODBC specification to offer more flexibility and enable the server to work with different RDBMS vendors. This means that the behaviour and performance of the ODBC query server depends not only on the underlying RDBMS but on the ODBC device driver that is used.

ODBC contains 3 levels of support. The ODBC query server has been written to the ODBC conformance level 1. To support this, the ODBC device driver must conform to ODBC level 1 function API. A device driver can conform to a level of functionality without supporting every function or attribute of that level. To be able to execute DR and RR functions correctly, the ODBC device driver must be able to support positioned updates and deletes.

## Modes of operation

When a LANDP ODBC query server is installed and customized, it has two modes of operation:
- Not-logged-on mode
- Logged-in to SQL mode

### Not-logged-on mode

When the server is in its not-logged-on mode, some functions can be requested to enter and leave the logged-in mode and perform other duties like opening and closing additional sessions (see “Using the ODBC query server in not-logged-on mode” on page 360).

To switch between the SQL mode and not-logged-on mode, the ODBC server provides two server functions (OQ and CQ). Not-logged-on mode is the initial mode of the server after successful loading.

### Query mode

In this mode, the application program is able to send SQL statements through the SQ function and receive SQL responses through the LANDP network. Additional functions to fetch, insert, delete, and update a row are also available.

The query mode has a choice of external data types returned by the ODBC query server. The application can choose between the traditional LANDP for OS/2 query server data type interface or the ODBC data type interface. The differences between the two interfaces are as follows:

- The values used in the I/O structures for describing the types
- The actual data format returned, decimals, numerics, date and time types

Unlike the LANDP for OS/2 query server, there is currently no shared-file mode. If any of the shared-file mode functions from the LANDP for OS/2 query server are issued against the ODBC query server, a return code of AD (the function is not supported) is received.

The following example of a sequence of operations illustrates how the operation modes are entered:

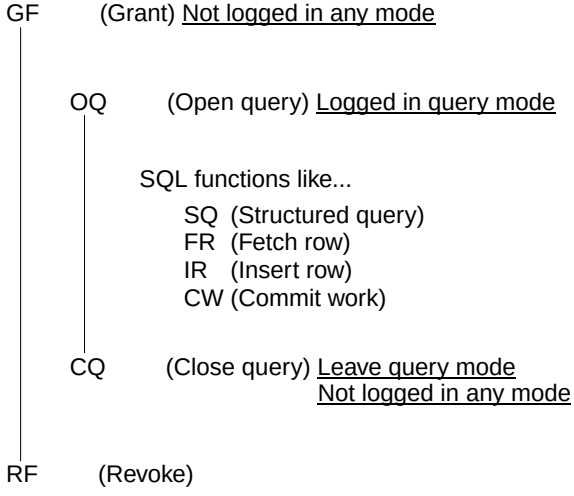


*Table 36. Function Codes used in the ODBC query server*

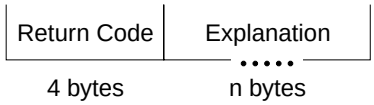
| Function code | Description | Env. | Page |
|---|---|---|---|
| **Functions available in any mode:** | | | |
| CS | Close session | --N- | 361 |
| HL | Habilitate (enable) logging | --N- | 362 |
| IL | Inhibit logging | --N- | 362 |
| OS | Open session | --N- | 363 |
| TS | Test status | --N- | 364 |
| **Functions available in not-logged-on mode:** | | | |
| CD | Change database | --N- | 360 |
| GF | Grant | --N- | 361 |
| OQ | Open query mode | --N- | 362 |
| RF | Revoke | --N- | 364 |
| **Functions available in query mode:** | | | |
| CQ | Close query mode | --N- | 368 |
| CW | Commit work | --N- | 368 |
| DQ | Describe query | --N- | 369 |
| DR | Delete row | --N- | 370 |
| DT | Describe table | --N- | 370 |
| EQ | End query | --N- | 371 |
| FR | Fetch row | --N- | 372 |
| IR | Insert row | --N- | 373 |
| RI | Obtain RDBMS information | --N- | 374 |
| RR | Replace row | --N- | 374 |
| RW | Rollback work | --N- | 375 |
| SP | Set parameters | --N- | 375 |
| SQ | Structured query | --N- | 376 |

The first and second columns of the table give the function code and the name of the function. The third column shows the operating environment of the function, as explained in “Operating environments” on page xxvi. “--N-” means that it is available from LANDP for Windows only. The last column refers to the page where you can find the function described.

## Return code and explanation handling

If an error is returned as a direct result of an SQL statement, the ODBC query server provides the 4-byte field containing the error code from the RDBMS. This code is the native SQL error code. Concatenated to the error code, the server provides the returned explanation. The length of the Reply DATA area determines how much explanation data can be returned.

For return codes caused by SQL statements (for example, QE), Reply DATA is as follows:




If the length fields are left blank in the Reply DATA and PARMLIST areas, the length is the same as specified in the CPRB when the request is made.

## I/O structure blocks

This section describes the I/O blocks passed to or returned (as parameters) from the server. The Column Name, Column Type, and Field Length fields are repeated for each column descriptor being transmitted.

### Short descriptor

The following block defines the Short Descriptor. The Field Type and Field Length fields are repeated for each field descriptor being transmitted.

| **Short Descriptor Block** | | | | |
|---|---|---|---|---|
| **Field Name** | **Offset** | **Size** | **Type** | **Content** |
| Queryhandle | 0 | 2 | Integer | Query handle under which current function has been performed |
| Numoffields | 2 | 2 | Integer | Number of field descriptors (field type and length) that follow |
| | | | | |
| Field Type | 4xN | 2 | Integer | Type of the table field that is described here |
| Field Length | 2+(4xN) | 2 | Integer | Length of the table field that is described (not including the space for null-indicators) |
| | | | | |
| **Note:** This is used in reply (SP, EQ, FR, DT, IR, DR, RR). | | | | |

**Note:** The field length does not include the two additional bytes needed for the indicator variable if null values are allowed. It also does not include the half bytes needed for the used length in the 'Varchar' field types (448, 449, 456, 457, 476, 477).

### Long descriptor

The following block defines the Long Descriptor. The Name Length and Field Name fields are repeated for each field descriptor being transmitted.

| Long Descriptor Block | | | | |
|---|---|---|---|---|
| **Field Name** | **Offset** | **Size** | **Type** | **Content** |
| Name Length | 32xN | 2 | Integer | Number of characters that are valid in the following column name |
| Field Name | 2+(32N) | 30 | Char | Name of the described column |
| | | | | |
| **Note:** This is used in reply (DT, DQ). | | | | |

## Pre-Fetch block

The block below describes the mechanism by which application programs can pre-fetch parameters to a SELECT statement before it is sent. The double lines in the table show that several parameters can be issued in the same call. The Parameter Value field is repeated for each field descriptor being transmitted.

| Pre-fetch Parameter Block | | | |
|---|---|---|---|
| **Field Name** | **Size** | **Type** | **Content** |
| | | | |
| Fieldtype (1) | 2 | Integer | Type of the parameter marker ('?') described here |
| Fieldlength (2) | 2 | Integer | Length of the parameter marker described (not including the space for null-indicators) |
| Parameter value | (2) | (1) | Parameter Value |
| | | | |
| **Note:** This is used in request (SP). | | | |

## Using the ODBC query server

This § provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields. If you need more information about the CPRB fields, see Appendix A, “Connectivity programming request block” on page 669.

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | 26 | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | | Request DATA length |
| 22 | 4 | Address | Request DATA address |

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 26 | 2 | 26 | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | EHCODB## | Server name |
| **Note:** The value of ## is defined during customization and relates to the particular ODBC query server with which the application program communicates. | | | |

The following fields are variable and are discussed in each function request description:
- Function code
- Request DATA length
- Reply DATA length

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 44 | 2 | | Replied PARMLIST length |
| 46 | 2 | | Replied DATA length |

If the request was successful, the *router return code* and the *server return code* are both X'00000000'. In all other cases, see the appropriate section in the *LANDP Problem Determination* book, to see if you should take any action. The return values in Reply PARMLIST and Reply DATA should be ignored if there is an error.

The following fields are variable and are discussed in each function request description:
- Replied PARMLIST length
- Replied DATA length

**PARMLIST:** Request PARMLIST and Reply PARMLIST fields are described for each function. Many functions share a common format for the Request PARMLIST area, which is given here for convenience. Not all fields are required for all functions, and some variants exist.

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Query handle (used in query mode) |
| 10 | 2 | Session identifier (used in all modes) |

**DATA:** Request DATA and Reply DATA contain data to be sent to or received from the server. The use of these areas is explained in the description of each function request.

## Using the ODBC query server in not-logged-on mode

Besides the functions that can be requested when in query mode, other functions are supplied to enter and leave the query mode, and to perform the following additional tasks:

- To start and end the server services (GF, RF), and to enter and leave query mode (OQ, CQ).
- To request and close an additional working session (OS, CS). The ODBC query server provides a multiple session capability which allows the application program to have more than one session open at a time. Each session has special attributes and a separate commit unit. Thus, an application program can work simultaneously on different databases.
- To provide server status feedback (TS).
- To change the database for processing (CD).

## Request reference in not-logged-on mode

These are the functions available to control sessions and connections. This section also includes functions that can be issued in any mode. (See Table 36 on page 356 to differentiate between the two types.) They are listed in alphabetical order of function code.

### Change database (CD function)

This function causes the corresponding issuing session to use the database identified in Request DATA. Before issuing a CD function, the session should have issued a CQ (close query mode) function call to make sure that all queries are closed (by a CQ function call). If all queries are not closed, the CD call fails.

The ODBC query server session user must be authorized to access the database that is to be used.

| CPRB Field | Content/Description |
|---|---|
| Function code | CD |
| Request DATA length | 60 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 20 | Database name to be used in ODBC query server session |
| 20 | 20 | User ID |
| 40 | 20 | Password to database |

## Close session (CS function)

This function closes a session previously opened with an OS function request. It can be used at any time with an opened session identifier. If the requesting session has an uncommitted opened transaction, a ROLLBACK is forced. All changes made to the database since the last COMMIT operation are lost. This function closes all open database handles for the session.

| CPRB Field | Content/Description |
|---|---|
| Function code | CQ |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier to be closed |

## Grant (GF function)

This function starts the server operation. The server rejects any request (except the session managing requests OS, CS, and TS) until this function is requested. This gives the database administrator (DBA) some control over the server. Optionally, for compatibility reasons, this function accepts one parameter byte in Request DATA with value ’0’. It must be requested only once by any of the served workstations.

**Note:** If database name, user ID, and password are not supplied in a GF call prior to an OQ, the OQ function fails.

| CPRB Field | Content/Description |
|---|---|
| Function code | GF |
| Request DATA length | 0, 1, or 60 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

**If Request DATA length is 1**

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | '0' |

**If Request DATA length is 60**

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 20 | Database name to be used in ODBC query server session |
| 20 | 20 | User ID |
| 40 | 20 | Password to database |

## Habilitate (enable) logging (HL function)

This function is not supported by the ODBC query server. To facilitate migration, the function is treated as a non-operation and no error code is returned.

## Inhibit logging (IL function)

This function is not supported by the ODBC query server. To facilitate migration, the function is treated as a non-operation and no error code is returned.

## Open query mode (OQ function)

This function logs in an application program to the ODBC query server in query mode. The requesting application program connects the session to the ODBC driver and is allowed to perform any of this mode's specific function requests.

This function also informs the ODBC query server whether the application program wishes to use the ODBC data type interface rather than the default LANDP for OS/2 query server data type interface.

| CPRB Field | Content/Description |
|---|---|
| Function code | OQ |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | 'O' - ODBS Data Type Interface. Any other value specifies the LANDP for OS/2 query server data type interface. |
| 10 | 2 | Session identifier (X'0000' is the main session) |

## Open session (OS function)

This function opens an additional working session for an application program. By requesting this function, the application program requests a session identifier that is used in following operations requested for the new session. Any function can be requested from the additional session in the same way as in the primary session. Request the desired function with the session identifier in Request PARMLIST offset X'000A'. The session identifier must always be specified when you close the session with a CS function request.

An additional session is like starting another application program from the same workstation. It has separate attributes (including working mode) and a separate commit unit (transaction integrity). The use of additional sessions enables an application to interact with different databases concurrently.

| CPRB Field | Content/Description |
|---|---|
| Function code | OS |
| Request DATA length | 0 or 60 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

If request DATA length is 60, the serve uses the database name, user ID, and password supplied in the GF function.

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 20 | Database name to be used in ODBC query server session |
| 20 | 20 | User ID |
| 40 | 20 | Password to database |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | New session identifier |

## Revoke (RF function)

When this function is requested, the server waits until all the opened transactions are closed and it prevents new transactions from starting. It is intended for the DBA program to control server operations and must be performed only once by any of the served workstations.

| CPRB Field | Content/Description |
|---|---|
| Function code | RF |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

## Test status (TS function)

This function determines the server status. The status can be open and servicing, or closed. The return code for open is X'0100 xxxx' and for close is X'0000 0000'. The function can be used to check the status of the ODBC query server.

| CPRB Field | Content/Description |
|---|---|
| Function code | TS |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Server return code | 256 if open, 0 if closed |
| Replied DATA length | 0 |

| CPRB Field | Content/Description |
|---|---|
| Replied PARMLIST length | 0 |

## Using the ODBC query server in query mode

You enter this mode with an open query (OQ) function request and end it with a close query (CQ) function request. When entering this mode, the application program can select which data type interface it wishes to use. The main function in this mode is the structured query (SQ) function, which allows the application program to perform any SQL statement that is supported by the underlying RDBMS. If the performed statement is a SELECT statement, the application program obtains a query handle together with the first row of the result table. The query handle allows the application program to fetch the rest of the result table using the fetch row (FR) function.

When all the rows have been fetched (read), the query handle is closed without the application program's intervention. The query handle can then be re-issued for following SELECT statements. This introduces the idea of the "Active Query", which is a statement that has outstanding resultant rows. You can close the query handle (closing the active query) by reading all the result rows, issuing an end query (EQ) function or issuing either a COMMIT (CW) or ROLLBACK (RW). The applicaton program can also issue a close query (CQ) function, which leaves the query mode but also closes all related query handles that are active.

This mode also offers a set of functions that allow the updating (RR), deleting (DR) and inserting (IR) of rows with respect to the current row of a given handle. These functions are especially useful because they help the application programmer to avoid building up complicated insert, update or delete SQL statements.

Other supplied functions ensure general data-integrity handling for committing or rolling back the transaction (CW, RW). You can also obtain descriptions of the table (DT) or a query's result table (DQ).

Finally, the query mode provides a function to enable the application program to use parameter markers in it's SQL statements (these parameter markers are represented by ? in SQL statements). This function (SP) allows the application program to "pre-fetch" parameters to SQL statements before the statement is sent using the SQ function.

There are two data type interfaces to the query mode. The first is the LANDP for OS/2 query server data type interface and the second is the ODBC data type interface itself. "Supported data types" on page 366 defines the mappings between the external (what the application program sees), the SQL data types (what the RDBMS sees) and the internal (what the ODBC query server expects). When using the ODBC data type interface, the values for describing data types have been taken from the ODBC specification. When extracting or inserting data into the RDBMS, the ODBC query server needs to know what data type the data is in and what data type the data is expected to be.

The ODBC specification states that the data types used by the application program should be native to the underlying RDBMS. However, not all RDBMSs use the same data types, and therefore an application written to a specific RDBMS may not work correctly when run against another RDBMS. ODBC does try to enable RDBMS-independent programs, but again the behavior can be affected by the underlying database.

## Supported data types

The supported data types are:

*Table 37. Data types allowed in structured query*

| | | | |
|---|---|---|---|
| *QS Field type (Hex)* | *ODBC Field type (Decimal)* | *Type* | *Field length* |
| 0180 / 0181 | 91 / 591 | Date Record | (note 1) |
| 0184 / 0185 | 92 / 592 | Time Record | (note 1) |
| 0188 / 0189 | 93 / 593 | Timestamp Record | (note 1) |
| 01c0 / 01c1 | 12 / 512 | Varying char string (2-byte length) | (note 3) |
| 01c4 / 01c5 | 1 / 512 | Fixed Length char string | n - bytes |
| 01e0 / 01e0 | 8 / 508 | Float (double precision) | 8 |
| 01e0 / 01e0 | 7/ 507 | Real | 4 |
| 01e0 / 01e0 | 6 / 506 | Float | 8 |
| 01e4 / 01e5 | 3 / 503 | Decimal | (note 2) |
| 01e8 / 01e9 | 2 / 502 | Numeric (zoned ) | (note 2) |
| 01f0 / 01f1 | 4 / 504 | 4-byte integer | 4 |
| 01f4 / 01f5 | 5 / 505 | 2-byte integer | 2 |
| 01f6 / 01f7 | -2 / 498 | Binary Data | n - bytes |

The following notes refer to Table 37.

1. The length of data and time fields depends on the interface selected. In the LANDP for OS/2 query server, data types, interface date, and time types are returned as character strings. Date strings are 10 bytes long, time strings are 8 bytes long, and timestamp strings are 26 bytes long. The format of the character string depends on the ODBC device driver. ODBC specifies the format `yyyy-mm-dd` for date types and `hh:mm:ss` for time types, but some ODBC device drivers may return a different format. When the ODBC data type interface is being used, the date and time type data is returned as structures (defined below as C data structures). In these cases, the field lengths are the size of the structures.

```
struct date_record
{
  short  year;
  short  month;
  short  day;
}
```

```
struct time_record
{
   short   hour;
   short   minute;
   short  second:
}
```

```
struct timestamp_record
{
   short  year;
   short  month;
   short  day;
   short  hour;
   short  minute;
   short  second;
   int  fraction;
}
```

2. Byte 1 = precision and Byte 2 = scale. In ODBC data type interface, the returned data is a character string. The length of the character string depends on the precision and the scale. If the scale is greater than 0, the length is precision + 2 (one for the sign and one for the decimal point), otherwise the length is the precision + 1 (for the sign).

   When using the LANDP for OS/2 query server data interface, these values are returned as packed decimals or packed numerics, where the length is (precision/2 + 1).

3. When using varying character strings (01c0/01c1), have a 2-byte field preceding each string in the data stream (this field contains the length of the string). The field length description in the short descriptor is the maximum length of that column. When using the ODBC data type interface and issue input functions (IR, RR and SP), the number of bytes expected for a varying character string is given in the preceding 2-byte field. At all other times, the number of bytes expected is the maximum length of the column.

   ***Example*** (data values are in hex )

```
SQL column definition

LANDP for OS/2 query server data type interface

SQ returned data       00 00 03 00 41 42 43 00 00 00 00 00 00 00 xx xx
IR input data          00 00 03 00 41 42 43 00 00 00 00 00 00 00 xx xx


ODBC data type interface

SQ returned data       00 00 03 00 41 42 43 00 00 00 00 00 00 00 xx xx
IR input data          00 00 03 00 41 42 43 xx xx
```

Date values are in hexadecimal. The first two bytes represent the null indicator, the second two the field length. In all occurrences except for the input data when using ODBC data type interface, the next ten bytes represent the data for the vchar_column. For the input date when using the ODBC data type interface the next 3 bytes represent the data. The xx's represent another data field in the result data set.

The QS Field Type and ODBC Field Type columns each contain 2 entries. The first is the value used to describe a field type that does not have a null indicator and does not allow null values. The second describes the field type that does have a null indicator and does allow null values. If the field type allows null values, an additional indicator variable (2-bytes) is added preceding the data field (precedes the length fields for 01c0 / 01c1 types). However, this addition is not reflected in the field length.

## Request reference for ODBC query server in query mode

These are the functions available in query mode. They are listed in alphabetical order of function code.

### Close query mode (CQ function)

This function logs off an application program from the ODBC query server in query mode. After requesting this function, the requesting application program returns to the not-logged mode. If the requesting session has an uncommitted opened transaction, a COMMIT function is performed. Open query handles are closed.

| CPRB Field | Content/Description |
|---|---|
| Function code | CQ |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

### Commit work (CW function)

This function terminates a transaction and commits the database changes made during that transaction. It has the same effects as requesting SQ with the sentence 'COMMIT WORK' in Request DATA. The query handles are closed.

| CPRB Field | Content/Description |
|---|---|
| Function code | CW |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 or (to obtain error information) ≥ 4 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0<br>or<br>Length of error code + error message |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

## Describe query (DQ function)

This function obtains information about a given query. It is used to get feedback about the result columns of an SQL SELECT statement. The SQL statement is not performed. The descriptor is returned in the Reply PARMLIST area. A specific value, the null handle (X'FFFF'), is returned in the short descriptor block of the Reply PARMLIST. This can be used along with the IR function request to insert new rows into the table. The names of the query columns are returned in Reply DATA in the long-descriptor-block format.

| CPRB Field | Content/Description |
|---|---|
| Function code | DQ |
| Request DATA length | Length of the SELECT statement |
| Request PARMLIST length | 26 |
| Reply DATA length ≥ 1 or (to obtain error information) ≥ 4 | |
| Reply PARMLIST length | 26 |
| Replied DATA length | Long descriptor block length |
| Replied PARMLIST length | Length of descriptor |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | SELECT statement |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Short descriptor block |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Long descriptor block or error information |

## Delete row (DR function)

This function deletes the row at which the current query handle points.

**Note:** Some ODBC device drivers will need the query handle identifying the table to be updated to have been opened with the FOR UPDATE clause.

| CPRB Field | Content/Description |
|---|---|
| Function code | DR |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 or (to obtain error information) ≥ 4 |
| Reply PARMLIST length | 26 |
| Replied PARMLIST length | Length of descriptor |
| Replied DATA length | 0 |
| Replied PARMLIST length | Length of descriptor |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Query handle returned by the SQ function |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Short descriptor block |

## Describe table (DT function)

This function obtains information about a table. Send the table name listed in Request DATA. The same information is returned as in SQ and FR function requests plus the names of the respective fields. These field names are in Reply DATA in the long-descriptor-block format. A specific value, the null handle (X'FFFF'), is returned in the short descriptor block of the Reply PARMLIST area. This can be used along with the IR function to insert new rows into the table.

| CPRB Field | Content/Description |
|---|---|
| Function code | DT |
| Request DATA length | Length of table name |
| Request PARMLIST length | 26 |
| Reply DATA length | ≥ 1 or (to obtain error information) ≥ 4 |
| Reply PARMLIST length | 26 |
| Replied DATA length | Long descriptor block length |
| Replied PARMLIST length | Length of descriptor |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Table name |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Short descriptor block |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Long descriptor block or error information |

## End query (EQ function)

Using this function is one way to close an opened query handle without reading the remaining rows. If you request an FR function and the return code is ED (end of data), the query handle is also closed. The CQ, CW, and RW function requests also close all the query handles.

| CPRB Field | Content/Description |
|---|---|
| Function code | EQ |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 or (to obtain error information) ≥ 4 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0<br>or<br>Length of error code + error message |
| Replied PARMLIST length | Length of descriptor |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Query handle returned by SQ |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Short descriptor block |

## Fetch row (FR function)

This function obtains the next row of a query sent with the SQ (Structured Query) function. If the return code is ED, the query handle will be closed.

| CPRB Field | Content/Description |
|---|---|
| Function code | FR |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | ≥ 1 or (to obtain error information) ≥ 4 |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of data fetched |
| Replied PARMLIST length | Length of descriptor |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Query handle returned by SQ |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Short descriptor block |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Data fetched or error information |

## Insert row (IR function)

This function inserts a new row into the table implicitly specified by the query handle or explicitly through a table name. The short descriptor is also returned in Reply PARMLIST.

| CPRB Field | Content/Description |
|---|---|
| Function code | IR |
| Request DATA length | Length of row |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 or (to obtain error information) ≥ 4 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0<br>or<br>Length of error code + error message |
| Replied PARMLIST length | Length of descriptor |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Query handle returned by SQ or null handle returned from DT or DQ |
| 2 | 1 | Method identifier. If this is set to 'T', the table name at offset 12 is used to insert the row. |
| 10 | 2 | Session identifier (X'0000' is the main session) |
| 12 | | Table name |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Row to insert |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Short descriptor block |

## Obtain RDBMS information (RI function)

This function obtains information concerning the current active path database.

| CPRB Field | Content/Description |
|---|---|
| Function code | RI |
| Request DATA length | 0 |
| Reply DATA length | ≥ 69 |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of RDBMS information |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 20 | Name of the DBMS product being accessed by the ODBC Driver. |
| 20 | 20 | Name of the database currently in use. |
| 40 | 20 | Name of the driver being used to access the data source. |
| 50 | 10 | Version of the driver being used. |
| 60 | 5 | The version of ODBC being used by the driver. |
| 65 | 4 | ODBC Conformance level (format integer) - 1 -Core, 2 - Level 1, 3 - Level 2. |

## Replace row (RR function)

This function replaces the row to which the current query handle points, with the row supplied in Request DATA.

The query handle identifying the table to be updated must have been opened with the FOR UPDATE clause.

| CPRB Field | Content/Description |
|---|---|
| Function code | RR |
| Request DATA length | Length of row |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 or (to obtain error information) ≥ 4 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 or<br>Length of error code + error message |
| Replied PARMLIST length | Length of descriptor |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Query handle returned by SQ |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | New row to replace old row |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Short descriptor block |

## Rollback work (RW function)

This function is used to roll back work. It terminates a transaction and cancels the database changes that were made during that transaction. It has the same effect as requesting SQ with the sentence 'ROLLBACK WORK' in Request DATA. The query handles are closed.

| CPRB Field | Content/Description |
|---|---|
| Function code | RW |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 or (to obtain error information) ≥ 4 |
| Replied DATA length | 0 or<br>Length of error code + error message |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values for LANDP for OS/2 | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

## Set parameters (SP function)

This function sets the parameters to be used with the next SQ function. Parameters are passed in the Request DATA in the format of *pre-fetch parameters*, described on page “Pre-Fetch block” on page 358.

You can send one or more parameters within the same SP function request. Following SP requests stack the parameters in a parameter area. The parameters are used in the same order as they are sent to the server. They are deleted when an SQ or a CQ function is requested.

**Note:** The SP function does not expect NULL values. If a NULL values is included in a NULL indicator (in the pre-fetch block), the return code ED will be returned by the SQ function. If you wish to use a NULL values as part of the search criteria or as part of an INSERT INTO statement then this should be written into the statement. For example:

```
, SELECT * FROM CUSTOMER WHERE BALANCE IS NULL
```

Also, when using varying character string types, the length of data expected is defined in the length field of the pre-fetch block.

| CPRB Field | Content/Description |
|---|---|
| Function code | SP |
| Request DATA length | Length of query parameters to set |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 2 | Session identifier (X'0000' is the main session) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Pre-fetch parameters block |

## Structured query (SQ function)

This function sends and executes the SQL sentence found in Request DATA. If the sentence is performed successfully, the return code is zero.

All SQL statements requested through the SQ function can only contain valid language elements for the specific SQL.

If the sentence is a SELECT statement, the function returns the first row of the selected query result. The data returned in Reply PARMLIST is a descriptor of the row thus obtained. If the result of the query is null, (no rows), the return code is ED. The format of the descriptor is a short descriptor block. The first two bytes of the short descriptor block represent the query handle, a number that identifies this SELECT statement as

an “opened query”. Following rows can be obtained by requesting the FR function using the query handle returned. The query handle must be used for retrieving and updating the rest of the data with the function requests FR, RR, and IR.

If you are going to use the RR function request at the returned identifier, the query handle identifying the table to be updated must have been opened with the FOR UPDATE OF clause for each column to be updated.

| CPRB Field | Content/Description |
|---|---|
| Function code | SQ |
| Request DATA length | Length of query |
| Request PARMLIST length | 26 |
| Reply DATA length | ≥ 1 or (to obtain error information) ≥ 4 |
| Reply PARMLIST length | 26 |
| Replied DATA length | Length of data fetched |
| Replied PARMLIST length | Length of descriptor |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Query handle |
| 10 | 2 | Session identifier (X'0000' is the main session) |
| **Note:** The query handle is a number which is used by the ODBC query server to identify the query. | | |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Query statement |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Short descriptor block |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Data fetched or error information |

## ODBC query server

# Chapter 17. Electronic journal server

The electronic journal server allows the application programmer to create, manage, and process data in electronic journal files. These files can, for example, contain all the transactions performed in your branch office.

The electronic journal server offers:

- Access to and the handling of the records stored in the electronic journal using functions for adding, retrieving, updating, and deleting (logically)

  A search mechanism can be used to search for specific records in electronic journals, see “Search operations” on page 385.
- Application program control of the electronic journal files using functions for creating, accessing, and reusing the physical disk space

Once the electronic journal server is loaded, the application must wait until this server is initialized and operational. The application must issue a test initialization (TI) function call to verify that the server is initialized and operational.

*Table 38. Function Codes used in the Electronic Journal Server*

| Function code | Description | Env. | Page |
|---|---|---|---|
| AL | Allocate physical electronic journal | 02NL | 392 |
| AR | Add record | 02NL | 392 |
| DA | Deallocate physical electronic journal | 02NL | 394 |
| DL | Delete record | 02NL | 394 |
| IL | Inquire logical | 02NL | 395 |
| IP | Inquire physical | 02NL | 396 |
| RL | Release electronic journal | 02NL | 398 |
| RR | Retrieve record | 02NL | 398 |
| RS | Reset electronic journal file | 02NL | 400 |
| SL | Select current electronic journal | 02NL | 401 |
| TI | Test initialization | 02NL | 402 |
| TS | Terminate session | 02NL | 402 |
| UP | Update record | 02NL | 403 |

The first and second columns give the function code and the name of the function. The third column shows the operating environment of the function. “02NL” means that it is available from LANDP for DOS, LANDP for OS/2,r LANDP for Windows, or LANDP for Linux servers. The last column refers to the page where you can find the function described.

© Copyright IBM Corp. 1992, 2002

## Electronic journal environments

During customization, you define whether you are going to use *physical* or *logical* journals:

**Physical journal environment**
A physical electronic journal is a file. Records are directly added to and accessed in the physical journals. In the physical journal environment no logical journals exist.

**Logical journal environment**
Each file, a physical electronic journal, is divided into parts. Logical journals are accessed by applications using logical names. Many times the logical journal environment offers operational advantages.

## Physical electronic journals

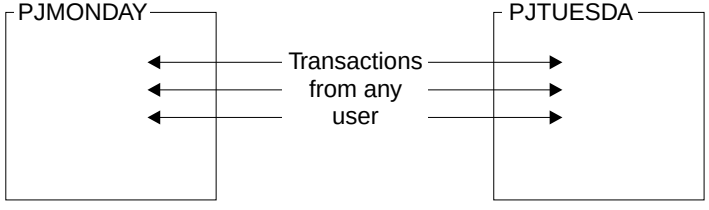
A *physical journal name* is assigned to each electronic journal file.

In this environment, records are always added to, updated in, deleted from, or retrieved from a specific journal using the physical journal name. A physical journal is, for example, used to store records for:

- A particular period
- A specific user
- A specific user during a particular period

### Example 1

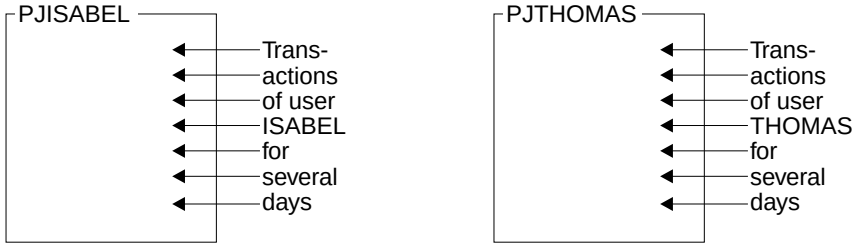
Physical electronic journals are dynamically associated with a particular period. One physical journal is allocated to the transactions for Monday, another to Tuesday, and so on. The physical journal names are PJMONDAY, PJTUESDA, and so on.




All users add their records to today's physical journal. Every workday a new physical electronic journal is allocated. Earlier physical journals are kept for an appropriate period.

### Example 2

Associate each physical electronic journal with a particular user. One physical journal is, in this example, allocated to user ISABEL, and another to user THOMAS.




**Note:** Records can be added to any physical electronic journal. The application is responsible for selecting the proper journal.

### Example program to process physical journals

Five journaling periods are kept. Therefore, five physical journals (files) have to be defined at customization. Each workday a new physical journal is used.

The daily procedure could be as follows:

1. Start of day:
   a. Release oldest journal (RL function). This electronic journal should be saved if required.
   b. Reset oldest journal (RS function).
   c. Allocate (AL function) today's journal to the now empty file. That is, assign a journal name to it, for example PJMONDAY.
2. During business day:
   - Records are added (AR function) to today's journal.
   - Records are processed, that is, they are:
     – Retrieved (RR function) and displayed
     – Updated (UP function)
     – Deleted (DL function)
3. End of day:
   a. If cash balance is not correct, search the journal, for example, for a specific:
      - Amount
      - Transaction code
      - Time
   b. Correct the journal.

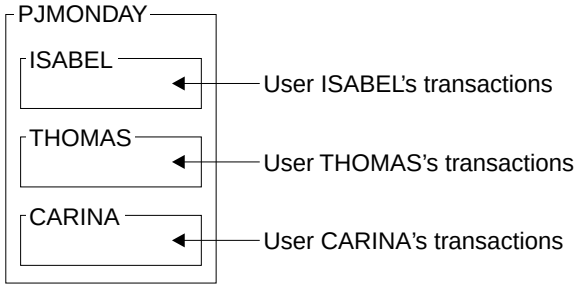
## Logical electronic journals

The records within a physical journal (file) can also be grouped. The resulting group is called a logical journal and is accessed by the application using the logical journal name. This allows you to split a physical journal into many generations of logical journals. The electronic journal server supports up to 99 logical journals per physical journal. The maximum number of logical journals per physical journal to be used is defined during customization.

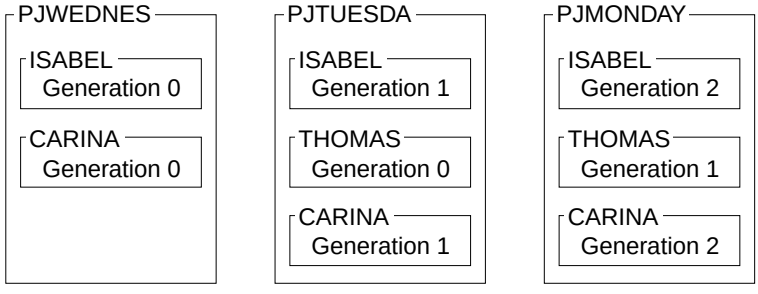
Typical examples of logical journal names used for access are:
- User ID
- Employee number or name
- Workstation ID
- Cash box number

In this environment the records are always accessed using the logical journal name. As logical journals are part of physical journals, physical journals must be maintained as in the physical journal environment, that is, they must be reset (RS function) and allocated (AL function).

In the following example, the relationship between a physical journal and its logical journals is shown. Three users add records to and retrieve records from their own logical journals which are part of physical journal PJMONDAY.




Multiple generations of logical journals are represented by multiple physical journals. Here is an example of three generations of logical journals.




One logical journal represents the current generation and the others represent previous generation journals. Data retrieval, updating, or deletion is supported for all journal generations, but records can only be added to the current generation.

It is not mandatory that all users have their current logical journal in the same physical journal. It is more than likely that the current logical journals are located in different physical journals at any given time. This depends on your installation.

### Example program to process logical journals

Up to five journaling generations are kept and up to 20 tellers use the electronic journals. Therefore, five files (physical journals) and 20 logical journals have to be defined during customization.

The daily procedure could be as follows:

1. Start of day:
   a. Tellers release (RL function) the oldest logical journals. (The physical journal is released when all associated logical journals have been released.) Between the use of the functions RL and RS the electronic journal should be saved if necessary.
   b. Reset (RS function) the released physical journal.
   c. Allocate (AL function) today's physical journal, for example PJ900123.
   d. Tellers select (SL function) today's physical journal to be current. (After the function SL has been performed the previous current logical journal becomes the most recent previous generation.)
2. During business day:
   a. Records are added (AR function) to the current generation of logical journals.
   b. Records are processed, that is, they are:
      - Retrieved (RR function) and displayed
      - Updated (UP function)
      - Deleted (DL function)
3. End of day:
   a. If cash balance is not correct, search the journal, for example, for a specific:
      - Amount
      - Transaction code
      - Time
   b. Correct journal

## Data integrity of electronic journal files

Data integrity of the electronic journal files is, to a large extent, provided by the shared-file server (see Chapter 14, "Shared-file server" on page 255). The shared-file server contains data integrity functions:

**BT** Begin transaction

**ET** End transaction

Between BT and ET a sequence of record operations is performed. The accessed records are locked (held) until ET or RB is issued. When ET is issued, the records used since the last BT can no longer be changed. The BT–ET sequence is the normal error free case.

When technical or operational problems occur, you can at any time use:

**RB** Rollback

When this function is used all record manipulations done since the last BT are canceled.

There are two data integrity options for electronic journals that you can specify during customization. They are defined in keyword SEPSESS in the EJOUPRF vector.

1. SEPSESS=Y.

   If the Y (Yes) option is chosen, the electronic journal server obtains a separate shared-file server session for data access. The electronic journal server issues the data integrity functions BT and ET, and if necessary RB.

   Opening and closing of a separate shared-file session is controlled by a flag in the relevant electronic journal function:

   **'F'** Session to begin
   **'M'** Session to continue
   **'L'** Session to end
   **'O'** Session to begin and to end

   Example of use of the separate session option to add four records:

   ```
   AR  with option 'F'        (first record)
   AR  with option 'M'        (second record)
   AR  with option 'M'        (third record)
   AR  with option 'L'        (fourth record)
   ```

   The electronic journal server issues a BT when encountering a record with the option 'F' and an ET when the option is 'L'.

   You can explicitly terminate a session using the terminate session (TS) function.

2. SEPSESS=N. This is the default.

   If the N (No) option is chosen, the application must issue the BT and ET shared-file server functions. The electronic journal server does not issue the BT or ET function.

   Example of use of the single session option:

   ```
   BT
   RR (for deleting)
   DL
   ET
   ```

*Automatic Rollback:*

The following return codes in the specified function cause an RB (*rollback*) function to be requested automatically (if a separate session option is selected):

| Function | Return code |
|---|---|
| **AR — Add record** | P1, P2, PH, PI, JC, JD |
| **RR — Retrieve record** | P1, JA, JC, JD, JF |
| **DL — Delete record** | P1 |
| **UP — Update record** | P1, PH, PI, PL |

After the DL and UP functions the lock (hold) is released and another RR function must be requested to use another record.

## Search operations

The electronic journal and the store-for-forwarding servers can retrieve a record depending on search criteria submitted by the application program. The field names of a defined record are used to define the search criteria.

## Relationship to the record format definitions

There is a relationship between the search capabilities of the electronic journal or store-for-forwarding servers and the record format definitions of the individual records to be journaled or stored.

When the application program passes a record to be journaled it also passes the name of the record format definition describing that particular record. With this information the electronic journal server can locate the individual fields in that record. It can also check if the record contains key fields. If so, it can locate such fields and copy them into the fixed header portion of the records. This is required because the underlying shared-file server needs the key fields at fixed positions in the shared-file record. In this way the key fields can be located in the individual records although they may be at a physically different position in the records.

When the application program searches for records matching special search criteria, the electronic journal (or the store-for-forwarding) server retrieves a record. It identifies the record format name stored with that particular record. It can then retrieve the format description of that record and locates the individual fields of the record for comparison. It can then check if a particular record contains a specific field requested in a search operation.

The serves use the customization information, to recognize the names of key fields. When a server detects a key field in a search argument, it accesses the record through the shared-file server key.

The same mechanism is used to search for fields that are not key fields using the record format definition such fields can also be located in the individual records.

## Search definition

The application program requesting a retrieve operation defines a character string describing the type of search operations to be performed.

The supported operations are a functional subset of SQL. The syntax used is similar to SQL. However, unlike SQL, only the first (or previous or next) record matching the search criteria is returned to the application program.

The character string defining the search criteria consists of one or more comparisons of the form:

| FIELD NAME | COMPARISON | OPERATOR VALUE |
|---|---|---|

**Field name**
The name of the field in the record as defined during customization using the record format definition. Variable length fields are not allowed.

**Comparison operator**
Specifies a comparison between the content of FIELD NAME and VALUE.

| | |
|---|---|
| = | Equal to |
| ^= | Not equal to |
| <> | Not equal to |
| != | Not equal to |
| > | Greater than |
| >= | Greater than or equal to |
| ^> | Not greater than |
| !> | Not greater than |
| < | Less than |
| <= | Less than or equal to |
| ^< | Not less than |
| !< | Not less than |

Compound comparison operators (>=, <=, <>, ^=, !=, ^>, !>, ^<, and !<) may not be separated by a blank.

**Value** Defines the value with which the contents of FIELD NAME should be compared. The following rules apply:

- **Character Strings (store format C)**
  Character values are enclosed in apostrophes ('). Apostrophes as character values are written using two apostrophes. Within quotes, comparison operators (for example **=** ) are allowed as a part of a character string value. Examples:

  ```
  ABC is written 'ABC'
  A'S is written 'A''S'
  <(A is written '<(A'
  ```

  The length of the string *must be exactly equal to* the length of the defined field.

- **Numeric values (store format N, B, I, J, P)**
  Numeric values are coded without apostrophes conforming with the field format defined during customization. They are specified as, for example:

  ```
  34567
  12.47
  +0088 (same as 88, leading zeroes and plus sign not needed)
  -35.2300 (same as -35.23, trailing zero decimals not needed)
  ```

- **Hexadecimal (store format H)**
  Hexadecimal values are enclosed in apostrophes ('). The data has to be in the same format as it is stored in memory.

The length of the string *must be exactly equal to* the length of the defined field. If the value '27' (hex) belongs to the data string, no special action is required, as shown:

```
X'015D4C4C7070202626' is written as ' ]LLpp &&;amp;'
X'015D274C7027272626' is written as ' ]'Lp''&&;amp;'
```

## Using the logical operators AND and OR

Search definitions can be composed of several comparisons and the logical operators AND and OR. These are reserved keywords and cannot be used as field names. In the following discussion of their use, the symbol **C** is used here to denote a comparison. Rules for using the logical operators are:

- AND chains can be used. They are composed of comparisons and the AND logical operator.

  Example: **C** AND **C** AND **C**
- OR chains can be used. They are composed of comparisons and the OR logical operator.

  Example: **C** OR **C** OR **C** OR **C**
- Search definitions combining AND and OR chains must be defined using parentheses around all OR chains.

  Example: **C** AND (**C** OR **C**) AND **C**

  Example: (**C** OR **C** OR **C** OR **C**) AND (**C** OR **C**)

Up to 23 iterations are possible. This results in 24 OR and 24 AND terms connecting 25 comparisons. This is also true for the format with mixed AND and OR operators, it can also have 24 ANDs connecting 25 comparisons. The ORs in this format do not add to the limit of 24 ANDs. Each pair of parentheses reduces the number of possible comparisons by 2.

The following is an example of a search definition string:

```
(AMOUNT=50000 OR AMOUNT=60000) AND DATE>102888 AND OPER_ID='TELLER01'
```

This results in a search operation for records with these characteristics:

- AMOUNT being 50000 or 60000
- DATE being greater than 102888
- OPER_ID being TELLER01

## Search processing

All fields that are used as shared-file server key fields must be defined as such during customization. Each key field in a specific shared file must have a unique field name.

The FIELD NAMEs used in the search definitions can, but need not, be defined as shared-file server secondary keys. If a FIELD NAME is also a key field, processing is faster.

An example: Suppose there is a search definition, where DATE is a secondary key, but AMOUNT is not:

```
AMOUNT > 30000 AND DATE = 101089
```

The processing in this example means that records with DATE equal to 101089 are accessed through the shared-file server keyed access. However, if neither DATE, nor AMOUNT is a secondary key, *all* records are sequentially read and compared by the server.

If a search definition contains multiple shared-file server key fields, the first in the search definition is used as the accessing key. If both AMOUNT and DATE, in the above example, were defined as key fields, all records with AMOUNT > 30000 are accessed.

Some search examples are now given.

1. Search the first record with AMOUNT equal to 1000.

```
Search mode flag:   F
Search string:      AMOUNT = 1000
```

2. Search all records with AMOUNT equal to 1000 in ascending order.

```
* Standard procedure *
Step 1:  Search mode flag: F
         Search string:    AMOUNT = 1000

Step 2:  Search mode flag: N
         Search string:    AMOUNT = 1000

Repeat Step 2 until error code PC is returned.
```

```
* Optimized procedure *
This procedure can only be used if AMOUNT is key field.
Step 1:  Search mode flag: F
         Search string:    AMOUNT = 1000

Step 2:  Search mode flag: N
         Search string:    AMOUNT >= 1000

Repeat Step 2 until error code PC or a record with AMOUNT
greater than 1000 is returned.
```

3. Search all records with AMOUNT equal to 1000 in descending order.

```
* Standard procedure *
Step 1:  Search mode flag: P
         Search string:    AMOUNT = 1000

Repeat Step 1 until error code PC is returned.
```

```
* Optimized procedure *
This procedure can only be used if AMOUNT is key field
Step 1:  Search mode flag: F
         Search string:    AMOUNT > 1000
         (To move the shared-file PCB to the correct position.
         The returned record should be discarded).

Step 2:  Search mode flag: P
         Search string:    AMOUNT <= 1000

Repeat Step 2 until error code PC or a record with AMOUNT
less than 1000 is returned.
```

4. Search all records with AMOUNT greater than 1000 in ascending order.

```
* Standard procedure *
Step 1:  Search mode flag: F
         Search string:    AMOUNT > 1000

Step 2:  Search mode flag: N
         Search string:    AMOUNT > 1000

Repeat Step 2 until error code PC is returned.
```

5. Search all records with AMOUNT greater than 1000 in descending order.

```
* Standard procedure *
Step 1:  Search mode flag: P
         Search string:    AMOUNT > 1000

Repeat Step 1 until error code PC is returned.

* Optimized procedure *
This procedure can only be used if AMOUNT is key field
Step 1:  Search mode flag: P
         Search string:    AMOUNT > 0

Repeat Step 1 until error code PC or a record with AMOUNT
less than or equal to 1000 is returned.
```

6. Search all records with AMOUNT less than 1000 in ascending order.

```
* Standard procedure *
Step 1:  Search mode flag: N
         Search string:    AMOUNT < 1000

Repeat Step 1 until error code PC is returned.

* Optimized procedure *
This procedure can only be used if AMOUNT is key field
Step 1:  Search mode flag: N
         Search string:    AMOUNT < 9999

Repeat Step 1 until error code PC or a record with AMOUNT
equal to or greater than 1000 is returned.
```

7. Search all records with AMOUNT less than 1000 in descending order.

```
* Standard procedure *
Step 1:  Search mode flag: P
         Search string:    AMOUNT < 1000

Repeat Step 1 until error code PC is returned.

* Optimized procedure *
This procedure can only be used if AMOUNT is key field
Step 1:  Search mode flag: F
         Search string:    AMOUNT ≥ 1000

         (This is to move the Shared-File PCB to the correct
         position.  The returned record should be discarded).

Step 2:  Search mode flag: P
```

```
Search string:    AMOUNT < 1000

Repeat Step 2 until error code PC is returned.
```

### Search hints

- An AR function requested after an RR function in the same transaction can move some of the shared-file pointers, so the next RR function may return unexpected records.
- Take care when modifying the search criteria in a transaction because different shared-file pointers can be used and not all pointers are at the same data set position.
- A record definition can include fields named 'AND' and 'OR', but these field names cannot be used as search arguments. Field names like 'ANDXXX' or 'ORXXX' can be used in a search argument.

## Using the electronic journal server

This section provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields. If you need more information about the CPRB fields, see Appendix A, "Connectivity programming request block" on page 669.

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | 26 | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | 26 | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 26 | 2 | 26 | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | ELECJO## | Server name |
| **Note:** The value of ## is the unique server identifier for the client workstation as defined during customization. | | | |

The following fields are variable and are discussed in each function request description:

- Function code
- Request DATA length
- Reply DATA length

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 44 | 2 | | Replied PARMLIST length |
| 46 | 2 | | Replied DATA length |

If the request was successful, the *router return code* and the *server return code* are both X'00000000'. In all other cases, see the appropriate section in the *LANDP Problem Determination* book to see if you should take any action. The return values in Reply PARMLIST and Reply DATA should be ignored if there is an error.

The following fields are variable and are discussed in each function request description:
- Replied PARMLIST length
- Replied DATA length

**PARMLIST:** The Request PARMLIST and Reply PARMLIST fields for the electronic journal server are:

| Offset | Length | Content |
|---|---|---|
| 0 | 1 | Flag1: BT/ET flag |
| 1 | 1 | Flag2: Next or previous flag |
| 2 | 1 | Flag3: Search mode |
| 3 | 1 | Flag4: Delete flag |
| 4 | 1 | Flag5: Hold flag |
| 5 | 1 | Reserved |
| 6 | 2 | Journal generation number, see Note 1 |
| 8 | 2 | Maximum access number, see Note 1 |
| 10 | 8 | Logical journal name or physical journal name |
| 18 | 8 | Record format name |

**Notes:**

1. LANDP for DOS, LANDP for OS/2, and LANDP for Linux: The journal generation number and the maximum access number are stored in the normal personal system format, in a byte-reversed form. All other fields are in character format.
2. Flags not used must be filled with blanks (X'20').

**DATA:** Request DATA and Reply DATA contain data to be sent to or received from the server. The use of these areas is explained in the description of each function request.

## Obtaining database error messages

The following functions return a database error message when a database error occurs: AR, DL, DR, RR, RS, and UP. By checking the Replied DATA length, you can determine if a database error has occurred.

If the Replied DATA length is greater than zero and different from that shown for the function, a database error *has* occurred. The Reply DATA then contains the database error message. You must specify a Reply DATA length that is large enough to avoid a truncation of this error message.

## Request reference

These are the functions that can be used with the electronic journal server. They are listed in alphabetical order of function code.

## Allocate physical electronic journal (AL function)

This function allocates the next available physical journal and assigns a physical journal name to it. This function is required for the logical and physical environments. Electronic journal data sets are allocated from FBEJ00 to FBEJxx. When all data sets are in use, the electronic journal server looks for the next data set in wrap around mode, so the oldest one is taken. If this oldest data set is not empty (because a reset function has not been requested), a J8 return code is supplied to the application program even if some other data sets of the list are empty. This applies to logical and physical environments.

You can only have one allocated file that is without records, or not yet selected.

| CPRB Field | Content/Description |
|---|---|
| Function code | AL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 8 | Physical journal name |

## Add record (AR function)

This function adds a record to an electronic journal.

**Physical electronic journal environment**
Adds a record to the respective physical electronic journal. This can be any allocated physical journal.

**Logical electronic journal environment**
Adds a record to the *current* electronic journal.

| CPRB Field | Content/Description |
|---|---|
| Function code | AR |
| Request DATA length | Length of record to be added: 0 to 4064 |
| Request PARMLIST length | 26 |
| Reply DATA length | 32 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 32 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | BT/ET flag (separate session only). Specify one of:<br>**'F'** First<br>**'M'** Middle<br>**'L'** Last<br>**'O'** Only |
| 10 | 8 | Logical journal name or physical journal name |
| 18 | 8 | Record format definition name<br><br>If a search key field that is defined in the key names table is missing in the record format definition, the corresponding search key field in the physical record header is set to low values according to the field format specified. |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | See above | Data record to be added to electronic journal file |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| **'':** Information from the header stored with the record. The header is added to the stored record by the electronic journal server. | | |
| 0 | 2 | Length of record (as supplied on request) |
| 2 | 14 | System date and time (yyyymmddhhmmss) |
| 16 | 5 | Sequence number (for server internal use) |
| 21 | 2 | ID of originating workstation |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 23 | 1 | Record status, set to 'I' |
| 24 | 8 | Format name (as supplied at request) |

## Deallocate physical electronic journal (DA function)

This function deallocates a file (physical electronic journal). The file is marked as eligible for reuse. This function can be executed only if no records have been added and it has not been selected (SL function). Only the last file in the chain can be deallocated.

| CPRB Field | Content/Description |
|---|---|
| Function code | DA |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 8 | Physical journal name |

## Delete record (DL function)

This function deletes a record previously retrieved-for-deleting with an RR function with hold flag = 'H'. The RR and DL functions must be requested within the same transaction.

In the logical electronic journal environment, records can only be deleted by the "owner" of the logical electronic journal that initially created the record.

The record status of deleted records is set to 'D'. No other fields in the header are updated. The data of the journal record remains unchanged and the record can be found on following search operations.

| CPRB Field | Content/Description |
|---|---|
| Function code | DL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 32 |

| CPRB Field | Content/Description |
|---|---|
| Reply PARMLIST length | 26 |
| Replied DATA length | 32 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | BT/ET flag<br>**'M'** Not the last record (middle)<br>**'L'** Last record |
| 10 | 8 | Logical journal name or blank (if physical electronic journal) |
| 18 | 8 | Record format name |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Length of record |
| 2 | 14 | System date and time (yyyymmddhhmmss) |
| 16 | 5 | Sequence number |
| 21 | 2 | Originating workstation ID |
| 23 | 1 | Record status, changed to 'D' |
| 24 | 8 | Format name |

## Inquire logical (IL function)

This function allows an application program to retrieve:

- Names of all physical electronic journals that contain a logical electronic journal with a specified logical electronic journal name. A list of all physical electronic journals associated with a logical electronic journal and the corresponding process control flags is returned.
- A list of all logical journal names present, but only in the logical electronic journal environment.

| CPRB Field | Content/Description |
|---|---|
| Function code | IL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 1024 |
| Reply PARMLIST length | 26 |

| CPRB Field | Content/Description |
|---|---|
| Replied DATA length | 0 to 1024<br><br>(Nine times the number of expected generations or eight times the number of expected logical journals) |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 8 | Logical journal name or blank |

| Reply DATA Values with Logical Journal Name supplied | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Name of first physical electronic journal containing a generation of the logical electronic journal |
| 8 | 1 | Associated process control flag |
| 9 | 8 | Name of second physical electronic journal containing a generation of the logical electronic journal |
| 17 | 1 | Associated process control flag |
| And so on for all generations present | | |

| Reply DATA Values with no Logical Journal Name supplied | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Name of first logical electronic journal |
| 8 | 8 | Name of second logical electronic journal |
| And so on for all generations present | | |

## Inquire physical (IP function)

This function allows an application program to retrieve the control information pertinent to a physical electronic journal. This function is valid in both electronic journal environments. Two options are provided:

1. If no physical electronic journal name is provided, the names and file status of all allocated physical electronic journals are returned.
2. If a physical journal name is supplied, the function returns:
   - File status and the allocation date and time
   - All logical electronic journal names present and the corresponding process control flags

In the physical electronic journal environment, only the status and the allocation date and time are returned.

| CPRB Field | Content/Description |
|---|---|
| Function code | IP |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 4096 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 33 to 4096<br><br>(Nine times the number of logical journals plus 32, or nine times the number of physical journals) |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 8 | Physical journal name or blank |

| Reply DATA Values with Physical Journal Name supplied | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Name of physical electronic journal |
| 8 | 1 | Corresponding file status (see below) |
| 9 | 14 | Allocation date and time |
| 23 | 8 | Name of first logical electronic journal |
| 31 | 1 | Associated process control flag (see below) |
| 32 | 8 | Name of second logical electronic journal |
| 40 | 1 | Associated process control flag (see below) |
| And so on for all logical electronic journals present | | |

| Reply DATA Values with no Physical Journal Name supplied | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Name of first physical electronic journal |
| 8 | 1 | File status (see below) |
| 9 | 8 | Name of second physical electronic journal |
| 17 | 1 | File status (see below) |
| And so on for all physical electronic journals present | | |

| File Status | |
|---|---|
| **'A'** | Allocated |
| **'U'** | Updated |
| **'R'** | Released |

| Control Flag | |
|---|---|
| **'S'** | Selected |
| **'C'** | Current |
| **'A'** | Available |
| **'R'** | Released |

## Release electronic journal (RL function)

This function is used by the application program:

- In a physical environment, to globally release a specific physical electronic journal.
- In a logical environment, to release records associated with a logical electronic journal in the oldest accessible physical journal. A physical journal has the released status when all the logical journals in this data set are released.

Typically, this function is requested when the data is no longer needed, for example, when the data is archived.

| CPRB Field | Content/Description |
|---|---|
| Function code | RL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 8 | Physical journal name or logical journal name |

## Retrieve record (RR function)

This function retrieves an electronic journal record.

**Physical electronic journal environment**
The record is retrieved from the electronic journal identified by a physical electronic journal name.

**Logical electronic journal environment**
The record is retrieved from the current or previous generation of logical journal identified by a logical electronic journal name.

Sequential search operations are forward or backward. The server also provides a method to search for a specific journal record. A search definition is specified, see page 385 for details. The first, previous, or next record that meets the search criteria is returned.

| CPRB Field | Content/Description |
|---|---|
| Function code | RR |
| Request DATA length | 0 to 1024 (length of search arguments) |
| Request PARMLIST length | 26 |
| Reply DATA length | 1024 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 33 to 4096 (Length of record plus 32) |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | BT/ET flag |
| 2 | 1 | Search mode |
| 3 | 1 | Delete flag |
| 4 | 1 | Hold flag |
| 6 | 2 | Journal generation number (binary, byte-reversed) |
| 8 | 2 | Maximum access number (binary, byte-reversed) |
| 10 | 8 | Logical journal name or physical journal name |
| 18 | 8 | Record format name. Not needed unless search criteria used |

A detailed description of the contents follows:

- BT/ET flag. Specify one of:
  **'F'** First
  **'M'** Middle
  **'O'** Only
  **'L'** Last

  If hold flag is 'H', only the values 'F' and 'M' are used.
- Search mode:
  - Without search criteria, retrieve:
    **'F'** First record in the electronic journal
    **'N'** Next record, following the previously read
    **'P'** Previous record, preceding the record previously read
  - With search criteria, retrieve:
    **'F'** First record matching the search criteria
    **'N'** Next record matching the search criteria
    **'P'** Previous record matching the search criteria
  - If a search operation with search mode 'F', 'N', 'P' is prematurely ended by the server (because the maximum allowed number of physical disk accesses was reached), it must be resubmitted with search mode 'N' or 'P' respectively.

- Delete flag. Specify one of:
  **'Y'** Retrieve records with status = 'D', 'I' or 'U'. That is, also logically deleted records are retrieved
  **'N'** Retrieve records with status = 'I' or 'U'
- Hold flag. Specify one of:
  **'H'** If you want to queue and hold the records for a later update or delete (only one hold may be outstanding)
  **'N'** If you do not want to hold the record
- Journal generation number to select any logical electronic journal:
  **00** The current logical electronic journal is selected for retrieval
  *nn* The −*n*th generation is selected for retrieval
- Maximum access number. Used to limit search operation to a certain number of disk access. Valid values are from X'0000' to X'FFFF' To use the value specified in the electronic journal profile, set this field to X'2020'.

The search argument is stored in Reply DATA.

| **Reply DATA Values** | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| The record found that matches the search definition is returned preceded by the following fixed length header: | | |
| 0 | 2 | Length of record. This is the actual length of the record even if only a part of it could be stored in Reply DATA |
| 2 | 14 | System date and time (yyyymmddhhmmss) |
| 16 | 5 | Sequence number (for server internal use) |
| 21 | 2 | Originating workstation ID |
| 23 | 1 | Record status:<br>**'I'** As originally added<br>**'U'** Updated<br>**'D'** Logically deleted |
| 24 | 8 | Format name |
| 32 | nnn | Record |

## Reset electronic journal file (RS function)

This function resets all files that have a status of R (released). If no file can be reset the server returns a nonzero return code. A forced reset option is provided to unconditionally reset a specific file.

An RS function causes the server to issue a shared-file server ZD function, see page 295.

When using the AL function, see "Allocate physical electronic journal (AL function)" on page 392 to note the required use of the RS function.

| CPRB Field | Content/Description |
|---|---|
| Function code | RS |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 1024 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 to 1024<br><br>Number of files with status released (maximum of eight times the number of physical journals) |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 8 | Physical journal name. Specified only for a forced reset. If this field is filled with blanks, all released physical electronic journals are reset |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Name of first reset physical electronic journal |
| 8 | 8 | Name of second reset physical electronic journal |
| nn | 8 | Name of last reset physical electronic journal |

Reply DATA values are not returned for a forced reset.

## Select current electronic journal (SL function)

This function is only used in the logical electronic journal environment. It is used to select the current electronic journal for the next journal period. There are two options:

**'N'** (next)
: The server assigns the current logical electronic journal to the most recently allocated physical journal.

**'P'** (previous)
: The server reverts a previous selection with the 'N' option. This function can only be executed if no records have been added to the current journal and if there is a previous generation of the logical journal.

In both cases, the electronic journal access control table and the electronic journal control record are updated.

| CPRB Field | Content/Description |
|---|---|
| Function code | SL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 1 | 1 | 'N' or 'P' (next or previous) |
| 10 | 8 | Logical journal name |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 8 | Physical journal name now associated with the logical electronic journal |

## Test initialization (TI function)

This function checks whether the electronic journal is initialized. This function is usually requested until the server's return code is 0. The shared-file and the system manager servers must be operational before the electronic journal server can be made operational.

| CPRB Field | Content/Description |
|---|---|
| Function code | TI |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

## Terminate session (TS function)

This function terminates a separate session for the shared-file server access. It is used only if the separate session integrity option has been specified and an open session must be terminated explicitly.

| CPRB Field | Content/Description |
|---|---|
| Function code | TS |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | BT/ET flag (separate session option only)<br>**' '** ET should be requested, but not RB<br>**'R'** Issue RB followed by ET |

## Update record (UP function)

This function updates a record previously retrieved-for-update with an RR function with hold flag = 'H'. The RR and UP functions must be requested within the same transaction.

The record status of updated records is set to 'U'. No other fields in the header are updated.

The key field data values, the record format definition, and the length of the record *may not* be modified.

| CPRB Field | Content/Description |
|---|---|
| Function code | UP |
| Request DATA length | 0 to 4064 (length of record stored in Request DATA) |
| Request PARMLIST length | 26 |
| Reply DATA length | 32 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 32 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | BT/ET flag<br>**'M'** Not the last record (middle)<br>**'L'** Last record |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 8 | Logical journal name or blank (if physical electronic journal) |
| 18 | 8 | Record format names |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 to 4064 | Record to replace the existing record |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Length of record |
| 2 | 14 | System date and time (yyyymmddhhmmss) |
| 16 | 5 | Sequence number (for server internal use) |
| 21 | 2 | Originating workstation ID |
| 23 | 1 | Record status, changed to 'U' |
| 24 | 8 | Format name |

# Chapter 18. Store-for-forwarding server

The store-for-forwarding server helps the application programmer to handle situations when the LANDP workgroup, temporarily or periodically, has no host connection.

- The store-for-forwarding server stores records locally, on a hard disk, in the LANDP workgroup.
- A forwarding server transmits the records to the host. LANDP provides a forwarding server, or you can develop your own.

*Table 39. Function Codes used in the Store-for-Forwarding Server*

| Function code | Description | Env. | Page |
|---|---|---|---|
| AR | Add record | 02NL | 409 |
| CS | Set communication status | 02NL | 410 |
| DL | Delete record | 02NL | 411 |
| DS | Disable storing | 02NL | 412 |
| EN | Enable storing | 02NL | 412 |
| IS | Inquire status | 02NL | 413 |
| QD | Query file | 02NL | 414 |
| RR | Retrieve record | 02NL | 414 |
| RS | Reset store-for-forwarding file | 02NL | 417 |
| SI | Initiate store | 02NL | 417 |
| TB | Begin transmission | 02NL | 418 |
| TE | Stop transmission | 02NL | 419 |
| TI | Test initialization | 02NL | 420 |
| TS | Terminate session | 02NL | 420 |
| UP | Update record | 02NL | 420 |

The first and second columns give the function code and the name of the function. The third column shows the operating environment of the function, as explained in “Operating environments” on page xxvi. “02NL” means that it is available from LANDP for DOS, LANDP for OS/2, LANDP for Windows, or LANDP for Linux servers. The last column refers to the page where you can find the function described.

Once the store-for-forwarding server is loaded, the application must wait until this server is initialized and operational. The application must issue a test initialization (TI) function call to verify that the server is initialized and operational.

## Storing records

Functions are available to:

**AR** Store (add) records
**DS** Disable storing
**EN** Enable storing

© Copyright IBM Corp. 1992, 2002

The maximum record length is 4064 bytes. Stored records are numbered by the store-for-forwarding server beginning with 1 for the first record in a file. A record status is stored in the header part of each record. The status of a stored record can be:

**'I'** Unchanged, as initially created
**'U'** Updated
**'P'** Retrieved for forwarding by the LANDP forwarding server
**'R'** Retrieved for deletion by an application

The records are physically deleted when they have been forwarded to the host or made available to an application. This space is automatically reused by the shared-file or the query server.

## Processing stored records

For processing stored records, functions are provided to:

**RR** Retrieve
**UP** Update
**DL** Delete

The RR and DL functions are also used by the LANDP forwarding server to retrieve records for forwarding and later deletion.

An application can retrieve any record, no matter what its status is. Records retrieved for deletion by an application, record status 'R', are not processed by the LANDP forwarding server.

## Data integrity of store-for-forwarding files

Data integrity of the store-for-forwarding files is, to a large extent, provided by the shared-file server, SHFILE## (see Chapter 14, "Shared-file server" on page 255). The shared-file server contains data integrity functions:

**BT** Begin transaction
**ET** End transaction

Between BT and ET, a sequence of record operations is performed. The accessed records are locked (held) until ET (or RB, rollback) is issued. When ET is issued, the records used since the last BT can no longer be changed. The BT–ET sequence is the normal error free case.

When technical or operational problems occur, you can at any time use:

**RB** Rollback

When this function is used all record manipulations done since the last BT are canceled.

There are two data integrity options for store-for-forwarding files that you can specify during customization. They are defined in keyword SEPSESS in the SFORWPRF vector.

1. SEPSESS=Y.

   If the Y (Yes) option is chosen, the store-for-forwarding server obtains a separate shared-file server session for the data accesses. The store-for-forwarding server issues the data integrity functions BT and ET, and if necessary RB.

   Opening and closing of a separate shared-file session is controlled by a flag in the relevant store-for-forwarding functions:

   **'F'** Session to begin
   **'M'** Session to continue
   **'L'** Session to end
   **'O'** Session to begin and to end

   You can explicitly terminate a separate session using the TS function.

   Example of use of the separate session option to add four records:

   ```
   AR  with option 'F'        (first record)
   AR  with option 'M'        (second record)
   AR  with option 'M'        (third record)
   AR  with option 'L'        (fourth record)
   ```

   The store-for-forwarding server issues a BT when encountering a record with the option F and an ET when the option is L.

   You can explicitly terminate a session using the terminate session (TS) function.

2. SEPSESS=N. This is the default.

   If the N (No) option is chosen, the application must issue the BT and ET shared-file server functions. The store-for-forwarding server does not issue the BT or ET function. If an application does not issue the proper shared-file server function calls, data integrity is exposed.

   Example of use of the single session option:

   ```
   BT
   RR (for deleting)
   DL
   ET
   ```

*Automatic Rollback:* The following return codes in the specified function cause an RB (rollback) function to be requested automatically (if a separate session option is selected):

| Function | Return code |
|---|---|
| **AR — Add record** | P1, P2, PH, PI, S2, S6 |
| **RR — Retrieve record** | P1, S2, S7 |
| **DL — Delete record** | P1, S0, S2, S9 |
| **UP — Update record** | P1, PH, PI, PL |

After the DL and UP functions the lock (hold) is released and another RR function must be requested to use another record.

## Obtaining database error messages

The following functions return a database error message when a database error occurs: AR, DL, DR, DX, RR, RS, and UP. By checking the length of *Replied DATA*, you can determine if a database error has occurred.

If the *Replied DATA length* is greater than zero, a database error *has* occurred. Replied DATA then contains the database error message. You must specify a Replied DATA length that is large enough to avoid a truncation of this error message.

## Search operations

The search operations of the store-for-forwarding server are identical to those of the electronic journal server. See “Search operations” on page 385.

## Using the store-for-forwarding server

This section provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields. If you need more information about the CPRB fields, see Appendix A, “Connectivity programming request block” on page 669.

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | 26 (see note) | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 26 | 2 | 26 | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | SFORFORW | Server name |

The following fields are variable and are discussed in each function request description:
- Function code
- Request DATA length
- Reply DATA length

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 40 | 4 | | Server return code |
| 44 | 2 | | Replied PARMLIST length |
| 46 | 2 | | Replied DATA length |

If the request was successful, the *router return code* and the *server return code* are both X'00000000'. In all other cases, see the appropriate section in the *LANDP Problem Determination* book. to see if you should take any action. The return values in Reply PARMLIST and DATA should be ignored if there is an error.

The following fields are variable and are discussed in each function request description:
- Replied PARMLIST length
- Replied DATA length

**PARMLIST:** The Request PARMLIST and Reply PARMLIST for the store-for-forwarding server is explained in the function descriptions. Flags not used must be filled with blanks (ASCII X'20').

**DATA:** Request DATA and Reply DATA contain data to be sent to or received from the server. The use of these areas is explained in the description of each function request.

## Request reference

These are the functions that can be used with the store-for-forwarding server. They are listed in alphabetical order of function code.

## Add record (AR function)

This function is used to add a record to a store-for-forwarding file and mark it with record status 'I'. The store-for-forwarding server stores the record for later:
- Forwarding to the host application program by the forwarding server
- Retrieval by an application program

| CPRB Field | Content/Description |
|---|---|
| Function code | AR |
| Request DATA length | Length of record to be added: 0 to 4064 |
| Request PARMLIST length | 26 |
| Reply DATA length | 32 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 32 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | BT/ET flag (separate session integrity option)<br>**'F'** First<br>**'M'** Middle<br>**'L'** Last<br>**'O'** Only |
| 10 | 8 | Store-for-forwarding file name |
| 18 | 8 | Record format definition name |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | See above | Record to be added<br><br>If data is missing in a field defined as a key field, the corresponding data in the field is set to low values according to the field format specified. |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| Information from the header is stored with the record. | | |
| 0 | 2 | Length of record (as supplied on request) |
| 2 | 14 | System date and time (yyyymmddhhmmss) |
| 16 | 5 | Sequence number |
| 21 | 2 | ID of originating workstation |
| 23 | 1 | Record status, set to 'I' |
| 24 | 8 | Format name (as supplied at request) |

The sequence number that is used to identify a record within a file is increased by one for each record. It is used in a wrap around mode. It is reset to 1 when the store-for-forwarding file is reset. The sequence number does not relate to the actual physical position in the file.

## Set communication status (CS function)

This function is used by a forwarding server to inform the store-for-forwarding server of a change of the communication status of any of the communication links. This function is useful if you develop your own forwarding server.

| CPRB Field | Content/Description |
|---|---|
| Function code | CS |
| Request DATA length | 27 |
| Request PARMLIST length | 26 |

| CPRB Field | Content/Description |
|---|---|
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Status of SESSION1 |
| 1 | 8 | File name if status = 'R' |
| 9 | 1 | Status of SESSION2 |
| 10 | 8 | File name if status = 'R' |
| 18 | 1 | Status of SESSION3 |
| 19 | 8 | File name if status = 'R' |

### Status of sessions:

**'U'** Session up
**'D'** Session down
**'R'** Rejected, negative response
**' '** Session not defined

## Delete record (DL function)

This function is used to physically delete a record previously retrieved using the function RR with the option *retrieve-for-deleting*.

| CPRB Field | Content/Description |
|---|---|
| Function code | DL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 32 |
| Reply PARMLIST length | 26 or 30 |
| Replied DATA length | 32 |
| Replied PARMLIST length | 26 or 30 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | BT/ET flag (separate session integrity option)<br>**'O'** Only<br>**'F'** First<br>**'M'** Middle<br>**'L'** Last |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 3 | 1 | Request qualifier<br>**'F'** Request from forwarding server<br>**other** Request from application program (Your own forwarding server) |
| 10 | 8 | Store-for-forwarding file name |
| 18 | 8 | Record number. Returned when the record was retrieved for deleting. It is used to locate the record to be physically deleted. |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Length of record (binary) |
| 2 | 14 | Date and time |
| 16 | 5 | Sequence number |
| 21 | 2 | Originating workstation ID |
| 23 | 1 | Record status |
| 24 | 8 | Format name |

## Disable storing (DS function)

This function is used to disable a store-for-forwarding file for the AR (add record) function.

| CPRB Field | Content/Description |
|---|---|
| Function code | DS |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 8 | Store-for-forwarding file name |

## Enable storing (EN function)

This function is used to enable a store-for-forwarding file for the AR (add record) function.

| CPRB Field | Content/Description |
|---|---|
| Function code | EN |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 8 | Store-for-forwarding file name |

## Inquire status (IS function)

This function is used to obtain the status of a store-for-forwarding file. The information returned to the application program is:

- Flag indicating whether records are present in the file or not
- State of store-for-forwarding file (storing of records enabled or disabled)
- Transmission status (transmission active or stopped)
- Session status

| CPRB Field | Content/Description |
|---|---|
| Function code | IS |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 4 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 4 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 8 | Store-for-forwarding file name |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Records in file ('Y' or 'N') |
| 1 | 1 | Enabled for storing ('Y' or 'N') |
| 2 | 1 | Transmission enabled ('Y' or 'N') |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 3 | 1 | Session status ('U', 'D' or 'R'); see "Status of sessions:" on page 411 |

## Query file (QD function)

This function is used by the forwarding server to obtain the store-for-forwarding server file map indicating which files contain records to forward.

| CPRB Field | Content/Description |
|---|---|
| Function code | QD |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 18 | 8 | Store-for-forwarding file information map |

A bit map is returned containing one bit for each file. The bit is turned on if the file contains records to be forwarded and transmission has not been stopped with the TE function.

## Retrieve record (RR function)

This function is used to retrieve a record from a store-for-forwarding file. One record is returned at a time. Any record can be accessed, that is, not only those that were created by the user ID requesting the function.

Sequential search operations are forward or backward. The server also provides a method to search for a specific record. A search definition is specified. The first, previous, or next record that meets the search criteria is returned.

The LANDP application program can retrieve all records with record status 'I', 'U', 'P', and 'R' in three different ways:

- Retrieve-only
- Retrieve-for-deleting
- Retrieve-for-updating

The forwarding server can work with records with record status 'I', 'U', and 'P'.

Record status 'P' means that the record was sent to the host application program by the LANDP forwarding server, but no positive host acknowledgement has been received. The retrieve-for-deleting function also returns records marked 'P' if there is a permanent host link failure, to allow an application program to process the stored records.

A record that has been retrieved for deleting by an application program is marked with a record status 'R', indicating that the application program assumes responsibility for proper processing and deletion. The LANDP forwarding server does *not* send records with record status 'R'.

| CPRB Field | Content/Description |
|---|---|
| Function code | RR |
| Request DATA length | 0 to 1024 (search definition length) |
| Request PARMLIST length | 26 |
| Reply DATA length | 4096 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 33 to 4096 (Length of record actually read plus 32) |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | BT/ET flag (Separate session integrity option) |
| 1 | 1 | File information map option |
| 2 | 1 | Search mode |
| 3 | 1 | Request qualifier |
| 4 | 1 | Retrieve mode |
| 5 | 1 | Not used |
| 6 | 2 | Reserved |
| 8 | 2 | Maximum access number (binary, byte-reversed) |
| 10 | 8 | Store-for-forwarding file name |
| 18 | 8 | Record format name. Required only if search criteria used |

- BT/ET flag:
  **'F'** First
  **'M'** Middle
  **'O'** Only
  **'L'** Last

  *Retrieve-only* and *retrieve-for-deleting* can use any of these options, *retrieve-for-updating* can only use 'F' and 'M'.
- File information map option
  **'N'** Does not return the file information map

**other** Returns file information map

- Search mode

  Without search criteria, retrieve:
  **'F'** First user record in the file
  **'N'** Next record, (following the previously read)
  **'P'** Previous record (relative to the record previously read)

  With search criteria, retrieve:
  **'F'** First record matching the search criteria
  **'N'** Next record matching the search criteria
  **'P'** Previous record matching the search criteria

  If a search operation with search mode 'F', 'N', or 'P' is prematurely ended by the server (because the maximum allowed number of physical disk accesses was reached), it must be resubmitted with search mode 'N', 'N', or 'P' respectively.

- Request qualifier:
  **'F'** Request from forwarding server
  **other** Request from application program (your own forwarding server)

  Used to mark the *retrieved-for-deleting* record either as 'P' (potential duplicate) when this flag is 'F', or as 'R' (handled by the application program) when this flag is not 'F'.

- Retrieve mode:
  **'R'** Retrieve only
  **'D'** Retrieve for deleting
  **'U'** Retrieve for updating

  The record number obtained with the *retrieve-for-deleting* option must be passed to the store-for-forwarding server with the delete function to physically delete the record.

  Retrieve-for-deleting marks the record to be available for deleting (in the same or later transaction). Retrieve-for-updating holds the record until end of transaction.

- Maximum access number. Used to limit search operation to a certain number of disk access. Valid values are from X'0x0000' to X'0xFFFF' To use the value specified in the store-for-forwarding server profile, set this field to X'0x2020'.

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 to 1024 | Search definition |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 8 | Record number (Used in a following DL function request) |
| 18 | 8 | File information map (Only returned if file information map option is not equal to 'N') |

### File information map:

A bit map is returned containing one bit for each file. The bit is turned on if the file contains records to be forwarded and transmission has not been stopped with the TE function.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Length of record (This is the actual length of the record even if only a part of it could be stored in Reply DATA) |
| 2 | 14 | System date and time (yyyymmddhhmmss) |
| 16 | 5 | Sequence number |
| 21 | 2 | Originating workstation ID |
| 23 | 1 | Record status |
| 24 | 8 | Format name |
| 32 | nn | Record |
| The record matching the search definition is returned preceded by a fixed length header. | | |

## Reset store-for-forwarding file (RS function)

This function is used to reset an entire store-for-forwarding file *unconditionally.* It is provided to allow recovery from permanent error situations and should normally not be used.

| CPRB Field | Content/Description |
|---|---|
| Function code | RS |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 8 | Store-for-forwarding file name |

## Initiate store (SI function)

This function is used by the forwarding server to inform the store-for-forwarding server about the relationship between communication sessions and store-for-forwarding files, and if automatic transmission is active. The name of the shared-file server is returned.

| CPRB Field | Content/Description |
|---|---|
| Function code | SI |
| Request DATA length | 2 to 128<br>Twice the number of files (exactly) |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | 1, 2, or 3. Forwarding session ID associated with first file |
| 1 | 1 | Automatic transmission selected for first file: 'Y' or 'N' |
| 2 | 1 | 1, 2, or 3. Forwarding session ID associated with second file |
| 3 | 1 | Automatic transmission selected for second file: 'Y' or 'N' |
| And so on for all defined store-for-forwarding files. | | |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 8 | Shared-file server name |

## Begin transmission (TB function)

This function starts the transmission of records. The forwarding server processes the records with a record status of 'I', 'U', and 'P'. Records with status 'R' (retrieved by application program) are not transmitted.

This function is required if automatic transmission is not selected during customization for the store-for-forwarding file or if transmission has been explicitly stopped with the TE function.

Transmission can be initiated for a specific file or for all files associated with the session.

| CPRB Field | Content/Description |
|---|---|
| Function code | TB |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |

| CPRB Field | Content/Description |
|---|---|
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 5 | 1 | File or session name at offset 10:<br>**'D'** File name supplied Valid names in range: ″FBSF01 ″ to ″FBSF64 ″<br>**'S'** Session name supplied<br>Valid names in range: ″SESSION1″ to ″SESSION3″ |
| 10 | 8 | Store-for-forwarding file name or forwarding session name |

## Stop transmission (TE function)

This function is used to stop transmission of records. Any transmission, either automatically or explicitly started can be stopped by the application program using this function. If stopped with this function, it has to be explicitly started by the application program with function TB, even if automatic transmission is specified during customization.

One specific file or all files associated with a session name are inhibited for transmission.

| CPRB Field | Content/Description |
|---|---|
| Function code | TE |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 5 | 1 | File or session name at offset 10:<br>**'D'** File name supplied<br>Valid names in range: ″FBSF01 ″ to ″FBSF64 ″<br>**'S'** Session name supplied<br>Valid names in range: ″SESSION1″ to ″SESSION3″ |
| 10 | 8 | Store-for-forwarding file name or forwarding session name |

## Test initialization (TI function)

This function checks whether the store-for-forwarding server is initialized. This is usually requested until the server's return code is 0. The shared-file and system manager servers must be operational before the store-for-forwarding server can be made operational.

| CPRB Field | Content/Description |
|---|---|
| Function code | TI |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

## Terminate session (TS function)

This function terminates a separate session for the shared-file server access. It is used only when the separate session integrity option has been specified and an open session must be terminated explicitly.

| CPRB Field | Content/Description |
|---|---|
| Function code | TS |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | BT/ET flag (separate session option only)<br>**' '** ET should be requested, but not RB<br>**'R'** Issue RB followed by ET |

## Update record (UP function)

This function is used to update a record previously retrieved by using function RR with the option *retrieve-for-updating*.

| CPRB Field | Content/Description |
|---|---|
| Function code | UP |
| Request DATA length | Length of record: 1 to 4064 |

| CPRB Field | Content/Description |
|---|---|
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | BT/ET flag<br>**'M'** Middle (not the last) record<br>**'L'** Last record |
| 18 | 8 | Format name |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | See above | Record to replace the existing record (length as specified above) |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Length of record (binary) |
| 2 | 14 | Date and time |
| 16 | 5 | Sequence number |
| 21 | 2 | Originating workstation ID |
| 23 | 1 | Record status, changed to 'U' |
| 24 | 8 | Format name |

## Example of handling off-line situations

If the host application program is temporarily not available, records are stored for later transmission. Later processing (forwarding) can be made using:

- LANDP forwarding server.
- Your own application program. The application program has to ensure correct forwarding (or any other relevant processing) of the stored records and their deletion from the file.

## Forwarding using the LANDP forwarding server

Often, this is the intended use of the store-for-forwarding server. Storing and forwarding are separate and independent processes.

### Example of using store-for-forwarding in off-line situations

```
BUILD record
SEND record to host application program
CHECK status of SEND record
if SEND was correct
   then Record is sent to the host
   else Add record (AR with option 'O', only record)
```

It is assumed that a separate shared-file session data integrity option has been specified during customization. If more than one record has to be added during one transaction, use the options 'F', 'M', and 'L'.

Since the forwarding server operates completely independently of the store-for-forwarding server and the application program, the application program can monitor the forwarding process. This is typically done at the end of the day, and probably only when the teller totals do not agree.

### Example of monitoring forwarding

```
INQUIRE forwarding status (IS function)
if no records in the file
   then DISPLAY "No records left to be forwarded"
   else { DISPLAY "Not all records transmitted"
                  "Session status" (if records rejected)
                  "Transmission status"
          act appropriately
        }
```

## Example of local data collection for later forwarding

Records need not be sent to the host application program when they are created. They can be collected and sent at a later time.

A local data capture application program can, after validation of the data, transfer the records into one of the store-for-forwarding files for later transmission to the host.

Start and stop transmissions (TB and TE) are used as needed to control transmission.

### Example of initiating local record storing

```
INQUIRE (IS) forwarding status
if all records forwarded
  then { STOP transmission (TE)
         ENABLE storing (EN)
       }
  else DISPLAY "Forwarding in progress, try later"
```

The program called at the end of local data collection could be structured as follows:

### Example of initiating locally collected data forwarding

```
DISABLE storing (DS)
START forwarding (TB)
```

## Your own application program for forwarding

Stored records can be forwarded by your application program. Before forwarding can be started, the environment required for forwarding has to be established. Information about session and file relationship and automatic transmission initialization, has to be supplied by using the SI function.

In the following example all stored records of a computer in the LANDP workgroup are transmitted before on-line transmission is resumed.

An application program used for forwarding must use the separate session option when calling the store-for-forwarding server to retrieve and delete records.

### Example of an application program for forwarding

```
Inquire status of host session
if session available
   then { Initiate store (SI)
          if records in file (QD)
             then do { RETRIEVE (RR) record for delete,
                                  and save record number
                       SEND record to host application program
                       if send was correct
                          then DELETE (DL) using saved record number
                          else if session down
                                  then LEAVE DO, and repeat processing
                                  else act appropriately.
                                       (The record remains in the
                                       file with status 'R')
                     } until all records of computer in the
                             LANDP workgroup retrieved
             else { BUILD record
                    SEND record to host application program
                  }
        }
   else AR STORE record for forwarding
```

If the store-for-forwarding file is linked to a forwarding session during customization, it should be defined without automatic forwarding, or the forwarding server must be explicitly stopped using the TE function.

The function QD returns a 64-bit map indicating which file has records to be forwarded.

# Part 5. Application integration servers

This part of the book describes the servers provided by the LANDP family to allow access to non-LANDP environments and allows resource and data sharing with non-LANDP programs. It has the following chapters:

**Chapter 19, “CICS interface server” on page 427**

The CICS interface server allows LANDP applications to access CICS OS/2™ Version 1.2 (or later) facilities and data located in the same or another workstation in the same LANDP workgroup.

The application that requests services from the server can run in any workstation within the LANDP workgroup. You can install more than one server in one LANDP workgroup. Therefore, an application can request services from more than one server concurrently.

This chapter presents the functions provided by this server for servicing requests originating in application programs or other servers.

**Chapter 20, “Dynamic data exchange access server” on page 435**

The DDE access server provides an interface between the LANDP environment and some OS/2 methods for exchanging data among applications. You can also use this server to use the clipboard.

This server allows you to access these services from remote workstations.

This chapter presents the functions provided by this server for servicing requests originating in application programs or other servers. The chapter also contains programming examples showing how the server can be used.

© Copyright IBM Corp. 1992, 2002

# Chapter 19. CICS interface server

The CICS interface server enables a LANDP client (or server acting as client) to access a CICS system that is running under OS/2 or AS/400.

The LANDP client/server mechanism allows more than one CICS interface server to run in the same LANDP logical LAN. A client can therefore request functions from more than one CICS interface server at the same time.

The client that requests the use of the CICS interface server can be installed in any LANDP for DOS or LANDP for OS/2 workstation. The CICS interface server must however be installed in the same LANDP for OS/2 workstation as the CICS system that provides the CICS ECI that it uses.

Under LANDP for DOS, the client can be a Windows 3.1 or a non-Windows application. Under LANDP for OS/2, the client can be an OS/2 windowed or non-windowed application.

*Table 40. Function Codes used in the CICS interface server*

| Function code | Description | Env. | Page |
|---|---|---|---|
| CC | Call CICS | -2- | 429 |
| RC | Read CICS | -2- | 433 |

The first and second columns give the function code and the name of the function. The third column shows the operating environment of the function. "-2-" means that it is available from LANDP for OS/2 servers only. The last column refers to the page where you can find the function described.

## Running with CICS

Each time a CC (call CICS) function is requested, a CICS program is invoked. The *logical units of work (LUW)* must be controlled by your user program.

Since the CICS interface server provides the transaction support when using CICS, all restrictions and dependencies that apply to CICS also apply to the CICS interface server.

## CICS external call interface

The CICS interface server uses the CICS *External Call Interface (ECI)*. Many of the fields used with the server have identical syntax, meaning and rules as those for the CICS ECI. As a result these fields are passed to CICS, and received from CICS, *transparently* in the Request PARMLIST and Reply PARMLIST areas. The CICS interface server does not modify these fields in any way. Check your CICS documentation for information about the format and rules for using these fields.

© Copyright IBM Corp. 1992, 2002

The Request DATA and Reply DATA areas contain the data that is passed to or received from the CICS system through the CICS ECI. You should ensure that your clients contain Request DATA and Reply DATA areas that are large enough to send or receive the maximum expected message size. Otherwise, the message is truncated. The fields correspond to those defined in the ECI-COMMAREA for the CICS ECI.

## Synchronous and asynchronous server use

A CICS transaction is requested using the CC function, in which the operational mode is specified in Request PARMLIST: parallel synchronous (SYNC) or parallel asynchronous (ASYNC).

- If parallel *synchronous* mode is specified, the calling thread or process waits until transaction execution is completed, and the reply from the CICS interface server has been received. This reply (contained in the Reply PARMLIST and Reply DATA areas) is then returned to the requesting thread or process for the CC function.
- If parallel *asynchronous* mode is specified, the calling process can continue its processing without the need to wait for a reply from the CICS system.

  When a reply is received, the CICS interface server posts a LANDP message. This message can be received either using the supervisor local function WM (see “Wait for asynchronous events (WM function)” on page 24 for details), or in the OS/2 or Windows event queue using the SP function (see “Start posting events (SP function)” on page 20 for details).

  When the client has been notified by the server that the event has been processed, the client then requests the RC function in order to obtain the reply from the CICS interface server (contained in the Reply PARMLIST and Reply DATA areas).

## Using the CICS interface server

This section provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields. If you need more information about the CPRB fields, see Appendix A, “Connectivity programming request block” on page 669.

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | 64 | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 26 | 2 | 64 | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | EHCTRAN | Server name |

The following fields are variable and are discussed in each function request description:
- Function code
- Request DATA length
- Reply DATA length

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 44 | 2 | 64 | Replied PARMLIST length |
| 46 | 2 | | Replied DATA length |

If the request was successful, the *router return code* and the *server return code* are both X'00000000'. In all other cases, see the appropriate section in the *LANDP Problem Determination* book to see if you should take any action. The values returned in Reply PARMLIST and Reply DATA should be ignored when an error occurs.

You can find the abend code that CICS returns in the AbendCode field of Reply PARMLIST. You should check in the appropriate CICS literature for the meaning of this code.

The following fields are variable and are discussed in each function request description:
- Replied PARMLIST length
- Replied DATA length

## Request reference

This section describes the functions that clients can request, from the CICS interface server. They are listed in alphabetical order of function code.

### Call CICS (CC function)

This function is requested by the client to pass a request and data to CICS, for CICS to start a CICS program.

| CPRB Field | Content/Description |
|---|---|
| Function code | CC |
| Request DATA length | ≤ 32767 |
| Request PARMLIST length | 64 |
| Reply DATA length | ≤ 32767 |
| Reply PARMLIST length | 64 |
| Replied DATA length | ≤ 32767 |
| Replied PARMLIST length | 64 |

**Note:** In the following table, the term *transparently* is used. For an explanation, see "CICS external call interface" on page 427.

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Parallel SYNC or ASYNC indicator (see "Synchronous and asynchronous server use" on page 428)<br><br>**ECI_SYNC_PARALLEL**<br>X'0200', *parallel synchronous processing*<br><br>**ECI_ASYNC_PARALLEL**<br>X'0300', *parallel asynchronous processing* |
| 2 | 8 | Program name: passed transparently to CICS (this field must be padded with blanks) |
| 10 | 8 | Userid: passed transparently to CICS (this field must be padded with blanks)<br><br>Used by CICS security |
| 18 | 8 | Password: passed transparently to CICS (this field must be padded with blanks)<br><br>Used by CICS security |
| 26 | 4 | Transid: passed transparently to CICS (this field must be padded with blanks)<br><br>Contains CICS transaction code |
| 34 | 6 | Not used. |
| 40 | 2 | Timeout: passed transparently to CICS<br><br>Specifies the maximum wait time in seconds within which CICS must respond to the CICS call interface server request. This value must be less than the LANDP timeout for the request |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 44 | 2 | Extend Mode: passed transparently to CICS<br><br>This field determines how the *logical unit of work* (see “Running with CICS” on page 427 for more information) is processed. The values that the ExtendMode can take are the following:<br>**X'0000'** ECI_NO_EXTEND<br><br>Terminates connection to CICS when the call is completed. This results in an automatic end-of-task syncpoint for CICS resources, and the logical unit of work is terminated.<br>**X'0100'** ECI_EXTEND<br><br>The connection to CICS is maintained, when the call is completed. Therefore, a logical unit of work can cover several different calls.<br>**X'0200'** ECI_COMMIT<br><br>The connection to a previous CICS is cancelled. This is the equivalent processing as when ECI_NO_EXTEND is used with a no-op program. The current logical unit of work is terminated.<br>**X'0200'** ECI_CANCEL<br><br>This has the same effect as ECI_COMMIT (above).<br>**X'0300'** ECI_BACKOUT<br><br>Forces a transaction to be rolled back to its original state. |
| 46 | 6 | Not used |
| 52 | 2 | EventCode<br><br>This event code is specified by the client in order to identify the client to which a reply should be sent. It is used with the WM and QE supervisor local functions.<br><br>The event code should be set to any nonzero value in order to be used. When set to zero, it is not possible to identify specific replies that are pending to be read. Instead, RC function returns replies in the order in which they become available. With this type of processing, the client specifies an event code ## with the WM request. |
| 54 | 4 | LuwToken: passed transparently to CICS<br><br>This parameter allows you to:<br>• Distinguish between requests that are made in parallel<br>• Allocate a request made in parallel with other requests, to a specific LUW.<br><br>Your client should initialize this field to zero. CICS then updates this field with the value of a valid luwtoken on the first (or only) call of a LUW. |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 58 | 6 | Reserved (must be initialized to zero) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Data to be passed to CICS program |

**Note:** In the following table, it is stated that a field is returned *transparently* by CICS. For further explanation, refer to “CICS external call interface” on page 427.

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 30 | 4 | AbendCode: returned transparently by CICS to your client program |
| 42 | 2 | SysReturnCode: returned transparently by CICS<br><br>The CICS interface server writes an entry to the LANDP Log Error File. You can read this file by using the EHCPDT program.<br><br>Also, the CICS interface server returns a U9 return code. See the appropriate section in the *LANDP Problem Determination* book for an explanation. |
| 54 | 4 | LuwToken: returned transparently by CICS<br><br>CICS updates the luwtoken field with the value of a valid luwtoken on the first (or only) call of a LUW. |
| **Note:** This reply is returned only when using CC function in synchronous mode, see “Synchronous and asynchronous server use” on page 428 for further information. | | |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Data returned by the CICS program |
| **Note:** This reply is returned only when using CC function in synchronous mode, see “Synchronous and asynchronous server use” on page 428 for further information. | | |

## Read CICS (RC function)

This function is requested by the client in order to receive the reply sent by CICS to an original request that was sent using the CC function in *asynchronous* mode. See “Synchronous and asynchronous server use” on page 428 for further information.

| CPRB Field | Content/Description |
|---|---|
| Function code | RC |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | ≤ 32767 |
| Reply PARMLIST length | 64 |
| Replied DATA length | ≤ 32767 |
| Replied PARMLIST length | 64 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | EventCode<br><br>This event code is specified by the client in order to identify the client to which a reply should be sent. It is used with the WM and QE supervisor local functions.<br><br>The event code should be set to any nonzero value in order to be used. When set to zero, it is not possible to identify specific replies that are pending to be read. Instead, the RC function returns replies in the order in which they become available. With this type of processing, the client specifies an event code ## with the WM request. |

**Note:** In the following table, it may be stated that a field is returned *transparently* by CICS. For further explanation, see “CICS external call interface” on page 427.

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 30 | 4 | AbendCode: returned transparently by CICS |
| 42 | 2 | SysReturnCode: returned transparently by CICS<br><br>The CICS interface server also writes an entry to the LANDP Log Error File. You can read this field by using the EHCPDT program. |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 54 | 4 | LuwToken: returned transparently by CICS<br><br>This parameter allows you to:<br>• Distinguish between requests that are made in parallel<br>• Allocate a request made in parallel with other requests, to a specific LUW.<br><br>CICS updates the luwtoken field with the value of a valid token. |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Data returned by the CICS program. This CICS program was started by a previous CC function request in *asynchronous* mode |

# Chapter 20. Dynamic data exchange access server

The DDE access server allows LANDP clients (or servers acting as clients) to communicate with applications that use the dynamic data exchange (DDE) conventions. The server also allows clipboard and DDE services to be accessed remotely. Their location is transparent to the client. However, the DDE access server must be installed in the same **OS/2** workstation as the OS/2 application providing the DDE services.

The DDE access server “acts as a client” when it obtains a DDE service from an OS/2 application. The LANDP client first requests a function from the DDE access server, and the DDE access server then requests a DDE service from an OS/2 application.

The DDE access server is requested by the LANDP client using the LANDP common application programming interface (API). The LANDP client can be an OS/2 windowed or non-windowed program. Non-windowed client programs can run under DOS or OS/2.

The DDE access server provides two sets of functions: high-level and low-level. The high-level functions enable LANDP applications to access DDE operations with a single request. The low-level functions closely maps the DDE operations which provides more flexibility but requires more complex coding.

The DDE access server controls the use of DDE internal resources, and manages the DDE conversation and required OS/2 resources. The LANDP client does not need to be concerned with terms such as windows, messages, window manipulators, atoms, and so on.

The DDE access server also allows LANDP applications to copy data to, or obtain data from, the system clipboard.

*Table 41. Function Codes used in the DDE Access Server*

| Function code | Description | Env. | Page |
|---|---|---|---|
| GD | Get data | -2- | 437 |
| ID | Insert data | -2- | 438 |
| LO | Lock topic | -2- | 439 |
| LP | Load program | -2- | 439 |
| QA | Query applications | -2- | 440 |
| RC | Run command | -2- | 440 |
| UL | Unlock topic | -2- | 441 |
| **Low-level interface functions:** | | | |
| AD | Advise | -2- | 443 |
| EC | Execute command | -2- | 444 |
| IC | Initiate conversation | -2- | 445 |
| KD | Peek data | -2- | 446 |
| PD | Poke data | -2- | 446 |
| RD | Request data | -2- | 447 |
| TC | Terminate conversation | -2- | 448 |

© Copyright IBM Corp. 1992, 2002

*Table 41. Function Codes used in the DDE Access Server (continued)*

| Function code | Description | Env. | Page |
|---|---|---|---|
| UN | Unadvise | -2- | 448 |
| **Clipboard interface functions:** | | | |
| CT | Copy text | -2- | 449 |
| PT | Paste text | -2- | 450 |

The first and second columns give the function code and the name of the function. The third column shows the operating environment of the function. "-2-" means that it is available from LANDP for OS/2 servers only. The third column shows the operating environment of the function, as explained in "Operating environments" on page xxvi. The last column refers to the page where you can find the function described.

## Using the DDE access server

This section provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields. If you need more information about the CPRB fields, see Appendix A, "Connectivity programming request block" on page 669.

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | EHCLAD | Server name |

The following fields are variable and are discussed in each function request description:

- Function code
- Request PARMLIST length
- Request DATA length
- Reply DATA length

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 46 | 2 | | Replied DATA length |

If the request is successful, the *router return code* and the *server return code* are both X'00000000'. In all other cases, see the appropriate section in the *LANDP Problem Determination* book to see if you should take any action. The values returned in Reply PARMLIST and REPLY DATA should be ignored when an error occurs, *unless otherwise explicitly stated*.

The following field is variable and is discussed in the function request description:
- Replied DATA length

## DDE access server high-level functions

The DDE access server provides a set of *high-level functions* that enable LANDP client programs to perform basic DDE operations with only a single request to the DDE access server. The high-level functions provided by the DDE access server are:

- Get Data (GD) from a DDE server application.
- Insert Data (ID) in a DDE server application.
- Run Command (RC) on a DDE server application.
- Query Applications (QA) and topics currently supported.
- Load Program (LP)
- Lock Topic (LO) of a DDE server application and thereby prevent other LANDP client programs from using this topic.
- Unlock Topic (UL) of a DDE server application and thereby allow other LANDP client programs to use this topic.

## Request reference for high-level functions

This section describes the functions that clients can request for high-level functions, from the DDE access server. They are listed in alphabetical order of function code.

### Get data (GD function)

This function is requested by the client to obtain information about an application, topic, or item.

Because GD is a high-level function, it is not necessary to specify the handle of the conversation. The conversation must not have been previously opened by requesting an IC function.

The data received is a null-terminated string. If the item is a range of values, the values can be separated by tabs, line-feeds, and carriage-returns.

| CPRB Field | Content/Description |
|---|---|
| Function code | GD |
| Request DATA length | 0 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Application, Topic, Item (must be separated by commas) |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Data received from the DDE server application (null-terminated) |

## Insert data (ID function)

This function is requested by the client to insert *data* into an application, topic, or item.

Because ID is a high-level function, it is not necessary to specify the handle of the conversation. The conversation must not have been previously opened by requesting an IC function.

The data to be inserted must match the appropriate format required by the DDE server application.

| CPRB Field | Content/Description |
|---|---|
| Function code | ID |
| Request DATA length | 0 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Application, Topic, Item, Data (must be separated by commas) |

## Lock topic (LO function)

This function obtains exclusive access to a topic of a DDE server application. No other LANDP application can then have access to the locked topic until the LANDP application finishes processing or an unlock (UL) is requested.

The locking takes place at LANDP application level. You should be aware that other applications can therefore use this topic by using DDE or other methods.

A topic must be locked before the application initiates a conversation that uses the topic.

| CPRB Field | Content/Description |
|---|---|
| Function code | LO |
| Request DATA length | 0 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Application,Topic (must be separated by a comma) |

## Load program (LP function)

This function is requested by the client to start a program remotely, by starting a session where the program will run.

If batch files are to be used, the program to be run must be CMD.EXE and the batch file parameters must be /K or /C and the batch file which is to be run.

The LP function requires that both parameters are specified. If the parameters have no value assigned, a null must be specified.

| CPRB Field | Content/Description |
|---|---|
| Function code | LP |
| Request DATA length | 0 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | A null-terminated string containing the drive, path, file name, and extension of the program to be loaded.<br><br>A null-terminated string containing the input arguments (separated by blanks) to be passed to the loaded program. |

## Query applications (QA function)

This function is requested by the client to determine which applications and topics are available for a DDE client application which is the LANDP application.

When application is set to null then any application present can have the related topic. When topic is set to null, all the topics of the application are related.

| CPRB Field | Content/Description |
|---|---|
| Function code | QA |
| Request DATA length | 0 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Application, Topic (must be separated by a comma) or<br>Topic or<br>Application or<br>Nothing |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Data, consisting of pairs of 'Application,Topic' (separated by commas)<br><br>Pairs are themselves separated from other pairs by *nulls* |

## Run command (RC function)

This function is requested by the client to instruct a DDE server application to perform a specified command.

Because RC is a high-level function, it is not necessary to specify the handle of the conversation. The conversation must not have been previously opened by requesting an IC function.

The syntax of the command must conform to relevant *concrete application rules* which are interchange rules defined by the target DDE application. The specified command must be valid and should retain the DDE command format.

Make sure the DDE server application accepts the command requested. Consult the corresponding information available with the DDE server application.

### Standard DDE command format rules

1. There can be more than one command entered together, but each single command must be contained in brackets.
   For example, [OPEN(″SHEET1.SHT″)][BEEP()]
2. The parameters of a command must be contained in parenthesis.
   For example, [DRAW(″BARCHAR″)]
3. The parameters of a command must be linked by commas.
   For example, [ADD(3,7)]
4. Commas and parentheses cannot appear in parameters, except inside a quoted string.
   For example, [CONCAT(″aurea″,″,dicta″)

| CPRB Field | Content/Description |
|---|---|
| Function code | RC |
| Request DATA length | 0 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Application, Topic, Item, Command (must be separated by commas) |

## Unlock topic (UL function)

This function is requested by the client to release exclusive access to a topic for a DDE client application which is the LANDP application. Any LANDP application can then have access to the unlocked topic, until a lock (LO) is requested.

A topic can be unlocked before the application finishes a conversation using this topic.

| CPRB Field | Content/Description |
|---|---|
| Function code | UL |
| Request DATA length | 0 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Application, Topic (must be separated by a comma) |

## DDE access server low-level functions

The DDE access server provides a set of *low-level functions* that enable LANDP client programs to perform DDE operations with full control over the various conversations. The LANDP client must manage the opening and closing of conversations, and take account of all relevant DDE rules and conventions. In the *high-level functions* the DDE access server does such processing *automatically*.

The explicit control of conversations provides a better response time when many requests that involve the same application, topic, or item, or several items of an application or topic, are made. This is because for each request, the DDE access server will not open and close a conversation.

The low-level functions allow you to establish “hot links” between the LANDP clients and DDE server applications. Each time an item value changes, the DDE server application can send the data to the DDE access server. The DDE access server then notifies the client of the update using the standard LANDP event notification mechanism. The client can use WM and AA functions to be notified of such a change.

The DDE access server also notifies the client using another event identifier when the DDE server application has closed the conversation.

The low-level functions provided by the DDE access server are:
- Initiate Conversation (IC)
- Terminate Conversation (TC)
- Poke Data (PD) into an item
- Request Data (RD) item value
- Advise (AD) — establish a link with an item
- Unadvise (UN) — release a link with an item
- Peek Data (KD) returned in response to a previous request

- Execute Command (EC)

## Overview of low-level functions use

The first step must be to initiate a conversation using the IC function. Thereafter, other functions can be requested. The LANDP client can:

- Insert data using the PD function
- Request data using the RD function (requesting the data item value only once), or AD function (establishing a link between the DDE server application and the LANDP client, using the DDE access server and requesting multiple values)
- Instruct the DDE server application to perform an application command by requesting the EC function

For the AD and RD functions:

- The DDE access server notifies the LANDP client each time it receives data from the DDE server application.
- The LANDP client can use WM supervisor local function to obtain the notification that data is available.
- The returned data can be obtained by the LANDP client, by requesting the KD function.

After an AD function request, the LANDP client ends the established links by requesting the UN function.

Finally the LANDP client should release the conversation by requesting the TC function. No more requests to the particular conversation (application or topic) can be made before a new conversation is opened.

## Request reference for low-level functions

This section describes the functions that clients can request for low-level functions, from the DDE access server. They are listed in alphabetical order of function code.

## Advise (AD function)

This function is requested by the client to establish a link with an *item*. Each time the value of item changes, an asynchronous event notification message is sent to the client. To obtain this value, the client should request a KD (peek data) function using the transaction handle.

After the value changes, previous values of *item* that were not received by the client are lost.

You must specify the conversation handle that was returned by a previous IC function request.

When a hot link for an item has been requested, it is not then possible within the same conversation to request data or establish another link with the same item until the link has been successfully closed by the client.

As the result of requesting the AD function, the application may receive an asynchronous event notification with the transaction handle as the event ID, telling that new data is available.

| CPRB Field | Content/Description |
|---|---|
| Function code | AD |
| Request DATA length | 0 |
| Request PARMLIST length | 2 |
| Reply DATA length | 2 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 2 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Conversation handle |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Item name |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Transaction handle |

## Execute command (EC function)

This function is requested by the client when a DDE server application is required to perform a specified command.

The conversation must already have been opened with an IC function request. You must also have specified the handle of the conversation.

See the “Run command (RC function)” on page 440 for some standard DDE server application rules.

Make sure the DDE server application accepts the command requested. Consult the corresponding information available with the DDE server application.

| CPRB Field | Content/Description |
|---|---|
| Function code | EC |
| Request DATA length | 0 |

| CPRB Field | Content/Description |
|---|---|
| Request PARMLIST length | 2 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Conversation handle |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Item,Command (must be separated by a comma) |

## Initiate conversation (IC function)

This function is requested by the client to initiate a DDE conversation and must always be requested before other low-level functions are requested.

A conversation handle that is used for identifying the conversation is returned in Reply DATA, which should be used when making other requests within the conversation.

As the result of requesting the IC function, the application may receive an asynchronous event notification with the conversation handle as the event ID, telling that the conversation has ended.

| CPRB Field | Content/Description |
|---|---|
| Function code | IC |
| Request DATA length | 0 |
| Request PARMLIST length | 0 |
| Reply DATA length | 2 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 2 |
| Replied PARMLIST length | 26 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Application, Topic (must be separated by a comma) |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Conversation handle |

## Peek data (KD function)

This function is requested by the client to obtain data that was previously requested using an RD or AD function.

The transaction handle that was returned by the previous RD or AD function, must be specified in Request PARMLIST.

Data received is a null-terminated string.

| CPRB Field | Content/Description |
|---|---|
| Function code | KD |
| Request DATA length | 0 |
| Request PARMLIST length | 2 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Transaction handle |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Data received from the DDE server application (null-terminated) |

## Poke data (PD function)

This function is requested by the client to insert data into the conversation item.

You must specify the handle of the conversation and the conversation must have been previously opened by requesting an IC function.

The data to be inserted must match the appropriate format required by the DDE server application.

| CPRB Field | Content/Description |
|---|---|
| Function code | PD |
| Request DATA length | 0 |
| Request PARMLIST length | 2 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Conversation handle |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Item,Data (must be separated by a comma) |

## Request data (RD function)

This function is requested by the client to obtain information about the conversation item. It is necessary to specify the handle of the conversation, and the conversation must have been previously opened by requesting an IC function.

Although a successful return code confirms that the request has been successfully sent to the DDE server application, it does *not* confirm that the request has been worked on or accepted by the DDE server application. This is because the request is asynchronously sent.

As the result of requesting the RD function, the application may receive an asynchronous event notification with the transaction handle as the event ID, telling that new data is available.

| CPRB Field | Content/Description |
|---|---|
| Function code | RD |
| Request DATA length | 0 |
| Request PARMLIST length | 2 |
| Reply DATA length | 2 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 2 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Conversation handle |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Item name |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Transaction handle |

## Terminate conversation (TC function)

This function is requested by the client to terminate a DDE conversation.

Because the DDE access server must be capable of identifying the conversation that is to be closed, it is necessary to specify the handle of the conversation.

After TC has been requested, it is not possible to request any other function that uses this terminated conversation.

| CPRB Field | Content/Description |
|---|---|
| Function code | TC |
| Request DATA length | 0 |
| Request PARMLIST length | 2 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Conversation handle |

## Unadvise (UN function)

This function is requested by the client to close a link with the transaction started by a previous AD (Advise) function.

It is necessary to specify the handle of the transaction that was returned with the previous AD function request.

| CPRB Field | Content/Description |
|---|---|
| Function code | UN |
| Request DATA length | 0 |
| Request PARMLIST length | 2 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Transaction handle |

## DDE access server clipboard functions

The DDE access server provides a set of *clipboard functions* that provide simplified access to the clipboard. The LANDP client can perform basic clipboard functions by requesting a function from the DDE access server.

The clipboard functions provided by the DDE access server are:
- Copy text (CT) in text format into the clipboard.
- Paste text (PT) in text or link format from the clipboard.

## Request reference for clipboard functions

This section describes the functions that clients can request for clipboard functions, from the DDE access server. They are listed in alphabetical order of function code.

### Copy text (CT function)

This function is requested by the client to copy data to the clipboard. The data entered in Request DATA must be in text format.

Previous data inserted by this server or by any other OS/2 application, is lost. You should also note that data inserted in the clipboard can be changed by any other OS/2 application.

| CPRB Field | Content/Description |
|---|---|
| Function code | CT |
| Request DATA length | 0 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |

| CPRB Field | Content/Description |
|---|---|
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Data |

## Paste text (PT function)

This function is requested by the client to obtain data from the clipboard.

The data can be obtained in text or link format. When the data is in link format, it can be used to establish DDE conversations.

| CPRB Field | Content/Description |
|---|---|
| Function code | PT |
| Request DATA length | 0 |
| Request PARMLIST length | 0 to 2 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | X'0000' for text format, or X'0001' for link format |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Data (null-terminated)<br>or<br>Application, Topic, Item (null-terminated and separated by commas) |

## Examples using the high-level functions

The following sections provide examples which demonstrate how each high-level function is coded with RMTREQ.

### Example of using get data

This function gets the data values of an item. It opens a conversation with the entered application and topic, requests a data item value, receives the value returned, and finally closes the conversation.

In the example below, the request returns the values of the four first cells of the first row of ″Benefits″ spreadsheet, of an application with the name ″Cinco″. The CPRB entries for function GD are described in “Get data (GD function)” on page 437.

```
cprb.Server Name   = EHCLAD
cprb.Function code = GD
cprb.Request DATA  = "Cinco,Benefits,R1C1:R1C4"
RMTREQ(cprb)
cprb.Reply DATA    = Cell Values
```

### Example of using insert data

This function assigns data to an item. The function opens a DDE conversation with the entered application and topic, inserts the data value passed to the appropriate item, and finally closes the conversation. In the example below, the request assigns the value ″21″ to the item ″Hours″ of an application called ″Clock″, for the topic ″Time″. The CPRB entries for function ID are described in “Insert data (ID function)” on page 438.

```
cprb.Server name   = EHCLAD
cprb.Function code = ID
cprb.Request DATA  = "Clock,Time,Hours,21"
RMTREQ(cprb)
```

### Example of using run command

This function instructs a DDE server application to perform a specified command. The function opens a conversation, sends the required command to the DDE server application, and finally closes the conversation. In the example below, the request invokes the command ″ArrangeAll″ for an application ″Cinco″. The CPRB entries for function RC are described in “Run command (RC function)” on page 440.

```
cprb.Server name   = EHCLAD
cprb.Function code = RC
cprb.Request DATA  = "Cinco,System,,[ArrangeAll()]"
RMTREQ(cprb)
```

### Example of using query application

This function determines which DDE server applications are available, and which topics are supported by each of these applications. The response is a null-terminated string, which consists of pairs of “applications” and “topics”. The pairs are separated by nulls (X'00'), and the individual “applications” and “topics” are separated by commas. In this example, the request determines the DDE server applications that are available with ″System″. The answer could, for example, be the following null-terminated string:

″Cinco,System*0*Clock,System*0*″

The CPRB entries for function QA are described in “Query applications (QA function)” on page 440.

```
cprb.Server name   = EHCLAD
cprb.Function code = QA
cprb.Request DATA  = ,System
RMTREQ(cprb)
cprb.Replied DATA  = Responses
```

## Example of using load program

This function starts a program which is a DDE server application. Parameters can be passed to the program. In the example below, the request starts an application program called ″Cinco″, contained in the directory ″d:\CincoOs2″ and passes a parameter identifying the spreadsheet ″Benefits.wgs″, contained in the directory ″d::mydir″.

The CPRB entries for function LP are described in “Load program (LP function)” on page 439.

```
cprb.Server name   = EHCLAD
cprb.Function code = LP
cprb.Request
DATA  = "d:\CincoOs2\Cinco.exe\0d:\mydir\Benefits.wgs"
RMTREQ(cprb)
```

## Example of using lock topic

This function locks a topic of a DDE server application. Other LANDP applications are prevented from establishing a conversation with this topic. In the example below, the request locks the topic ″Time″ that is used in an application ″Clock″.

The CPRB entries for function LO are described in “Lock topic (LO function)” on page 439.

```
cprb.Server name   = EHCLAD
cprb.Function code = LO
cprb.Request DATA  = "Clock,Time"
RMTREQ(cprb)
```

## Example of using unlock topic

This function unlocks a topic of a DDE server application. Other LANDP applications can now establish a conversation with this topic. In the example below, the request unlocks the topic ″Time″ that is used in an application ″Clock″.

The CPRB entries for function UL are described in “Unlock topic (UL function)” on page 441.

```
cprb.Server name   = EHCLAD
cprb.Function code = UL
cprb.Request DATA  = "Clock,Time"
RMTREQ(cprb)
```

## Examples using the low-level functions

The following examples show how the low-level functions can be used. They correspond to those given for the high-level functions.

### Example of inserting a data value

This example inserts the value '21' into a DDE server application called ″Clock″, into its topic ″Time″ and item ″Hour″.

Compare this example with the one provided by “Example of using insert data” on page 451.

```
/* Open a new conversation. A conversation handle is obtained that will
   be used in the next functions dealing with this application, topic. */

          cprb.Server name   = EHCLAD
          cprb.Function code = IC
          cprb.Request DATA  = "Clock,Time"
          RMTREQ(cprb)
          cprb.Replied DATA  = conversation handle

/* Insert the value 21 into the item hour.  The values should keep the
   format specified by the application. */

          cprb.Server name      = EHCLAD
          cprb.Function code    = PD
          cprb.Request PARMLIST = conversation handle
          cprb.Request DATA     = "Hours,21"
          RMTREQ(cprb)


/* Close the conversation. */

          cprb.Server name      = EHCLAD
          cprb.Function code    = TC
          cprb.Request PARMLIST = conversation handle
          RMTREQ(cprb)
```

### Example of obtaining a data item value

This example requests and obtains the values contained in the first four cells of a spreadsheet called ″Cinco″ from its sheet ″Benefits″.

Compare this example with the one provided by “Example of using get data” on page 451.

```
/* Open a new conversation. A conversation handle is obtained that will
   be used in the next functions dealing with this application and
   topic. */

          cprb.Server name   = EHCLAD
          cprb.Function code = IC
          cprb.Request DATA  = "Cinco,Benefits"
          RMTREQ(cprb)
          cprb.Replied DATA  = conversation handle
```

```
/* Request the data from the DDE server application. */

           cprb.Server name      = EHCLAD
           cprb.Function code    = RD
           cprb.Request PARMLIST = conversation handle
           cprb.Request DATA     = "R1C1:R1C4"
           RMTREQ(cprb)
           cprb.Replied DATA     = transaction handle

/* Wait for the SPV notification that data is now available. */

           cprb.Server name      = SPV
           cprb.Function code    = WM
           cprb.Request DATA     = transaction handle + "EHCLAD  "
                                 + conversation handle + "EHCLAD  "
           RMTREQ(cprb)
           cprb.Replied PARMLIST = EventId

/* Obtain the data from the DDE access server. */

           cprb.Server name      = EHCLAD
           cprb.Function code    = KD
           cprb.Request PARMLIST = transaction handle
           RMTREQ(cprb)
           Replied DATA          = Data value

/* Terminate the conversation. */

           cprb.Server name      = EHCLAD
           cprb.Function code    = TC
           cprb.Request PARMLIST = conversation handle
           RMTREQ(cprb)
```

## Example of performing a command

This example does the ″ArrangeAll″ command of the spreadsheet application ″Cinco″ to arrange all the sheets that are visible on the screen.

Compare this example with the one provided by “Example of using run command” on page 451.

```
/* Open a new conversation. A conversation handle is obtained that will
   be used in the next functions dealing with this application, topic. */

           cprb.Server name   = EHCLAD
           cprb.Function code = IC
           cprb.Request DATA  = "Cinco,System"
           RMTREQ(cprb)
           cprb.Replied DATA  = conversation handle

/* Instruct the application command to be performed.
   The command should keep the format specified by the application. */

           cprb.Server name      = EHCLAD
           cprb.Function code    = EC
           cprb.Request PARMLIST = conversation handle
           cprb.Request DATA     = ",[ArrangeAll()]"
           RMTREQ(cprb)
```

```
/* Terminate the conversation. */

           cprb.Server name      = EHCLAD
           cprb.Function code    = TC
           cprb.Request PARMLIST = conversation handle
           RMTREQ(cprb)
```

## Example of establishing a link

This example establishes a hot-link with the values of the first four cells of the sheet ″Benefits″ of a spreadsheet application called ″Cinco″.

```
/* Open a new conversation. A conversation handle is obtained that will
   be used in the next functions dealing with this application, topic. */

           cprb.Server name   = EHCLAD
           cprb.Function code = IC
           cprb.Request DATA  = "Cinco,Benefits"
           RMTREQ(cprb)
           cprb.Replied DATA  = conversation handle

/* Request that a hot-link is initiated. */

           cprb.Server name      = EHCLAD
           cprb.Function code    = AD
           cprb.Request PARMLIST = conversation handle
           cprb.Request DATA     = "R1C1:R1C4"
           RMTREQ(cprb)
           cprb.Replied DATA     = transaction handle

/* Wait for the SPV notification that new data is now available.
   This means that the value of the item has now changed. */

           cprb.Server name      = SPV
           cprb.Function code    = WM
           cprb.Request DATA     = transaction handle + "EHCLAD  "
                                 + conversation handle + "EHCLAD  "
           RMTREQ(cprb)
           cprb.Replied PARMLIST = EventId

/* Obtain the data from the DDE access server. */

           cprb.Server name      = EHCLAD
           cprb.Function code    = KD
           cprb.Request PARMLIST = transaction handle
           RMTREQ(cprb)
           cprb.Replied DATA     = Data received

/* Close the hot-link. */

           cprb.Server name      = EHCLAD
           cprb.Function code    = UN
           cprb.Request PARMLIST = transaction handle
           RMTREQ(cprb)
```

```
/* Terminate the conversation. */

          cprb.Server name      = EHCLAD
          cprb.Function code    = TC
          cprb.Request PARMLIST = conversation handle
          RMTREQ(cprb)
```

## Examples using the clipboard functions

These examples show how the clipboard functions can be used.

### Example of copying text

This example inserts the value of six cells of a spreadsheet into the clipboard.

```
cprb.Server name   = EHCLAD
cprb.Function code = CT
cprb.Request DATA  = "January\t33\r\nFebruary\t35\r\nMarch\t40\r\n"
RMTREQ(cprb)
```

### Example of pasting text

This example requests and obtains the values contained in the application, topic, and item names of a DDE server application inserted into the clipboard. The response is a null-terminated string, that could be:

″Cinco,Benefits,R1C1:R1C4\0″

```
cprb.Server name       = EHCLAD
cprb.Function code     = PT
cprb.Request PARAMETER = 1
RMTREQ(cprb)
cprb.Reply DATA        = Data
```

# Part 6. System management servers

The LANDP family provides two servers that allow you to manage your system. This part of the book describes the functions of the system management server and the local resource manager. It contains the following chapters:

**Chapter 21, “System manager server” on page 459**

The system manager server provides common services that allow the administrator to maintain an entire LANDP workgroup from one LANDP for DOS or LANDP for OS/2 workstation. The functions of the server are to:

- Identify and control LANDP users
- Maintain user profiles and data
- Synchronize date and time
- Retrieve record definition structures
- Manage system and user logs and alerts

This chapter presents the functions provided by this server for servicing requests originating in application programs or other servers.

**Chapter 22, “Local resource manager” on page 513**

The local resource manager provides an API which makes the functions of the operator interface available for application programs. Using the local resource manager, the application program can also monitor the status of the printer manager and control the 3270 and 3287 emulators that are installed in the same workstation. Either the operator interface or the local resource manager can be installed in a LANDP for DOS workstation.

This chapter presents the functions provided by this server for servicing requests originating in application programs or other servers.

© Copyright IBM Corp. 1992, 2002

# Chapter 21. System manager server

This chapter provides information on how the server handles service requests. All functions operate in the LANDP for DOS, LANDP for OS/2, LANDP for Windows, and LANDP for Linux environments, except where stated otherwise.

This chapter also provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields. If you need more information about the CPRB fields, see Appendix A, “Connectivity programming request block” on page 669.

Functions supported by the system management server are grouped as follows. Within each group the functions are listed in alphabetical order of function code.

- User identification and control functions (page 462)
- User profile and application program data maintenance (page 474)
- Date and time synchronization (page 484)
- LANDP workgroup system status and common data maintenance (page 487)
- Retrieval of defined record structures (page 493)
- Data validation with defined record structures (page 497)
- Alerts management (page 498)
- System and user LOG management (page 505)

In the following table, the first and second columns give the function code and the name of the function. The third column shows the operating environment of the function. “02NL” means that it is available from LANDP for OS/2, LANDP for DOS, LANDP for Windows, and LANDP for Linux servers. The last column refers to the page where you can find the function described.

*Table 42. Function Codes used in the System Manager Server*

| Function code | Description | Env. | Page |
|---|---|---|---|
| **User identification and control functions** | | | |
| CP | Change password | 02NL | 463 |
| GL | Get signed-on user | 02NL | 463 |
| GI | Get signed-on user (Year-2000) | 02NL | 464 |
| GP | Get signed-on PC | 02NL | 465 |
| GW | Get signed-on PC (Year-2000) | 02NL | 466 |
| RE | Retrieve defined users list | 02NL | 467 |
| RF | Remote sign-off | 02NL | 467 |
| RK | Retrieve lists of users holding locks on resources | 02NL | 468 |
| RO | Retrieve signed-on users list | 02NL | 469 |
| GO | Retrieve signed-on users list (Year-2000) | 02NL | 470 |
| SF | Sign-off | 02NL | 471 |
| SN | Sign-on | 02NL | 471 |

© Copyright IBM Corp. 1992, 2002

*Table 42. Function Codes used in the System Manager Server (continued)*

| Function code | Description | Env. | Page |
|---|---|---|---|
| SO | Sign-on (Year-2000) | 02NL | 473 |
| **User profile and application program data maintenance:** | | | |
| AU | Add user to user profile file | 02NL | 476 |
| AX | Add user to user profile file (Year 2000) | 02NL | 476 |
| DU | Delete user in user profile file | 02NL | 477 |
| GU | Retrieve application program data | 02NL | 477 |
| RN | Retrieve user profile by record number | 02NL | 478 |
| XN | Retrieve user profile by record number (Year 2000) | 02NL | 479 |
| RU | Retrieve user profile | 02NL | 480 |
| RX | Retrieve user profile (Year 2000) | 02NL | 481 |
| SU | Update application program data | 02NL | 482 |
| UU | Update user profile | 02NL | 483 |
| UX | Update user profile (Year 2000) | 02NL | 484 |
| **Data and time synchronization:** | | | |
| GD | Retrieve system date and time | 02NL | 485 |
| SD | Set system date and time | 02N- | 486 |
| **LANDP workgroup system status and common data maintenance:** | | | |
| GG | Get system status | 02NL | 487 |
| RD | Retrieve LANDP workgroup common data | 02NL | 489 |
| UA | Update alert status | 02NL | 489 |
| UD | Update LANDP workgroup common data | 02NL | 490 |
| UI | Update LANDP workgroup identification | 02NL | 491 |
| UL | Update logging status | 02NL | 491 |
| UM | Update operator message status | 02NL | 492 |
| UO | Update message operator receiver | 02NL | 492 |
| **Retrieval of defined record structures:** | | | |
| RR | Retrieve record format | 02NL | 493 |
| **Data validation with defined record structure:** | | | |
| VR | Validate record | 02NL | 497 |
| **Alerts management:** | | | |
| AN | Alerts notification | 02NL | 499 |
| MO | Send message to NetView®* operator | 02NL | 502 |
| UN | Resolution notification | 02NL | 503 |
| CN | Check NMVT (in exit routine) | 02NL | 503 |

*Table 42. Function Codes used in the System Manager Server (continued)*

| Function code | Description | Env. | Page |
|---|---|---|---|
| **System and user log management:** | | | |
| RL | Retrieve LOG record | 02NL | 507 |
| R1 | Retrieve LOG record (Year 2000) | 02NL | 509 |
| WL | Write LOG record | 02NL | 510 |

You can find more information about system authorization levels, using the LEVELx keywords, in the SMGRUSER and SMGRPRF vectors in the *LANDP Installation and Customization* book.

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 26 | 2 | 26 | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 50 | 8 | Reserved | Resource origin |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | SMGR | Server name |

The following fields are variable and are discussed in each function request description:
- Function code
- Request PARMLIST length
- Request DATA length
- Reply DATA length

The following fields are discussed for the function requests where they take a different value from the one they have in the former table:
- Reply PARMLIST length
- Resource origin

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 44 | 2 | | Replied PARMLIST length |
| 46 | 2 | | Replied DATA length |

If the request was successful, the *router return code* and the *server return code* are both X'00000000'. In all other cases, see the appropriate section in the *LANDP Problem Determination* book to see if you should take any action. The return values in Reply PARMLIST and DATA should be ignored if there is an error.

The following fields are variable and are discussed in each function request description:
- Replied PARMLIST length
- Replied DATA length

**PARMLIST:** The Request PARMLIST and Reply PARMLIST fields for the system manager server are discussed in the function descriptions.

**DATA:** Request DATA and Reply DATA contain data to be sent to or received from the server. The use of these areas is explained in the description of each function request.

## User identification and control functions

In this section you can find the programming information required to write application programs or your own servers that issue service requests to the system manager server to control access to different components of the LANDP licensed programs family.

The functions available and the authorization level required for access are:

| **Function** | **System authorization level** | | | |
|---|---|---|---|---|
| | **Operator (O)** | **Admin (A)** | **Master (M)** | **R, S, F, or none** |
| Change password (CP) | Yes | Yes | Yes | Yes |
| Get signed-on user (GL) | Yes | Yes | Yes | No |
| Get signed-on user (Year-2000) (GI) | Yes | Yes | Yes | No |
| Get signed-on PC (GP) | Yes | Yes | Yes | No |
| Get signed-on PC (Year-2000) (GW) | Yes | Yes | Yes | No |
| Retrieve defined users list (RE) | No | Yes | Yes | No |
| Remote sign-off (RF) | Yes | Yes | Yes | Yes |
| Retrieve lists of users holding locks (RK) | No | Yes | Yes | No |

| Function | System authorization level | | | |
|---|---|---|---|---|
| | Operator (O) | Admin (A) | Master (M) | R, S, F, or none |
| Retrieve signed-on users list (RO) | No | Yes | Yes | No |
| Retrieve signed-on users list (Year-2000) (GO) | No | Yes | Yes | No |
| Sign-off (SF) | Yes | Yes | Yes | Yes |
| Sign-on (SN) | Yes | Yes | Yes | Yes |
| Sign-on (Year-2000) (SO) | Yes | Yes | Yes | Yes |

Valid characters for user identification (user ID) and password are:
- English letters: A to Z
- English letters: a to z (translated to corresponding uppercase letters, for user ID only)
- Digits: 0 to 9
- Characters: @, #, $, and %

## Change password (CP function)

This function changes the password of a signed-on user. The new password must be at least four characters.

The next sign-on by the same user requires the new password.

| CPRB Field | Content/Description |
|---|---|
| Function code | CP |
| Request DATA length | 12 to 16 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request DATA Values | | |
|---|---|---|
| Offset | Length | Content |
| 0 | 8 | Current password |
| 8 | 8 | New password |

## Get signed-on user (GL function)

This function gets the identification of the signed-on user, the authorization level, and the date and time of the last logon on the specified workstation.

If the Reply DATA length is 20, the complete information is returned. If it is any other value, only the user ID is returned.

| CPRB Field | Content/Description |
|---|---|
| Function code | GL |
| Request DATA length | 2 |
| Request PARMLIST length | 0 |
| Reply DATA length | 4 to 20 |
| Reply PARMLIST length | 20 |
| Replied DATA length | 4 to 20 |
| Replied PARMLIST length | 20 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Workstation identification |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 10 | Workstation system authorization level |
| 10 | 10 | Workstation application program authorization level |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | User ID (4 to 8 characters) |
| 8 | 12 | Date and time of the last sign-on, in the form yymmddhhmmss |

## Get signed-on user (Year-2000) (GI function)

This function gets the identification of the signed-on user, the authorization level, and the date and time of the last logon on the specified workstation.

**Note:** The GI command is similar to the GL command except that the date is in the form yyyymmddhhmmss instead of yymmddhhmmss.

If the Reply DATA length is 22, the complete information is returned. If it is any other value, only the user ID is returned.

| CPRB Field | Content/Description |
|---|---|
| Function code | GI |
| Request DATA length | 2 |
| Request PARMLIST length | 0 |
| Reply DATA length | 4 to 22 |

| CPRB Field | Content/Description |
|---|---|
| Reply PARMLIST length | 20 |
| Replied DATA length | 4 to 22 |
| Replied PARMLIST length | 20 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Workstation identification |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 10 | Workstation system authorization level |
| 10 | 10 | Workstation application program authorization level |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | User ID (4 to 8 characters) |
| 8 | 14 | Date and time of the last sign-on, in the form yyyymmddhhmmss |

## Get signed-on PC (GP function)

This function gets the identification of the workstation (PC_ID) where the specified user ID is signed on, and the authorization level of that user, and the date and time of the last logon of that user.

| CPRB Field | Content/Description |
|---|---|
| Function code | GP |
| Request DATA length | 4 to 8 |
| Request PARMLIST length | 0 |
| Reply DATA length | 14 |
| Reply PARMLIST length | 20 |
| Replied DATA length | 14 |
| Replied PARMLIST length | 20 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | User ID (4 to 8 characters) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 10 | Workstation system authorization level |
| 10 | 10 | Workstation application program authorization level |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Workstation identification |
| 2 | 12 | Date and time of the last sign-on, in the form yymmddhhmmss |

## Get signed-on PC (Year-2000) (GW function)

This function gets the identification of the workstation (PC_ID) where the specified user ID is signed on, and the authorization level of that user, and the date and time of the last logon of that user.

**Note:** The GW command is similar to the GP command except that the date is in the form yyyymmddhhmmss instead of yymmddhhmmss.

| CPRB Field | Content/Description |
|---|---|
| Function code | GW |
| Request DATA length | 4 to 8 |
| Request PARMLIST length | 0 |
| Reply DATA length | 16 |
| Reply PARMLIST length | 20 |
| Replied DATA length | 16 |
| Replied PARMLIST length | 20 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | User ID (4 to 8 characters) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 10 | Workstation system authorization level |
| 10 | 10 | Workstation application program authorization level |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Workstation identification |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 14 | Date and time of the last sign-on, in the form yyyymmddhhmmss |

## Retrieve defined users list (RE function)

This function gets the IDs of the users defined in the FBSS#USP file at customization time.

| CPRB Field | Content/Description |
|---|---|
| Function code | RE |
| Request DATA length | 0 |
| Request PARMLIST length | 0 |
| Reply DATA length | At least 8 |
| Reply PARMLIST length | 2 |
| Replied DATA length | At least 8 |
| Replied PARMLIST length | 2 |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Number of users defined in the FBSS#USP file. |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | User ID 1 |
| (This structure is repeated for each user defined in the FBSS#USP file.) | | |

## Remote sign-off (RF function)

This function signals to the system manager server that a signed-on user requests to be disconnected either by that user or by an administrator. If the sign-off is by the user, the password is required in the Request DATA area. If the sign-off is by an administrator, no password is required, but a user with administrative authority must be signed on at the requesting workstation.

The RF function provides a parameter that indicates the type of sign-off used for updating the user profile.

**Note:** This function is performed even if the server return code is U5. The user profile is only updated if both the router and server return codes are X'00000000'.

| CPRB Field | Content/Description |
|---|---|
| Function code | RF |
| Request DATA length | 4 to 16 |
| Request PARMLIST length | 1 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Type of sign-off. Any value can be chosen, except 'S', 'O', and 'T', because they are reserved for the system manager server (shutdown), system manager operator interface, and trace (debug) tools, respectively. |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | User ID (from 4 to 8 characters) |
| 8 | 8 | Password (from 4 to 8 characters). |

## Retrieve list of users holding locks on LANDP resources (RK function)

This function gets information about the users who have locked the global LAN data, or the system or application data of themselves or of another user.

| CPRB Field | Content/Description |
|---|---|
| Function code | RK |
| Request DATA length | 0 |
| Request PARMLIST length | 0 |
| Reply DATA length | At least 43 |
| Reply PARMLIST length | 2 |
| Replied DATA length | At least 43 |
| Replied PARMLIST length | 2 |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Number of users who have locked data in the LAN |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Signed-on user who has locked the data |
| 8 | 1 | **'Y'** Global LAN data is locked by this user<br>**'N'** Global LAN data is not locked by this user (the default) |
| 9 | 1 | **'Y'** System data of a user is locked by this user<br>**'N'** System data of a user is not locked by this user (the default) |
| 10 | 1 | **'Y'** Application data of a user is locked by this user<br>**'N'** Application data of a user is not locked by this user (the default) |
| 11 | 8 | ID of the user who has system or application data locked by this user |
| 19 | 6 | System authorization level of this user |
| 25 | 4 | Blanks |
| 29 | 10 | Application program authorization level of this user |
| 39 | 2 | Requester workstation ID |
| 41 | 2 | Reserved |
| The entire structure is repeated for each user who has locked data. | | |

## Retrieve signed-on users list (RO function)

This function gets information about users who are currently signed on.

| CPRB Field | Content/Description |
|---|---|
| Function code | RO |
| Request DATA length | 0 |
| Request PARMLIST length | 0 |
| Reply DATA length | At least 42 |
| Reply PARMLIST length | 2 |
| Replied DATA length | At least 42 |
| Replied PARMLIST length | 2 |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Number of users signed on in the LAN |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | User ID |
| 8 | 6 | System authorization level of this user |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 14 | 4 | Blanks |
| 18 | 10 | Application program authorization level of this user |
| 28 | 2 | Workstation ID |
| 30 | 12 | Date and time of the last sign-on, in the form yymmddhhmmss |
| The entire structure is repeated for each user who is signed on. | | |

## Get signed-on users list (Year-2000) (GO function)

This function gets information about users who are currently signed on.

**Note:** The GO command is similar to the RO command except that the date is in the form yyyymmddhhmmss instead of yymmddhhmmss.

| CPRB Field | Content/Description |
|---|---|
| Function code | GO |
| Request DATA length | 0 |
| Request PARMLIST length | 0 |
| Reply DATA length | At least 44 |
| Reply PARMLIST length | 2 |
| Replied DATA length | At least 44 |
| Replied PARMLIST length | 2 |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Number of users signed on in the LAN |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | User ID |
| 8 | 6 | System authorization level of this user |
| 14 | 4 | Blanks |
| 18 | 10 | Application program authorization level of this user |
| 28 | 2 | Workstation ID |
| 30 | 14 | Date and time of the last sign-on, in the form yyyymmddhhmmss |
| The entire structure is repeated for each user who is signed on. | | |

## Sign-Off (SF function)

This function signals to the system manager server that a signed-on user requests to be disconnected.

The SF function provides a parameter that shows whether the previous sign-on was accepted by the application program, and identifies the type of sign-off used for updating the user profile, see also “Sign-On (SN function)”. and “Sign-On (Year-2000) (SO function)” on page 473.

**Note:** This function is performed even if the server return code is U5, or if the supervisor return code is L2, L4, L6, or L9. The user profile is only updated if both return codes are X'00000000'.

| CPRB Field | Content/Description |
|---|---|
| Function code | SF |
| Request DATA length | 0 |
| Request PARMLIST length | 2 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | This field shows if the previous sign-on was accepted by the application program<br>**'N'** Not accepted<br>**other** Accepted (default) |
| 1 | 1 | Type of sign-off. Any value can be chosen, except 'S', 'O', and 'T', because they are reserved for the system manager server (shutdown), system manager operator interface, and trace (debug) tools, respectively. |

## Sign-On (SN function)

This function identifies a user, validates the password, and returns the user profile system data. The initial content of the user profile system data is defined during customization, but it may have been changed, in the meantime, by the user or by an administrator.

If a system authorization scheme is in effect, signing-on is required before accessing other system manager services.

To achieve correct operation of this function, the user must not be signed-on at any other workstation, and the user profile must not be blocked by another user.

Only one user can be signed-on at a time. On return, the Reply DATA area contains the user profile for the signed-on user.

| CPRB Field | Content/Description |
|---|---|
| Function code | SN |
| Request DATA length | 12 to 16 |
| Request PARMLIST length | 0 |
| Reply DATA length | 108 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 108 |
| Replied PARMLIST length | 0 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | User ID (4 to 8 characters) |
| 8 | 8 | Password (4 to 8 characters). |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | User ID |
| 8 | 8 | Password (overlapped with blanks) |
| 16 | 12 | Date and time of last password change, see note 3 on page 473 |
| 28 | 30 | User name. This field is an ASCII character string, which can be used at convenience. |
| 58 | 2 | Language ID. Any numeric value allowed, it can be used at convenience. |
| 60 | 6 | System authorization level |
| 66 | 4 | Blanks |
| 70 | 10 | Application program authorization level |
| 80 | 1 | Blank |
| 81 | 2 | Number of unsuccessful sign-ons in binary, see note 1 |
| 83 | 12 | Date and time of last sign-on, see note 3 on page 473 |
| 95 | 12 | Date and time of last sign-off, see note 3 on page 473 |
| 107 | 1 | Type of sign-off, see note 2 on page 473 |

**Notes:**

1. Each time the user fails to enter the password to sign-on, the system manager server increases the associated counter. When the sign-on is accepted, the system manager server passes it to the application program, then the application program may decide if it accepts or rejects the sign-on.

If this counter value is three or more, the sign-on to the system manager operator interface or to the trace (debug) tools is rejected.

2. This field is provided by the application program during sign-off, or by the system manager server at shutdown, see page 471. Any character value is allowed except 'S' (reserved for the shutdown), 'O' (reserved for the system manager operator interface to show that the last time a user was signed-on, it was from the system manager operator interface), and 'T' (reserved for the trace (debug) tools for the same reason).
3. Date and time are in the form: yymmddhhmmss.

## Sign-On (Year-2000) (SO function)

This function identifies a user, validates the password, and returns the user profile system data. The initial content of the user profile system data is defined during customization, but it may have been changed, in the meantime, by the user or by an administrator.

**Note:** The SO command is similar to the SN command except that the date is in the form yyyymmddhhmmss instead of yymmddhhmmss.

If a system authorization scheme is in effect, signing-on is required before accessing other system manager services.

To achieve correct operation of this function, the user must not be signed-on at any other workstation, and the user profile must not be blocked by another user.

Only one user can be signed-on at a time. On return, the Reply DATA area contains the user profile for the signed-on user.

| CPRB Field | Content/Description |
|---|---|
| Function code | SO |
| Request DATA length | 12 to 16 |
| Request PARMLIST length | 0 |
| Reply DATA length | 114 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 114 |
| Replied PARMLIST length | 0 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | User ID (4 to 8 characters) |
| 8 | 8 | Password (4 to 8 characters). |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | User ID |
| 8 | 8 | Password (overlapped with blanks) |
| 16 | 14 | Date and time of last password change, see note 3 |
| 30 | 30 | User name. This field is an ASCII character string, which can be used at convenience. |
| 60 | 2 | Language ID. Any numeric value allowed, it can be used at convenience. |
| 62 | 6 | System authorization level |
| 68 | 4 | Blanks |
| 72 | 10 | Application program authorization level |
| 82 | 1 | Blank |
| 83 | 2 | Number of unsuccessful sign-ons in binary, see note 1 |
| 85 | 14 | Date and time of last sign-on, see note 3 |
| 99 | 14 | Date and time of last sign-off, see note 3 |
| 113 | 1 | Type of sign-off, see note 2 |

**Notes:**

1. Each time the user fails to enter the password to sign-on, the system manager server increases the associated counter. When the sign-on is accepted, the system manager server passes it to the application program, then the application program may decide if it accepts or rejects the sign-on.

   If this counter value is three or more, the sign-on to the system manager operator interface or to the trace (debug) tools is rejected.
2. This field is provided by the application program during sign-off, or by the system manager server at shutdown, see page 471. Any character value is allowed except 'S' (reserved for the shutdown), 'O' (reserved for the system manager operator interface to show that the last time a user was signed-on, it was from the system manager operator interface), and 'T' (reserved for the trace (debug) tools for the same reason).
3. Date and time are in the form: yyyymmddhhmmss.

## User profile and application program data maintenance

This section gives you the programming information required to write application programs or your own servers that issue service requests to the system manager server to maintain the user profile and application program data.

The functions available and the authorization level required for using them are:

| Function | System authorization level | | | |
|---|---|---|---|---|
| | O | A | M | R, S, F, or none |
| Add user to user profile file (AU) | No | Yes | Yes | No |
| Add user to user profile file (Year-2000) (AX) | No | Yes | Yes | No |
| Delete user in user profile file (DU) | No | Yes | Yes | No |
| Retrieve application data (GU) | Yes (note 1) | Yes | Yes | Yes (note 1) |
| Retrieve user profile by record number (RN) | No | Yes | Yes | No |
| Retrieve user profile by record number (Year-2000) (XN) | No | Yes | Yes | No |
| Retrieve user profile (RU) | Yes (note 1) | Yes | Yes | Yes (note 1) |
| Retrieve user profile (Year2000) (RX) | Yes (note 1) | Yes | Yes | Yes (note 1) |
| Update application data (SU) | Yes (note 2) | Yes | Yes | Yes (note 2) |
| Update user profile (UU) | No | Yes | Yes | No |
| Update user profile (Year-2000) (UX) | No | Yes | Yes | No |

**Notes:**

1. Functions GU, RU and RX: Users without the authorization level A or M can only retrieve their own user profile and application data.
2. Function SU: Users without the authorization level A or M can only update their own application data.

The retrieve application data (GU) for update locks the application data. The retrieve user profile (RU) and retrieve user profile (Year-2000) (RX) for update locks the user profile data. In both cases, no other user can retrieve any of this data for update. The data remains locked until the update function is performed. You cannot retrieve data for update from any other user before completing the update function.

While users are signed on, their profile and application data remains locked and no other user with administrator authorization level can retrieve for update.

While a user profile is locked, its data can be retrieved by another authorized user. However, an attention code is returned to this user.

## Add user to user profile file (AU function)

This function adds a new user profile to the user profile file. It can be used only by users with the appropriate authorization level. The record number in the user profile file that contains the information about this user is returned in Reply PARMLIST.

This function creates the new user record but does not supply the application program data. This means that you must perform the SU function (update application data) to complete the definition of the new user. The new user application program data remains locked until the SU function is performed.

| CPRB Field | Content/Description |
|---|---|
| Function code | AU |
| Request DATA length | 108 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 4 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 4 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 108 | User profile. Same format as described in function SN. Password must be entered. |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 4 | Record number. Four ASCII characters, X'30' to X'39' |

## Add user to user profile file (Year-2000) (AX function)

This function adds a new user profile to the user profile file. It can be used only by users with the appropriate authorization level. The record number in the user profile file that contains the information about this user is returned in Reply PARMLIST.

This function creates the new user record but does not supply the application program data. This means that you must perform the SU function (update application data) to complete the definition of the new user. The new user application program data remains locked until the SU function is performed.

**Note:** The AX command is similar to the AU command except that the dates are in the form yyyymmddhhmmss instead of yymmddhhmmss.

| CPRB Field | Content/Description |
|---|---|
| Function code | AX |
| Request DATA length | 114 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 4 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 4 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 114 | User profile. Same format as described in function SO. (page 473) Password must be entered. |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 4 | Record number. Four ASCII characters, X'30' to X'39' |

## Delete user in user profile file (DU function)

This function deletes a user profile and the application program data from the user profile file. This cannot be done while the particular user is signed-on. This function can only be used by those who have the appropriate authorization level.

| CPRB Field | Content/Description |
|---|---|
| Function code | DU |
| Request DATA length | 4 to 8 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | User ID (4 to 8 characters) |

## Retrieve application program data (GU function)

This function retrieves the current application program data for a specific user ID. It can be used by the owner of the data and by another authorized user. A non-owner needs

to request the GU function with “retrieve for update” option before another user application program data can be modified. The retrieve-for-update parameter locks the user data until the update function is performed. Before the update function is performed, no other retrieve for update is allowed. The record number on the user profile file that contains the information for this user is returned in Reply PARMLIST.

If application program data validation is selected during customization, this data must follow the structure of the FBSSUSPR record, which has been defined during customization.

| CPRB Field | Content/Description |
|---|---|
| Function code | GU |
| Request DATA length | 4 to 8 |
| Request PARMLIST length | 1 |
| Reply DATA length | 1 to 1024 |
| Reply PARMLIST length | 26 |
| Replied DATA length | Up to 1024 |
| Replied PARMLIST length | 4 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Type of retrieve:<br>**’U’** Retrieve for update<br>**’ ’** Normal retrieve (default) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | User ID (4 to 8 characters) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 4 | Record number. Four ASCII characters, X'30' to X'39' |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 to 1024 | User data (X'0001' to X'0400') |

## Retrieve user profile by record number (RN function)

This function retrieves the user profile by record number from the user profile file. This is used when it is necessary to retrieve many user profiles in sequence. The retrieved user profile can only be displayed. No *retrieve-for-update* is available with this function. The functions RU and UU are used for updating.

The number of the retrieved record is returned. This number can be used with another RN function to get either the preceding or the following record.

| CPRB Field | Content/Description |
|---|---|
| Function code | RN |
| Request DATA length | 0 |
| Request PARMLIST length | 5 |
| Reply DATA length | 108 |
| Reply PARMLIST length | 4 |
| Replied DATA length | 108 |
| Replied PARMLIST length | 4 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 4 | Record number. Four ASCII characters, X'30' to X'39' |
| 4 | 1 | Qualifier<br>**'+'** Get the record following that specified<br>**'–'** Get the record preceding that specified<br><br>Any other value is ignored |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 4 | Record number. Four ASCII characters, X'30' to X'39' |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 108 | User profile. Same format as described in function SN (page 471) |

## Retrieve user profile by record number (Year-2000) (XN)

This function retrieves the user profile by record number from the user profile file. This is used when it is necessary to retrieve many user profiles in sequence. The retrieved user profile can only be displayed. No *retrieve-for-update* is available with this function. The functions RX and UX are used for updating.

The number of the retrieved record is returned. This number can be used with another XN function to get either the preceding or the following record.

**Note:** The XN command is similar to the RN command except that the dates are in the form yyyymmddhhmmss instead of yymmddhhmmss.

| CPRB Field | Content/Description |
|---|---|
| Function code | XN |
| Request DATA length | 0 |
| Request PARMLIST length | 5 |
| Reply DATA length | 114 |
| Reply PARMLIST length | 4 |
| Replied DATA length | 114 |
| Replied PARMLIST length | 4 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 4 | Record number. Four ASCII characters, X'30' to X'39' |
| 4 | 1 | Qualifier<br>**'+'** Get the record following that specified<br>**'–'** Get the record preceding that specified<br><br>Any other value is ignored |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 4 | Record number. Four ASCII characters, X'30' to X'39' |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 114 | User profile. Same format as described in function SO (page 473) |

## Retrieve user profile (RU function)

This function retrieves the user profile of a specific user ID. It can be used either by the owner of the data or by another user with the appropriate authorization level. If this data is retrieved for modification by a non-owner, using the function UU, the parameter *retrieve for update* must be used, otherwise the update function is not accepted. This can be done only once, a following attempt returning an error code. Also, the record number on the user profile file that contains the information for this user is returned in Reply PARMLIST.

| CPRB Field | Content/Description |
|---|---|
| Function code | RU |
| Request DATA length | 4 to 8 |
| Request PARMLIST length | 1 |
| Reply DATA length | 108 |
| Reply PARMLIST length | 4 |

| CPRB Field | Content/Description |
|---|---|
| Replied DATA length | 108 |
| Replied PARMLIST length | 4 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | This field specifies the type of retrieve:<br>**'U'** Retrieve for update<br>**' '** Normal retrieve (default) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | User ID (4 to 8 characters) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 4 | Record number. Four ASCII characters, X'30' to X'39' |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 108 | User profile. Same format as described in function SN |

## Retrieve user profile (Year-2000) (RX function)

This function retrieves the user profile of a specific user ID. It can be used either by the owner of the data or by another user with the appropriate authorization level. If this data is retrieved for modification by a non-owner, using the function UX, the parameter *retrieve for update* must be used, otherwise the update function is not accepted. This can be done only once, a following attempt returning an error code. Also, the record number on the user profile file that contains the information for this user is returned in Reply PARMLIST.

**Note:** The RX command is similar to the RU command except that the dates are in the form yyyymmddhhmmss instead of yymmddhhmmss.

| CPRB Field | Content/Description |
|---|---|
| Function code | RX |
| Request DATA length | 4 to 8 |
| Request PARMLIST length | 1 |
| Reply DATA length | 114 |
| Reply PARMLIST length | 4 |
| Replied DATA length | 114 |

| CPRB Field | Content/Description |
|---|---|
| Replied PARMLIST length | 4 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | This field specifies the type of retrieve:<br>**'U'** Retrieve for update<br>**' '** Normal retrieve (default) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | User ID (4 to 8 characters) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 4 | Record number. Four ASCII characters, X'30' to X'39' |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 114 | User profile. Same format as described in function SO (page 473) |

## Update application program data (SU function)

This function updates the current application program data for a specific user ID.

A non-owner needs first to retrieve the user profile before this function can be requested. After performing the SU function successfully, application data is unlocked.

The content of Request DATA is checked against the record structure defined during customization, FBSSUSPR, if application program validation was selected during customization.

| CPRB Field | Content/Description |
|---|---|
| Function code | SU |
| Request DATA length | 1 to 1024 |
| Request PARMLIST length | 4 to 8 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| CPRB Field | Content/Description |
|---|---|
| **Note:** If Request DATA length is not X'0000' and the SU function returns a nonzero return code, the condition *update pending* is not reset. That is, the user application program data remains locked.<br><br>If Request DATA length is X'0000', no update is done, but any condition *update pending*, set after a GU or AU function, is removed. | |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | User ID (4 to 8 characters) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 to 1024 | User data (X'0001' to X'0400') |

## Update user profile (UU function)

This function updates the current user profile of a specific user ID. Before this function can be used by a non-owner, the user profile must be retrieved using the function RU with the parameter 'U' (retrieve for update).

The content of Request DATA is validated and, if accepted, the user profile file is updated.

| CPRB Field | Content/Description |
|---|---|
| Function code | UU |
| Request DATA length | 108 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |
| **Note:** If Request DATA length is not X'0000' and the UU function returns a nonzero return code, the condition *update pending* is not reset.<br><br>If Request DATA length is X'0000', no update is done, but any condition *update pending*, set after an RU function, is removed. | |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 108 | User profile. Same format as described in function SN. If *password* contains blanks, the current password is maintained, otherwise the new password is validated. |

## Update user profile (Year-2000) (UX function)

This function updates the current user profile of a specific user ID. Before this function can be used by a non-owner, the user profile must be retrieved using the function RX with the parameter 'U' (retrieve for update).

The content of Request DATA is validated and, if accepted, the user profile file is updated.

**Note:** The UX command is similar to the UU command except that the dates are in the form yyyymmddhhmmss instead of yymmddhhmmss.

| CPRB Field | Content/Description |
|---|---|
| Function code | UX |
| Request DATA length | 114 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |
| **Note:** If Request DATA length is not X'0000' and the UX function returns a nonzero return code, the condition *update pending* is not reset.<br><br>If Request DATA length is X'0000', no update is done, but any condition *update pending*, set after an RX function, is removed. | |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 114 | User profile. Same format as described in function SO. If *password* contains blanks, the current password is maintained, otherwise the new password is validated. |

## Date and time synchronization

This section gives you the programming information required to write application programs or your own servers that issue service requests to the system manager server for date and time synchronization among the workstations in a LANDP workgroup.

The functions available and the authorization level required for using them are:

| Function | System authorization level | | | |
|---|---|---|---|---|
| | O | A | M | R, S, F, or none |
| Retrieve system date and time (GD) | Yes | Yes | Yes | Yes |
| Set system date and time (SD) | Yes | No | Yes | No |
| **Note:** No authorization level is needed for using the GD function. | | | | |

## Retrieve system date and time (GD function)

This function retrieves the system date and time maintained by the operating system in the workstation that contains the system manager server.

Date and time are returned in both binary and character format.

| CPRB Field | Content/Description |
|---|---|
| Function code | GD |
| Request DATA length | 0 |
| Request PARMLIST length | 0 |
| Reply DATA length | 22 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 22 |
| Replied PARMLIST length | 0 |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Hours. 0 to 23 in binary |
| 1 | 1 | Minutes. 0 to 59 in binary |
| 2 | 1 | Seconds. 0 to 59 in binary |
| 3 | 1 | Day of the week. 0=Sunday, 1=Monday, and so on, in binary |
| 4 | 1 | Day of the month. 1 to 31 in binary |
| 5 | 1 | Month. 1 to 12 in binary |
| 6 | 2 | Year. 1980 to 2099 in binary (in word-reversed mode) |
| 8 | 2 | Hours. 0 to 23 as two ASCII characters |
| 10 | 2 | Minutes. 0 to 59 as two ASCII characters |
| 12 | 2 | Seconds. 0 to 59 as two ASCII characters |
| 14 | 2 | Day. 1 to 31 as two ASCII characters |
| 16 | 2 | Month. 1 to 12 as two ASCII characters |
| 18 | 4 | Year. 1980 to 2099 as four ASCII characters |

## Set system date and time (SD function)

This function sets the date and time maintained by the operating system in the workstation that has the system manager server installed. If automatic synchronization was selected during customization, the date and time are updated in all workstations in the LANDP workgroup.

Date and time can be supplied in binary or in character format. It is possible to update the system time only.

On LANDP for Linux, the SD function is not supported, and by default a X'P1' error code is returned. To aid the migration of existing applications, the system manager server can be configured to return a user-specified code for the SD function on Linux. This error code is defined through the loader statement for the system manager server.

| CPRB Field | Content/Description |
|---|---|
| Function code | SD |
| Request DATA length | **3** Time in binary format<br>**6** Time in character format<br>**7** Date and time in binary format<br>**14** Date and time in character format |
| Request PARMLIST length | 2 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | **'B'** Binary format<br>**'C'** Character format (default) |
| 1 | 1 | **'T'** Update time only<br>**'D'** Update date and time (default) |

| Request DATA Values (Binary Option) | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Hours. 0 to 23 |
| 1 | 1 | Minutes. 0 to 59 |
| 2 | 1 | Seconds. 0 to 59 |
| 3 | 1 | Day. 1 to 31 |
| 4 | 1 | Month. 1 to 12 |
| 5 | 2 | Year. 1980 to 2099, in word-reversed mode |

| Request DATA Values (Character Option) | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Hours. 0 to 23 |
| 2 | 2 | Minutes. 0 to 59 |
| 4 | 2 | Seconds. 0 to 59 |
| 6 | 2 | Day. 1 to 31 |
| 8 | 2 | Month. 1 to 12 |
| 10 | 4 | Year. 1980 to 2099 |

## LANDP workgroup system status and common data maintenance

In this section you can find the programming information required to write application programs or your own servers that issue service requests to the system manager server to maintain data common to all the workstations in a LANDP workgroup.

The functions available and the authorization level required for using them are:

| **Function** | **System authorization level** | | | |
|---|---|---|---|---|
| | **O** | **A** | **M** | **R, S, F, or none** |
| Get system status (GG) | Yes | Yes | Yes | Yes |
| Retrieve LANDP workgroup common data (RD) | Yes | Yes | Yes | Yes |
| Update alerts status (UA) | Yes | No | Yes | No |
| Update LANDP workgroup common data (UD) | Yes | Yes | Yes | Yes |
| Update LANDP workgroup identification (UI) | Yes | No | Yes | No |
| Update logging status (UL) | Yes | No | Yes | No |
| Update operator messages status (UM) | Yes | No | Yes | No |
| Update message operator receiver (UO) | Yes | No | Yes | No |
| **Note:** No authorization level is needed for using the GG, RD, and UD functions. | | | | |

### Get system status (GG function)

This function retrieves the system status data.

| **CPRB Field** | **Content/Description** |
|---|---|
| Function code | GG |
| Request DATA length | 0 |

| CPRB Field | Content/Description |
|---|---|
| Request PARMLIST length | 26 |
| Reply DATA length | Up to 24 |
| Reply PARMLIST length | 26 |
| Replied DATA length | Up to 24 |
| Replied PARMLIST length | 0 |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Current LANDP workgroup ID. Eight ASCII characters |
| 8 | 1 | Status of the LOG:<br>**'A'** Active<br>**'D'** Deactivated<br>**'N'** Not defined |
| 9 | 1 | LOG file in wraparound mode:<br>**'Y'** In wraparound mode<br>**'N'** Not in wraparound or not defined |
| 10 | 1 | Status of the alerts support:<br>**'A'** Active<br>**'D'** Deactivated<br>**'N'** Not defined |
| 11 | 1 | File space for alerts:<br>**'Y'** File full<br>**'N'** File not full or alerts support not defined |
| 12 | 1 | Status of the PU-SSCP session for sending alerts:<br>**'Y'** In session<br>**'N'** Not in session or alerts support not defined |
| 13 | 1 | Status of the operator messages support:<br>**'A'** Active<br>**'D'** Deactivated<br>**'N'** Not defined |
| 14 | 1 | Automatic synchronization of date and time:<br>**'Y'** Automatic mode<br>**'N'** Not automatic mode |
| 15 | 1 | System date and time set after IPL:<br>**'Y'** System date and time have been set<br>**'N'** System date and time have not been set |
| 16 | 8 | User ID that receives operator messages |

**Note:** If Reply DATA length is insufficient to store the global system data, the information returned is truncated and a nonzero return code is set.

## Retrieve LANDP workgroup common data (RD function)

This function retrieves the LANDP workgroup common data. If the retrieved data is to be modified using the function UD, the parameter 'U' (retrieve for update) must be used. If not, the function is not accepted.

The LANDP workgroup common data can be used as and when required. The common data could, for example, contain:

- Branch office or department ID
- Name and address of the branch office

If LAN common data validation is selected during customization, this data must follow the structure of the FBSSGLUS record, which has been defined during customization.

| CPRB Field | Content/Description |
|---|---|
| Function code | RD |
| Request DATA length | 0 |
| Request PARMLIST length | 1 |
| Reply DATA length | 1 to 1024 |
| Reply PARMLIST length | 26 |
| Replied DATA length | Up to 1024 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | This field contains the type of retrieve:<br>**'U'** Retrieve for update<br>**' '** Normal retrieve (default) |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 to 1024 | Global user data (X'0001' to X'0400') |

## Update alerts status (UA function)

This function updates the status of the alerts process (see “Alerts management” on page 498). If alerts support was not selected during customization, this function is ignored and an error code is returned. If it is accepted, the new status is valid until the next UA function.

When the alerts support is deactivated, the AN (page 499), MO (page 502), and UN (page 503) functions are ignored and an error code is returned to the requester.

| CPRB Field | Content/Description |
|---|---|
| Function code | UA |
| Request DATA length | 0 |
| Request PARMLIST length | 1 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | New status for alerts process:<br>**'D'** Deactivate<br>**'A'** Activate (default) |

## Update LANDP workgroup common data (UD function)

This function updates the LANDP workgroup common data retrieved by a previous RD function with the parameter *retrieve for update*. The initial content of the global user data is defined during customization.

The contents of Request DATA are checked against the record structure defined during customization, FBSSGLUS, if it was selected.

| CPRB Field | Content/Description |
|---|---|
| Function code | UD |
| Request DATA length | 1 to 1024 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |
| **Note:** If Request DATA length is not X'0000' and the UD function returns a nonzero return code, the condition *update pending* is not reset.<br><br>If Request DATA length is X'0000', no update is done, but any condition *update pending*, set after an RD function, is removed. | |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 to 1024 | Global user data (X'0001' to X'0400') |

## Update LANDP workgroup identification (UI function)

This function updates the LANDP workgroup ID. This ID is the name assigned to a specific LANDP workgroup during customization. It is, if required, included in each alerts message.

The LANDP workgroup ID is used as the Service Point and identifies the LANDP workgroup in the error logs and network management alerts in the host. To easily isolate and recognize where the alerts come from, it is advisable to identify the service point with the LANDP SNA PU name used in the host configuration. This name should be unique in the complete network definition to avoid confusion. A LANDP workgroup ID must have from four to eight characters. Valid characters for the LANDP workgroup ID are:

- English letters: A to Z
- English letters: a to z (are translated to corresponding uppercase letters)
- Digits: 0 to 9
- Special characters: @, #, $, and %

| CPRB Field | Content/Description |
|---|---|
| Function code | UI |
| Request DATA length | 4 to 8 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | New LANDP workgroup ID (4 to 8 characters) |

## Update logging status (UL function)

This function updates the status of logging support. The new status is valid until the next UL function. If logging support was not selected during customization, this function is ignored and an error code is returned.

If logging support is deactivated, the message operator support is also deactivated. When the logging support is started, the message operator support is also started. However, if the message operator support is deactivated by the UM function while the logging support is off, it is necessary to start the message operator support by requesting another UM function.

When the logging support is deactivated, the WL function (page 510) is ignored and an error code is returned to the requester.

| CPRB Field | Content/Description |
|---|---|
| Function code | UL |
| Request DATA length | 0 |
| Request PARMLIST length | 1 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | New status for logging support:<br>**'D'** Deactivate<br>**'A'** Activate (default) |

## Update operator messages status (UM function)

This function updates the operator messages status. The new status is valid until the next UM function. If logging support was not selected during customization, this function is ignored and an error code is returned.

When the logging support is deactivated, all WL function requests with the *operator message type* are ignored and an error code is returned to the client.

| CPRB Field | Content/Description |
|---|---|
| Function code | UM |
| Request DATA length | 0 |
| Request PARMLIST length | 1 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | New status for logging operator message:<br>**'D'** Deactivate<br>**'A'** Activate (default) |

## Update message operator receiver (UO function)

This function updates the user ID that receives the operator messages. The new user ID is valid until the next UO function is requested.

If logging support for operator messages was not selected during customization, this function is ignored and an error code is returned.

The user ID currently receiving the operator messages is the only one capable of performing the UO function.

| CPRB Field | Content/Description |
|---|---|
| Function code | UO |
| Request DATA length | 4 to 8 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | New user ID that receives operator messages |

## Retrieval of defined record structures

In this section you can find the programming information required to write application programs or your own servers that issue service requests to the system manager server to retrieve records defined with the record definition facility.

The function available and the authorization level required for using it described in the following table:

| Function | System authorization level | | | |
|---|---|---|---|---|
| | **O** | **A** | **M** | **R, S, F, or none** |
| Retrieve record format (RR) | Yes | Yes | Yes | Yes |
| **Note:** No authorization level is needed for using the RR function. | | | | |

## Retrieve record format (RR function)

This function retrieves the structure of a record as it was defined during customization.

A parameter allows you to choose whether the description of the fields is included.

| CPRB Field | Content/Description |
|---|---|
| Function code | RR |
| Request DATA length | 1 to 8 |

| CPRB Field | Content/Description |
|---|---|
| Request PARMLIST length | 1 |
| Reply DATA length | Up to 4096 |
| Reply PARMLIST length | 26 |
| Replied DATA length | Up to 4096 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Requested format of reply:<br>**'D'** With description of the fields<br>**' '** Only the format attributes (default) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Record format name (1 to 8 characters) |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | Up to 4096 | Record format (up to X'1000') |

If Reply DATA length is less than the record format length, no data is passed to the application program and a nonzero return code is set.

## Reply DATA values — without field description

The following response is received when a record format is requested by the system manager server without the field description:

| Offset | Length | Content | |
|---|---|---|---|
| 0 | 2 | Number of fields (1 to 92 in binary) | Record data |
| 2 | 1 | Format type (see note “Retrieve record format (RR function)” on page 493r) | |
| 3 | 1 | Delimiter: /, \, $, %, &, or @. For DBCS and mixed, SBCS and DBCS, fields, characters \ and @ are not supported as delimiters. | |
| 4 | 1 | Decimal separator character: '.' or ',' | |
| 5 | 1 | Blank | |

| Offset | Length | Content | |
|---|---|---|---|
| 6 | 8 | Field name | First field |
| 14 | 2 | Field length (see note 2 on page 496) | |
| 16 | 1 | Store format (see note 3 on page 496) | |
| 17 | 1 | Numeric sign (1=signed, 2=unsigned) | |
| 18 | 1 | Number of decimal places (see note 4 on page 496) | |
| 19 | 1 | Blank | |
| 20 | 8 | Field name | Second field |
| 28 | 2 | Field length | |
| 30 | 1 | Store format | |
| 31 | 1 | Numeric sign | |
| 32 | 1 | Number of decimal places | |
| 33 | 1 | Blank | |
| And so on, to a maximum of 92 fields | | | |

## Reply DATA values — with field description

The following response is received when a record format is retrieved by the system manager server with the field description:

| Offset | Length | Content | |
|---|---|---|---|
| 0 | 2 | Number of fields (1 to 92 in binary) | Record data |
| 2 | 1 | Format type (see note 1 on page 496) | |
| 3 | 1 | Delimiter: /, \, $, %, &, or @. For DBCS and mixed, SBCS and DBCS, fields, characters \ and @ are not supported as delimiters. | |
| 4 | 1 | Decimal separator character: '.' or ',' | |
| 5 | 1 | Blank | |
| 6 | 8 | Field name | First field |
| 14 | 2 | Field length (see note 2 on page 496) | |
| 16 | 1 | Store format (see note 3 on page 496) | |
| 17 | 1 | Numeric sign (1=signed, 2=unsigned) | |
| 18 | 1 | Number of decimal places (see note 4 on page 496) | |
| 19 | 1 | Blank | |
| 20 | 30 | Field description | |

| Offset | Length | Content | |
|---|---|---|---|
| 50 | 8 | Field name | Second field |
| 58 | 2 | Field length | |
| 60 | 1 | Store format | |
| 61 | 1 | Numeric sign | |
| 62 | 1 | Number of decimal places | |
| 63 | 1 | Blank | |
| 64 | 30 | Field description | |
| And so on, to a maximum of 92 fields | | | |

**Notes:**

1. The format type is:
   - **'D'** At least one field in the record is of variable length (using the delimiter to end the field)
   - **'F'** All fields are of fixed length
2. A field length of 0 means that the field is of variable length.

   If store format is not 'P', a field length of 1 to 4096 is the actual length of the field. It includes one sign byte, if signed. No space is included for the decimal separator character, because it is not stored.

   For store format 'P', the field length shown is the length of the field when displayed, including any sign.
3. The store formats are:

   - **'C'** Character. All bytes must be greater than X'19'.
   - **'N'** ASCII numeric. All bytes must be between X'30' and X'39'. If signed, the rightmost byte can also assume values from X'B0' to X'B9'
   - **'B'** Inverted binary, always signed.
   - **'I'** Short integer, signed or unsigned.
   - **'J'** Long integer, signed or unsigned.
   - **'P'** Packed (COBOL) (signed or unsigned). All half-bytes must be between 0 and 9, except the rightmost one being reserved for the sign: F, C, or D
   - **'H'** Hexadecimal, unsigned only
   - **'D'** DBCS character. All characters must be DBCS.
   - **'M'** Mixed SBCS and DBCS character. Both character types are allowed.
4. Number of decimal places required for store formats 'N' and 'P'. It can at most be equal to the field length.
5. All records start at a boundary of 16 bytes. Therefore, after the last record an additional filler (blanks) may appear.

## Data validation with defined record structures

This section gives you the programming information required to write application programs or your own servers that issue service requests to the system manager server to validate data using the defined record structures.

The function available and the authorization level required for using it are:

| Function | System authorization level | | | |
|---|---|---|---|---|
| | O | A | M | R, S, F, or none |
| Validate record (VR) | Yes | Yes | Yes | Yes |
| **Note:** No authorization level is needed for using the VR function. | | | | |

## Validate record (VR function)

This function validates data using the record structure defined with the record definition facility (RECDEF.vector) during customization. If any error is detected, the incorrect field is returned.

The record definition used in the validation must be stated in the parameter area.

| CPRB Field | Content/Description |
|---|---|
| Function code | VR |
| Request DATA length | 1 to 4096 |
| Request PARMLIST length | 1 to 8 |
| Reply DATA length | Up to 8 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 8 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Record format name (1 to 8 characters) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 to 4096 | Data to be validated |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Incorrect field (1 to 8 characters) |

## Alerts management

The system manager server provides three functions for problem notification using the generic alerts architecture. The NetView licensed program in the host acts as the alert receiver. Every LANDP software component and application in the LANDP workgroup can provide useful problem notification to the network operator, in the same way that SNA nodes do.

An *alert* is an unsolicited record sent to the network operator, using network management vector transport (NMVT), by a network component that has detected a problem. The alert provides the following information:

- Identity of the alerts sender
- Problem severity indicator and problem description
- List of probable causes — user causes, install causes, and failure causes, if any
- List of recommended actions for the operator

A *resolution* is an unsolicited record sent to the network operator, using an NMVT, by a network component that has detected that a problem has been resolved. The resolution provides the following information:

- Identity of the resolution sender
- Resolution type and description
- List of actual causes — user causes, install causes, and failure causes, if any
- List of actual actions performed

Using the SNA server, the system manager server forwards the network management data (the NMVT) to the Management Services focal point in the network. This focal point provides centralized network management support for all workstations in a LANDP workgroup, and uses the SSCP-PU session established with the NetView program in the host.

The physical unit (PU) used for sending alerts is defined during customization. If required, the alerts are queued for delayed transmission. The alerts can be generated by:

- The supervisor
- LANDP servers
- Servers you have developed
- Applications

## Generation and transmission of alerts and resolutions

The design for communications and system management in the LANDP family is based upon the following principles:

- Local management services (LMS) functions are implemented within each server that can generate alerts. That is, the server or an application builds the NMVT, with the exception of some common subvectors, and sends the request to the system manager server.
- Most of the physical unit management services (PUMS) functions are implemented as a part of the system manager server, which completes the NMVT and requests transmission to the host by calling the SNA server. The system manager server also manages the queueing, and transmission of held and delayed alerts.

Elements of the NMVT that can be provided by the system manager server for all alerts are:
  - NMVT MS RU header (X'41038D')
  - Date and time MS common subvector (X'01')
  - Product ID MS common subvector (X'10')
  - Hierarchy resource list ID MS common subvector (X'05')
- Actual transmission of NMVTs in the SSCP-PU session is performed by the SNA server.

Alerts can be passed to the system manager server as subvectors or as complete NMVTs (with header) and are forwarded to the SNA server. If the NMVTs are complete, the system manager server forwards them without any modification. For more information, refer to the *IBM SNA Formats* book.

Held and delayed alerts, resolutions, and messages for the NetView operator are maintained in the file FBSS#ALR, to be forwarded when the session with the focal point becomes available. The drive and the path for this file are defined during customization.

In this section you can find the programming information required to write application programs or your own servers that issue service requests to the system manager server for problem notification using the generic alerts architecture.

The functions available and the authorization level required for using them, are:

| Function | System authorization level | | | |
|---|---|---|---|---|
| | O | A | M | R, S, F, or none |
| Alerts notification (AN) | Yes | Yes | Yes | Yes |
| Send message to NetView operator (MO) | Yes | Yes | Yes | Yes |
| Resolution notification (UN) | Yes | Yes | Yes | Yes |
| **Note:** No authorization level is needed for using the AN, MO, and UN functions. | | | | |

## Alerts notification (AN function)

This function is used by servers and application programs to send an alert to the system manager server. No authorization level is required when using this function. The server or application program can send the complete NMVT (network management vector transport) or only the sub-vectors. The system manager server assumes that the complete NMVT is provided if the first three bytes in Request DATA are X'41038D', if not the system manager server adds the NMVT header, the major vector length and the major vector key.

Depending on request parameters, the system manager server may include the date and time management services (MS) common sub-vector and the product ID MS common sub-vector.

Depending also on request parameters, the system manager server can issue a generic alert from the data supplied in the Request DATA area or from data contained in EHC#ALRU.DAT, a file generated by the user to define user alerts.

When data from EHC#ALRU.DAT is used, the run-time variable fields are passed to the system manager in the Request DATA area, if they are required for the specific alert.

When using the sub-vector X'05' (hierarchy):

- The originator of the alert must include its own hierarchy, including at least the workstation ID, in ASCII, and, optionally, the device identifier in EBCDIC.
- The system manager server completes the sub-vector with the LANDP workgroup ID as the service point, with LANDP in EBCDIC as the transaction program, and with the complete indicator set to X'00'.
- The LANDP workgroup ID defined during customization is used as the service point and identifies the LANDP workgroup in the error logs and network management alerts in the host. To ease the task of isolating and recognizing where the alerts come from, it is advisable to identify the service point with the LANDP SNA PU name used in the host configuration. This name should be unique in the complete network definition to avoid confusion.

To ensure forward compatibility and data integrity, the requester should send text messages in one of the following EBCDIC coded character sets:

- Coded graphic character set 00640-00500
- Coded graphic character set 00640-00500 Extended
- Coded graphic character set 01134-00500

| CPRB Field | Content/Description |
|---|---|
| Function code | AN |
| Request DATA length | NMVT length, or sum of the length of the sub-vectors, or sum of lengths of variable fields |
| Request PARMLIST length | 2 to 6 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

If the Request PARMLIST length is 2, the NMVT is given in Request DATA. If the Request PARMLIST length is 6, vectors used are taken from the file EHC#ALRU.DAT. Here Request PARMLIST identifies the generic alert in that file, and Request DATA contains variable fields for substitution into the generic alert.

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | **'Y'** MS common sub-vector X'01' (date and time) is included by the system manager server |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 1 | 1 | **'Y'** MS common sub-vector X'10' (Product ID) is included by the system manager server |
| And if length is 6: | | |
| 2 | 4 | NMVT identification number ('2000' to 'ZZZZ') |

| Request DATA Values if Request PARMLIST length is 2 | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Complete NMVT (up to 512 bytes). If you want the system manager server to add the header and one or both sub-vectors, X'01' and X'10', you must subtract the following values from the maximum length (512 bytes):<br>**header length** 12<br>**sub-vector X'01' length** 10<br>**sub-vector X'10' length** 29<br>Sub-vectors only (up to 500 bytes) |

| Request DATA Values if Request PARMLIST length is 6 | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | NMVT supplied variable fields. The format must match the offsets, lengths, and types of the variable fields defined in EHC#ALRU.DAT. |

**Note:** NMVT identification numbers from '0001' to '1ZZZ' are reserved for use by LANDP. Those used are stored in the files EHC#ALRN.DAT (for alert notifications) and EHC#RESN.DAT (for resolution notifications), provided by LANDP.

## Format of EHC#ALRU.DAT and EHC#RESU.DAT files

You should create these two files to contain any generic alerts and resolutions that you want to define.

**Note:** On LANDP for Linux, these filenames are in lowercase. However, the file identifier in the following table is in uppercase.

The two files contain the generic alert and resolution NMVTs. Each file has a header followed by several records, as follows:

| File Header Format | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | File identifier (EHC#ALRU or EHC#RESU) |
| 8 | 4 | Number of records (0000 to 9999) |

| File Header Format | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 12 | 52 | Reserved |

| Record Format | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 4 | Record number ('2000' to 'ZZZZ') |
| 4 | 30 | Description |
| 34 | 12 | Reserved |
| 46 | 2 | Variable field 1: offset (binary) |
| 48 | 2 | Variable field 1: length (binary) |
| 50 | 1 | Variable field 1: type ('A': ASCII, 'H': hexadecimal) |
| And so on for further variable fields until... | | |
| 81 | 2 | Variable field 8: offset (binary) |
| 83 | 2 | Variable field 8: length (binary) |
| 85 | 1 | Variable field 8: type ('A': ASCII, 'H': hexadecimal) |
| 86 | 2 | Complete NMVT length, or sum of the subvector lengths |
| 88 | | Complete NMVT (up to 512 bytes) or subvectors (up to 500 bytes) |

The data supplied in Request DATA is placed into the NMVT, according to the lengths specified in the variable field lengths, at the specified offsets. If the length is zero, no data is placed into the NMVT.

If the variable field type is ASCII, the system manager server translates the data to EBCDIC. If not, it copies the data unchanged.

## Send message to NetView operator (MO function)

This function is used by servers and clients to send a message to the system manager server, which forwards it to NetView. Double-byte character sets are supported. The Request DATA area must not contain SI/SO characters.

| CPRB Field | Content/Description |
|---|---|
| Function code | MO |
| Request DATA length | 0 to 140 |
| Request PARMLIST length | 6 to 10 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Reserved |
| 2 | 4 to 8 | Destination NetView operator ID |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | Up to 140 | Message to send (in ASCII characters) |

## Resolution notification (UN function)

When a problem that was signalled through the alert notification (AN) function has been resolved, a further notification should be sent to the same receiver. This is a resolution notification (an “unalert” condition), and uses exactly the same mechanism as the previous alert notification, except that the function requested is UN and the generic file that contains the resolution major vectors is called EHC#RESU.DAT instead of EHC#ALRU.DAT.

See “Alerts notification (AN function)” on page 499 for a description of the AN function.

## EHCALRH user exit

By means of a user exit routine the system manager server filters generic alerts and resolutions received through the AN and UN functions.

When an alert or resolution is received, the system manager server issues a Check NMVT (CN) function to a user server named EHCALRH (Alert Handler) (default) or as specified at customization. If this server is not defined or not loaded, the NMVT is forwarded to NetView. If EHCALRH is loaded, it can modify the NMVT and also decide whether to forward the NMVT to NetView, depending on rules chosen by the user. On return from the CN function, the Reply DATA area contains the NMVT to be forwarded to NetView. If the Replied DATA length is zero, no alert is forwarded.

## Check NMVT (CN function)

The system manager server issues a CN function request to the **EHCALRH** server of the following form:

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | CN | Function code |
| 14 | 2 | 0 | Request PARMLIST length |
| 20 | 2 | NMVT length | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 26 | 2 | 0 | Reply PARMLIST length |
| 32 | 2 | 512 | Reply DATA length |

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 34 | 4 | Address | Reply DATA address |
| 50 | 8 | Reserved | Resource origin |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | EHCALRH | Server name |

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 44 | 2 | 0 | Replied PARMLIST length |
| 46 | 2 | X'0000' (if NMVT is to be blocked), or NMVT length (if NMVT is to be forwarded) | Replied DATA length |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | Up to 512 | Complete NMVT |

| Reply DATA Values (if Replied DATA length>0) | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | Up to 512 | Complete NMVT to be forwarded |

## System and user LOG management

In this section you can find the programming information required to write application programs or your own servers that issue service requests to the system manager server to store and retrieve the system and user LOG.

The functions available and the authorization level required to use them are:

| Function | System authorization level | | | |
|---|---|---|---|---|
| | O | A | M | R, S, F, or none |
| Retrieve log record (RL) | Yes | No | Yes | No (note 1) |
| Retrieve log record (Year-2000) (R1) | Yes | No | Yes | No (note 1) |
| Write log record (WL) | Yes | Yes | Yes | Yes |

**Notes:**
1. The RL can also be issued by users having authorization level S.
2. No authorization level is needed for using the WL function.

## LOG file header

The LOG file contains as the first record the following header:

| Offset | Length | Content |
|---|---|---|
| 0 | 8 | File Identifier: **FBSS#LOG** |
| 8 | 8 | Number of records |
| 16 | 4 | Record length |
| 20 | 8 | Write pointer |
| 28 | 8 | Read pointer |
| 36 | 4 | Number of next record to write |
| 40 | 4 | Number of next record to read |
| 44 | 84 | Reserved |

**Note:** All values are ASCII characters.

## LOG record format

The record format in the LOG file is:

| Offset | Length | Content |
|---|---|---|
| 0 | 4 | LOG record sequence number, see note 1 on page 506 and note 3 on page 506 |
| 4 | 4 | Actual length of this record |
| 8 | 6 | System date (yymmdd), see note 1 on page 506 |
| 14 | 6 | System time (hhmmss), see note 1 on page 506 |

| Offset | Length | Content |
|---|---|---|
| 20 | 3 | Device identifier of logging application program or server. Blanks or numeric values |
| 23 | 2 | Identifier of logging workstation, see note 1 |
| 25 | 1 | Bucket, see note 4:<br>**'A' to 'E'**<br>Reserved<br>**'F'** LANDP servers<br>**Other** Identifier you have defined |
| 26 | 4 | Message number for error message. Blanks or numeric values |
| 30 | 9 | Product identifier, see note 2 |
| 39 | 3 | Product release indicator, see note 2 |
| 42 | 8 | Requester Identifier. Name of the logging server. Blanks for application program |
| 50 | 2 | Event. Unique event number in each server (place of detection). This identifies where this entry was logged by relating it to the server name. Blanks or numeric values |
| 52 | 8 | Return Code. Completion code provided by the server using the WL function |
| 60 | 1 | Message type:<br>**'E'** Error<br>**'I'** Information<br>**'O'** Operator, see note 5<br>**other** Identifier you have defined |
| 61 | 3 | Reserved |
| **Note:** All values in the above fields are ASCII characters. | | |
| 64 | 0 to 960 | Data supplied by logging server or application program. It could be control block, module name, parameters passed, and so on, see note 6. |

**Notes:**

1. Entry inserted by the system manager server.
2. Entry inserted by the system manager server if not specified in the WL function.
3. Sequence numbers increase from 0001 to 9999 and then restart at 0001. The value 0000 is not used.
4. Bucket is effectively a prefix to the error number allowing use of the full range of error numbers (0001 to 9999).
5. Operator message type. The system manager operator informs through audio and video signals that there is a message for the operator that should be displayed.
6. The header length must be subtracted from the maximum record length defined during customization. The result of this operation gives you the maximum data length available for logging servers or application programs.

## Error number and message tables

A formatted message table for each bucket is defined to store a table of error numbers and their associated messages. The name of each table is LOGx.MSG, where x is the bucket identifier. For example, the table LOGF.MSG, supplied with a LANDP component, contains the error numbers and the message text for LANDP bucket F.

During the execution of an RL function, if the record header contains a message number and a bucket, the system manager server looks for the appropriate error number table and, if found, adds the contents of the message number entry at the end of the user defined area.

You may build your own tables, one for each bucket you have defined.

The format of each record in LANDP for DOS is shown in the following table. The error numbers need not be in sequence.

| Offset | Length | Content |
|---|---|---|
| 0 | 4 | Error number. Four ASCII characters (X'30' to X'39') |
| 4 | 2 | Delimiter. Two blanks |
| 6 | Up to 248 | Error message text. Up to 248 ASCII characters |
| | 2 | End of record. LF (line feed) and CR (carriage return) |

You can use any text editor to build your own message tables in LANDP for DOS.

To build your own message tables in LANDP for OS/2 you can use the OS/2 text message file format, which is described in the *OS/2 Warp Control Program Programming Guide, G25H-7101*.

On LANDP for Linux, the message tables are generated by the use of *gencat*. In other words, they are standard Linux catalogs. Refer to your Linux LANDP for Linux for information about the use of *gencat*.

The format of the input to *gencat* is:

```
LOGxxxx: 'message'
```

where xxxx is the error number, consisting of 4 ASCII characters in the range X'30' through X'39'.

## Retrieve LOG record (RL function)

This function retrieves a record from the LOG file using the record sequence number and, optionally, selection criteria for:

- Workstation ID
- Bucket
- Requester
- Message type
- Device identifier
- Date

If a blank is inserted for a selection criteria, it means *any*. If more than one selection criteria is used the AND rules apply. This function allows you to retrieve:

- Oldest record satisfying selection criteria
- Oldest record without using selection criteria
- Most recent record satisfying selection criteria
- Most recent record without using selection criteria
- Record **nnnn**
- Record preceding **nnnn** satisfying selection criteria
- Record preceding **nnnn** without using selection criteria
- Record following **nnnn** satisfying selection criteria
- Record following **nnnn** without using selection criteria

| CPRB Field | Content/Description |
|---|---|
| Function code | RL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | Up to 1024 |
| Reply PARMLIST length | 26 |
| Replied DATA length | Up to 1024 |
| Replied PARMLIST length | 8 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 4 | LOG record sequence number:<br>**'FFFF'** Get the newest record<br>**'0000'** Get the oldest record<br>**'nnnn'** Get the nnnn record |
| 4 | 1 | Qualifier (only valid if previous field is **nnnn**):<br>**'+'** Get the record following **nnnn**<br>**'–'** Get the record before **nnnn**<br>**other** Get the **nnnn** record |
| 5 | 2 | Workstation ID selection criteria |
| 7 | 1 | Bucket selection criteria |
| 8 | 8 | Server selection criteria. # can be used to mean any valid character |
| 16 | 1 | Message type selection criteria |
| 17 | 3 | Device identifier selection criteria |
| 20 | 6 | Date selection criteria (yymmdd) |
| **Note:** All values are ASCII characters. | | |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 4 | LOG sequence number of retrieved record |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 4 | 4 | Logical number of retrieved record (relative to the oldest one). 0000 means the oldest record |
| **Note:** All values are ASCII characters. | | |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Record retrieved (up to 1024 bytes), see “LOG record format” on page 505 |

## Retrieve LOG record (R1 function)

This function retrieves a record from the LOG file using the record sequence number and, optionally, selection criteria for:

- Workstation ID
- Bucket
- Requester
- Message type
- Device identifier
- Date

**Note:** The R1 command is similar to the RL command except that the date specified in the Request PARMLIST date field is in the form yyyymmdd instead of yymmdd. The LOG record returned contains the century bytes in ASCII character format at offsets 61 and 62, for example, the century may be ‘19’ or ‘20’.

If a blank is inserted for a selection criteria, it means *any*. If more than one selection criteria is used the AND rules apply. This function allows you to retrieve:

- Oldest record satisfying selection criteria
- Oldest record without using selection criteria
- Most recent record satisfying selection criteria
- Most recent record without using selection criteria
- Record **nnnn**
- Record preceding **nnnn** satisfying selection criteria
- Record preceding **nnnn** without using selection criteria
- Record following **nnnn** satisfying selection criteria
- Record following **nnnn** without using selection criteria

| CPRB Field | Content/Description |
|---|---|
| Function code | R1 |
| Request DATA length | 0 |
| Request PARMLIST length | 28 |
| Reply DATA length | Up to 1024 |
| Reply PARMLIST length | 28 |

| CPRB Field | Content/Description |
|---|---|
| Replied DATA length | Up to 1024 |
| Replied PARMLIST length | 8 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 4 | LOG record sequence number:<br>**'FFFF'** Get the newest record<br>**'0000'** Get the oldest record<br>**'nnnn'** Get the nnnn record |
| 4 | 1 | Qualifier (only valid if previous field is **nnnn**):<br>**'+'** Get the record following **nnnn**<br>**'–'** Get the record before **nnnn**<br>**other** Get the **nnnn** record |
| 5 | 2 | Workstation ID selection criteria |
| 7 | 1 | Bucket selection criteria |
| 8 | 8 | Server selection criteria. # can be used to mean any valid character |
| 16 | 1 | Message type selection criteria |
| 17 | 3 | Device identifier selection criteria |
| 20 | 8 | Date selection criteria (yyyymmdd) |
| **Note:** All values are ASCII characters. | | |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 4 | LOG sequence number of retrieved record |
| 4 | 4 | Logical number of retrieved record (relative to the oldest one). 0000 means the oldest record |
| **Note:** All values are ASCII characters. | | |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Record retrieved (up to 1024 bytes), see “LOG record format” on page 505 |

## Write LOG record (WL function)

This function includes a record in the LOG file. Records are added sequentially. If the LOG file is full, the oldest record is replaced in a wraparound mode.

| CPRB Field | Content/Description |
|---|---|
| Function code | WL |
| Request DATA length | Up to 960 |
| Request PARMLIST length | 27 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Resource origin | Requester identifier |
| Replied PARMLIST length | 4 |

**Notes:**

1. If Request DATA length is less than the actual record length (or greater than 960 bytes), the information is truncated and a nonzero return code is set.
2. If Request DATA length is greater than the actual record length, the record is padded with blanks.

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Event |
| 2 | 1 | Bucket |
| 3 | 4 | Return code |
| 7 | 1 | Message type |
| 8 | 4 | Message number |
| 12 | 3 | Device identifier |
| 15 | 9 | Product identifier<br><br>If left blank, the product identifier is LANDP, and release is also inserted. |
| 24 | 3 | Product release indicator |
| **Note:** All values are ASCII characters. | | |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | Up to 960 | Data supplied by logging server or application |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 4 | LOG record sequence number of record just written. Four ASCII characters |

# Chapter 22. Local resource manager

The local resource manager allows **LANDP for DOS** application programs to have dialogs with the following LANDP for DOS components:

- 3270 Display/keyboard emulator (EMU3270) — see page 519, Chapter 28, “LANDP 3270 emulator API” on page 593, and Chapter 29, “LANDP 3270 emulator high-level language API” on page 613
- 3287 Printer emulator (EMU3287) — see page 521 and Chapter 30, “LANDP 3287 printer emulator API” on page 657
- Printer manager server (PRTMGR) — see page 541 and Chapter 11, “Printer manager server” on page 199

When IBM Financial Branch System Integrator is installed at the workstation, another emulator, a financial printer emulator, can be added to the previous list:

- Banking printer program (BPP) — see page 546

The local resource manager provides the same services as the operator interface (see the *LANDP Servers and System Management* book). The main difference between them however, is the way communication is established with the requesters. The operator interface functions are entered through a terminal and the output is displayed at the screen. For the local resource manager however, the requester is an application program (client) and the input and output are addressed to the same CPRB using the LANDP common application programming interface (API).

The following services are available using the local resource manager:

- Obtain communication and printer status
- Control communication with the host applications
- Control the printing of data
  - On up to three parallel attached printers
  - On up to four serially attached financial printers

**Note:** The client application must be installed in the same workstation as the server from which it is requesting service.

© Copyright IBM Corp. 1992, 2002

*Table 43. Function Codes used in the Local Resource Manager*

| Function code | Description | Env. | Page |
|---|---|---|---|
| **Functions used with the 3270 display/keyboard emulator:** | | | |
| EC | End connection | 0--- | 519 |
| GS | Get status | 0--- | 520 |
| **Functions used with the 3287 printer emulator:** | | | |
| AI | Activate application intervention | 0--- | 522 |
| AS | Activate service | 0--- | 523 |
| CA | Change printer assignment | 0--- | 524 |
| CC | Cancel pending functions | 0--- | 526 |
| CM | Change print density mode | 0--- | 528 |
| CP | Change LU priority | 0--- | 529 |
| EI | End operator intervention | 0--- | 531 |
| EL | End listing | 0--- | 533 |
| GS | Get status | 0--- | 534 |
| IS | Initiate session | 0--- | 536 |
| SS | Suspend service | 0--- | 537 |
| TS | Terminate session | 0--- | 539 |
| **Functions used with the printer manager server:** | | | |
| GS | Get status | 0--- | 541 |
| SA | Set alternate mode | 0--- | 542 |
| SE | Set exclusive mode | 0--- | 544 |
| **Functions used with banking printer program:** | | | |
| CC | Cancel pending functions | 0--- | 546 |
| GS | Get status | 0--- | 547 |
| HP | Change printer to remote mode | 0--- | 548 |
| LP | Change printer to local mode | 0--- | 549 |

The first and second columns give the function code and the name of the function. The third column shows the operating environment of the function. "0---" means that it is available from LANDP for DOS servers. The last column refers to the page where you can find the function described.

## Status codes

After function execution, the application program can get the various status values from Reply DATA, with a specific format.

The different status groups are arranged in separate fields. The application program can read each type of status at the appropriate offset. Nine status groups exist.

## Communication status

These status codes are used in *all* environments.

**C1** Host not connected
**C2** Host connected
**C3** In session
**C4** In session processing
**C5** Session ended automatically (X.25 communication)
**C6** Session lost
**C7** Session ended
**C8** Session force ended

These status codes inform the application program whether a physical connection with the host is available or not, and identify the session.

If a X.25 network is used for communications, the session can end automatically after a defined time. Status C5 is then returned.

## Pending communication function status

These status codes are used with the 3287 printer emulator (EMU3287).

**TS** TS function pending
**SS** SS function pending
**NC** No pending communication function

These status codes inform the application program that pending functions TS and SS are to be processed when the current listing finishes printing.

Only one function can be pending at any specific time.

## Printer attending mode status

These status codes are used with the *banking printer program* (BPP).

**P1** Remote
**P2** Remote force
**P3** Local
**P4** Local force
**P5** First request

Status codes P1 and P3 denote the status of the printer assigned. The control of the printer can be changed only by the local resource manager and by the operator interface. In remote mode the printer is used only by BPP and is not released for use by the application programs until the local mode is set. Status codes P2 and P4 denote the mode has been switched immediately, interrupting any current printing.

Status code P5 is issued before the BPP begins. The printer is waiting for the first request and must receive a set printer mode function from the local resource manager to be acquired and operate. If the first request comes from a workstation application program, the printer is set to local mode. If the first request comes from BPP, the printer is set to remote mode.

## Pending printer attending mode status

These status codes are used with the *banking printer program* (BPP).

**LP** LP function pending
**HP** HP function pending
**NP** No pending printer attending mode function

These status codes inform the application program that the functions LP or HP are to be processed after the current listing or transaction is finished.

Only one function can be pending at any one time.

## Printer activity mode status

These status codes are used with the 3287 printer emulator (EMU3287).

**S1** Suspend
**S2** Suspend force
**S3** Listing held
**NS** Normal printing activity mode

The status codes S1 and S2 show that the printer session has been suspended through the SS function. S3 status occurs when a printing request has been received when *application intervention* is active. This status is performed for the time that function AS is not executed or *application intervention* is not deactivated.

## Emulator attending status

These status codes are used with the 3287 printer emulator (EMU3287) and the *banking printer program* (BPP).

**P9** Attending this resource
**PA** Not attending this resource

P9 status shows that the printing request of the specified LU session is being attended to. For EMU3287 this means that this LU 1 session has priority 1. Sessions with priorities from 2 to 5 are in status PA. For BPP, P9 shows that the printer is operating in remote mode. When the printer is set to local mode by the operator, the BPP status is PA.

## Application intervention status

These status codes are used with the 3287 printer emulator (EMU3287).

**LA** Application intervention active
**NA** No application intervention

The LA status informs the application program that the AI function has been performed. The start of the print job is controlled by the application program for the time that this status is not deactivated by the EI function.

## End listing control status

These status codes are used with the 3287 printer emulator (EMU3287).

**LB** End of listing is under application program control
**LC** Waiting for the application program to end the listing
**NL** No expected end of listing from the application

The LB status occurs when the application program has sent a listing to be printed with the first line containing *LIST=nnnnnnnn*. This statement shows that the application program sends *ENDL=nnnnnnnn* as the last sentence. The listing can not be ended when this last sentence has not been received from the application program.

The LC status occurs when a listing is waiting for *ENDL=nnnnnnnn* sentence from the application program and an *end bracket* is received, or the normal end of listing timer elapses.

If the status code is NL, the listing ends normally, without application program intervention.

## Printer/Port status

These status codes are used with BPP and the printer manager.

**PB** Available
**PC** Adapter not installed
**PD** Printer not installed
**PE** Printer powered off
**PF** Printer not on line
**PG** Printer out of paper
**PH** Only known when used by this resource
**PI** Printer busy

The PB status shows that *everything* is available: adapter and printer installed, printer powered on and on line, printer with paper, printer not busy, and (for BPP) if the printer is being used by BPP.

The PH status is specially provided for the *banking printer program* (BPP). BPP uses the IBM financial printer server to manage the serial printer assigned. As the financial printer server can attend to requests from other application programs, BPP can request the port status only when the printer is in the remote state. If not in the remote state then PH status occurs.

## Using the local resource manager

This section provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields. If you need more information about the CPRB fields, see Appendix A, “Connectivity programming request block” on page 669.

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 26 | 2 | | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | EHCLRMGR | Server name |

The following fields are variable and are discussed in each function request description:
- Function code
- Request PARMLIST length
- Request DATA length
- Reply PARMLIST length
- Reply DATA length

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 44 | 2 | | Replied PARMLIST length |
| 46 | 2 | | Replied DATA length |

If the request was successful, the *router return code* and the *server return code* are both X'00000000'. In all other cases, see the appropriate section in the *LANDP Problem Determination* book to see if you should take any action. The return values in Reply PARMLIST and DATA should be ignored when an error occurs.

The following fields are variable and are discussed in each function request description:
- Replied PARMLIST length
- Replied DATA length

**Note:** The functions of the local resource manager return status information in Reply DATA in byte-reversed format. For example, the status C1 is returned as X'3143'.

## Functions relating to the IBM 3270 display/keyboard emulator

The emulator is described in Chapter 28, “LANDP 3270 emulator API” on page 593 and Chapter 29, “LANDP 3270 emulator high-level language API” on page 613. The local resource manager functions are listed in alphabetical order of function code.

## End connection (EC function)

This function ends the connection to the host computer. It is especially used in an X.25 Network. The EC function can be used when:

**Communication status** C2 or C3

**X25 Communication in OPE.CFG**
’1’ (yes)

| CPRB Field | Content/Description |
|---|---|
| Function code | EC |
| Request DATA length | 0 |
| Request PARMLIST length | 9 to 16 |
| Reply DATA length | 2 |
| Reply PARMLIST length | 9 |
| Replied DATA length | 2 |
| Replied PARMLIST length | 9 |

| Request PARMLIST Values when length is X'0009' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | ’EMU3270 ’ Service identification name (left-justified and padded with blanks) |
| 8 | 1 | ’1’, ’2’, ’3’, ’4’, or ’5’ Emulator LU 2 session number identifier |

| Request PARMLIST Values when length is X'0010' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | ’EMU3270 ’ Service identification name (left-justified and padded with blanks) |
| 8 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |
| 8 | 1 | ’1’, ’2’, ’3’, ’4’, or ’5’ Emulator LU 2 session number identifier |

In Reply DATA, the following communication status is returned. See “Status codes” on page 515 for an explanation.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | C1, C2, C3, .. or C8 Communication status |

## Get status (GS function)

This function obtains the current communication status of the LU 2 session assigned to a specified 3270 Display/Keyboard emulator. (Other uses of this function in the local resource manager are described on pages 534, 541, and 547.)

| CPRB Field | Content/Description |
|---|---|
| Function code | GS |
| Request DATA length | 0 |
| Request PARMLIST length | 9 to 16 |
| Reply DATA length | 2 |
| Reply PARMLIST length | 9 |
| Replied DATA length | 2 |
| Replied PARMLIST length | 9 |

| Request PARMLIST Values when length is X'0009' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3270 ' Service identification name (left-justified and padded with blanks) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 2 session number identifier |

| Request PARMLIST Values when length is X'0010' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3270 ' Service identification name (left-justified and padded with blanks) |
| 8 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 2 session number identifier |

In Reply DATA, the following communication status is returned. See "Status codes" on page 515 for an explanation.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | C1, C2, C3, .. or C8 Communication status |

## Functions relating to the IBM 3287 printer emulator

The emulator is described in Chapter 30, "LANDP 3287 printer emulator API" on page 657. The local resource manager functions are listed in alphabetical order of function code.

## Activate application intervention (AI function)

This function causes that the next printing request remains in the status S3 (*Listing Held*) until an AS function is requested to begin the listing. The AI function can be used when:

**Application intervention status** NA

**Pending communication function status**
NC

| CPRB Field | Content/Description |
|---|---|
| Function code | AI |
| Request DATA length | 0 |
| Request PARMLIST length | 9 to 16 |
| Reply DATA length | 12 |
| Reply PARMLIST length | 12 |
| Replied DATA length | 12 |
| Replied PARMLIST length | 12 |

| Request PARMLIST Values when length is X'0009' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |

| Request PARMLIST Values when length is X'0010' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |
| 9 | 1 | '1', '2', or '3' Printer number identifier |
| 10 | 1 | 'N', 'M', or 'C' Print density mode<br>'N' = 10 cpi, 'M' = 12 cpi, 'C' = 17 cpi |
| 11 | 1 | ' ' or '1', '2', '3', '4', or '5' Priority<br>(see the following Notes) |

**Notes:**

1. The priority number in Reply PARMLIST corresponds to the number assigned to a LU 1 session, and defines the order of service for print requests. Priority 1 corresponds to the LU 1 session currently in process.
2. If the adapter is not installed or the LU activity is suspended, then the priority field is returned with a blank which means the specified LU has *no* priority.

After processing is completed, the following 3287 emulator status codes are returned in Reply DATA. See “Status codes” on page 515 for an explanation of these codes.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | C1, C2, C3, .. or C8 Communication status |
| 2 | 2 | NC, TS, or SS Pending communication function status |
| 4 | 2 | P9 or PA Emulator attending status |
| 6 | 2 | NA or LA Application intervention status |
| 8 | 2 | NL, LB, or LC Listing control status |
| 10 | 2 | NS, S1, S2, or S3 Printing activity mode |

## Activate service (AS function)

This function reactivates the LU 1 session, after a previous SS or AI function has been executed.

The AS function can be used when:

**Printer activity mode** S1, S2, or S3

**Pending communication function status**
NC

| CPRB Field | Content/Description |
|---|---|
| Function code | AS |
| Request DATA length | 0 |
| Request PARMLIST length | 9 to 16 |
| Reply DATA length | 12 |
| Reply PARMLIST length | 12 |
| Replied DATA length | 12 |
| Replied PARMLIST length | 12 |

| Request PARMLIST Values when length is X'0009' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | ’EMU3287 ’ Service identification name (left-justified and padded with blanks) |

| Request PARMLIST Values when length is X'0009' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |

| Request PARMLIST Values when length is X'0010' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |
| 9 | 1 | '1', '2', or '3' Printer number identifier |
| 10 | 1 | 'N', 'M', or 'C' Print density mode<br>'N' = 10 cpi, 'M' = 12 cpi, 'C' = 17 cpi |
| 11 | 1 | ' ' or '1', '2', '3', '4', or '5' Priority<br>(see the following Notes) |

**Notes:**

1. The priority number in Reply PARMLIST corresponds to the number assigned to a LU 1 session, and defines the order of service for print requests. Priority 1 corresponds to the LU 1 session currently in process.
2. If the adapter is not installed or the LU activity is suspended, then the priority field is returned with a blank which means the specified LU has *no* priority.

After execution, the following 3287 emulator status codes are returned in Reply DATA. See "Status codes" on page 515 for an explanation of these codes.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | C1, C2, C3, .. or C8 Communication status |
| 2 | 2 | NC, TS, or SS Pending communication function status |
| 4 | 2 | P9 or PA Emulator attending status |
| 6 | 2 | NA or LA Application intervention status |
| 8 | 2 | NL, LB, or LC Listing control status |
| 10 | 2 | NS, S1, S2, or S3 Printing activity mode |

## Change printer assignment (CA function)

This function changes the printer assignment for the specified LU.

Specify the new printer assignment parameter in Request DATA. This parameter defines the new printer number. It can take a value from 1 to 3 but must be different from the current printer number and cannot exceed the number of printers available. The new printer must have the adapter installed.

The function mode P updates the printer assignment in the local resource manager configuration file. This file must be available when the function is requested.

The CA function can be used when:

**Priority**

Any, except 1.

**Pending communication function status**

NC.

| CPRB Field | Content/Description |
|---|---|
| Function code | CA |
| Request DATA length | 2 |
| Request PARMLIST length | 9 to 16 |
| Reply DATA length | 12 |
| Reply PARMLIST length | 12 |
| Replied DATA length | 12 |
| Replied PARMLIST length | 12 |

| Request PARMLIST Values when length is X'0009' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |

| Request PARMLIST Values when length is X'0010' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Function mode:<br>**'P'** save the change<br>**' '** do not save the change |
| 1 | 1 | '1', '2', or '3' New assigned printer number |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |
| 9 | 1 | '1', '2', or '3' Printer number identifier |
| 10 | 1 | 'N', 'M', or 'C' Print density mode<br>'N' = 10 cpi, 'M' = 12 cpi, 'C' = 17 cpi |
| 11 | 1 | ' ' or '1', '2', '3', '4', or '5' Priority<br>(see the following Notes) |

**Notes:**

1. The priority number in Reply PARMLIST corresponds to the number assigned to a LU 1 session, and defines the order of service for print requests. Priority 1 corresponds to the LU 1 session currently in process.
2. If the adapter is not installed or the LU activity is suspended, then the priority field is returned with a blank which means the specified LU has *no* priority.

After execution, the following 3287 emulator status codes are returned in Reply DATA. See "Status codes" on page 515 for an explanation of these codes.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | C1, C2, C3, .. or C8 Communication status |
| 2 | 2 | NC, TS, or SS Pending communication function status |
| 4 | 2 | P9 or PA Emulator attending status |
| 6 | 2 | NA or LA Application intervention status |
| 8 | 2 | NL, LB, or LC Listing control status |
| 10 | 2 | NS, S1, S2, or S3 Printing activity mode |

## Cancel pending functions (CC function)

This function cancels any functions that are pending.

The CC function can be used when:

**Pending communication function status**
TS or SS

| CPRB Field | Content/Description |
|---|---|
| Function code | CC |
| Request DATA length | 0 |
| Request PARMLIST length | 9 to 16 |
| Reply DATA length | 12 |
| Reply PARMLIST length | 12 |

| CPRB Field | Content/Description |
|---|---|
| Replied DATA length | 12 |
| Replied PARMLIST length | 12 |

| Request PARMLIST Values when length is X'0009' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |

| Request PARMLIST Values when length is X'0010' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |
| 9 | 1 | '1', '2', or '3' Printer number identifier |
| 10 | 1 | 'N', 'M', or 'C' Print density mode<br>'N' = 10 cpi, 'M' = 12 cpi, 'C' = 17 cpi |
| 11 | 1 | ' ' or '1', '2', '3', '4', or '5' Priority<br>(see the following Notes) |

**Notes:**

1. The priority number in Reply PARMLIST corresponds to the number assigned to a LU 1 session, and defines the order of service for print requests. Priority 1 corresponds to the LU 1 session currently in process.
2. If the adapter is not installed or the LU activity is suspended, then the priority field is returned with a blank which means the specified LU has *no* priority.

After execution, the following 3287 emulator status codes are returned in Reply DATA. See "Status codes" on page 515 for an explanation of these codes.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | C1, C2, C3, .. or C8 Communication status |
| 2 | 2 | NC, TS, or SS Pending communication function status |
| 4 | 2 | P9 or PA Emulator attending status |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 6 | 2 | NA or LA Application intervention status |
| 8 | 2 | NL, LB, or LC Listing control status |
| 10 | 2 | NS, S1, S2, or S3 Printing activity mode |

## Change print density mode (CM function)

This function changes the print density mode for the specified LU.

Specify the parameter used to define the new print density mode, in Request DATA.

The print density mode is updated in the local resource manager configuration file. This file must be available when the function is requested.

The CM function can be used when:

**Pending communication function status**
NC

| CPRB Field | Content/Description |
|---|---|
| Function code | CM |
| Request DATA length | 1 |
| Request PARMLIST length | 9 to 16 |
| Reply DATA length | 12 |
| Reply PARMLIST length | 12 |
| Replied DATA length | 12 |
| Replied PARMLIST length | 12 |

| Request PARMLIST Values when length is X'0009' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |

| Request PARMLIST Values when length is X'0010' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | 'N', 'M', or 'C' New print density mode<br>'N' = 10 cpi, 'M' = 12 cpi, 'C' = 17 cpi |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |
| 9 | 1 | '1', '2', or '3' Printer number identifier |
| 10 | 1 | 'N', 'M', or 'C' Print density mode<br>'N' = 10 cpi, 'M' = 12 cpi, 'C' = 17 cpi |
| 11 | 1 | ' ' or '1', '2', '3', '4', or '5' Priority (see the following Notes) |

**Notes:**

1. The priority number in Reply PARMLIST corresponds to the number assigned to a LU 1 session, and defines the order of service for print requests. Priority 1 corresponds to the LU 1 session currently in process.
2. If the adapter is not installed or the LU activity is suspended, then the priority field is returned with a blank which means the specified LU has *no* priority.

After execution, the following 3287 emulator status codes are returned in Reply DATA. See "Status codes" on page 515 for an explanation of these codes.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | C1, C2, C3, .. or C8 Communication status |
| 2 | 2 | NC, TS, or SS Pending communication function status |
| 4 | 2 | P9 or PA Emulator attending status |
| 6 | 2 | NA or LA Application intervention status |
| 8 | 2 | NL, LB, or LC Listing control status |
| 10 | 2 | NS, S1, S2, or S3 Printing activity mode |

## Change LU priority (CP function)

This function changes the LU priority in a printer queue. The LU with the current *in session processing* status has the highest priority, priority 1, which cannot be changed. The priority equal to ' ' or 0, can also not be changed.

Supply the function parameter in Request DATA. The parameter is used for defining the new priority for the LU and can take a value 2, 3, 4, or 5. After a LU priority has been changed, other LUs in the printer queue shift accordingly.

The new priority defined must be different from the present priority and cannot exceed the number of LUs with listings that are to be printed at the same printer.

LUs that have status *SUSPEND* have no priority.

The CP function can be used when:

**Pending communication function status**
NC

**Present priority** Any, except ' ', 0, or 1

| CPRB Field | Content/Description |
|---|---|
| Function code | CP |
| Request DATA length | 1 |
| Request PARMLIST length | 9 to 16 |
| Reply DATA length | 12 |
| Reply PARMLIST length | 12 |
| Replied DATA length | 12 |
| Replied PARMLIST length | 12 |

| Request PARMLIST Values when length is X'0009' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |

| Request PARMLIST Values when length is X'0010' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | '2', '3', '4', or '5' Function parameter: new priority number |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |
| 9 | 1 | '1', '2', or '3' Printer number identifier |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 10 | 1 | 'N', 'M', or 'C' Print density mode<br>'N' = 10 cpi, 'M' = 12 cpi, 'C' = 17 cpi |
| 11 | 1 | ' ' or '0', '1', '2', '3', '4', or '5' Priority<br>(see the following Notes) |

**Notes:**

1. The priority number in Reply PARMLIST corresponds to the number assigned to a LU 1 session, and defines the order of service for print requests. Priority 1 corresponds to the LU 1 session currently in process.
2. If the adapter is not installed or the LU activity is suspended, then the priority field is returned with a blank which means the specified LU has *no* priority.
3. The priority number returned equals 0 if the adapter is installed, the LU activity is not suspended, and no listing is queued.

After execution, the following 3287 emulator status codes are returned in Reply DATA. See "Status codes" on page 515 for an explanation of these codes.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | C1, C2, C3, .. or C8 Communication status |
| 2 | 2 | NC, TS, or SS Pending communication function status |
| 4 | 2 | P9 or PA Emulator attending status |
| 6 | 2 | NA or LA Application intervention status |
| 8 | 2 | NL, LB, or LC Listing control status |
| 10 | 2 | NS, S1, S2, or S3 Printing activity mode |

## End operator intervention (EI function)

This function deactivates the application intervention condition that is caused when the AI function is requested.

The EI function can be used when:

**Application intervention status** LA

**Pending communication function status**
NC

| CPRB Field | Content/Description |
|---|---|
| Function code | EI |
| Request DATA length | 0 |
| Request PARMLIST length | 9 to 16 |
| Reply DATA length | 12 |

| CPRB Field | Content/Description |
|---|---|
| Reply PARMLIST length | 12 |
| Replied DATA length | 12 |
| Replied PARMLIST length | 12 |

| Request PARMLIST Values when length is X'0009' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |

| Request PARMLIST Values when length is X'0010' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |
| 9 | 1 | '1', '2', or '3' Printer number identifier |
| 10 | 1 | 'N', 'M', or 'C' Print density mode<br>'N' = 10 cpi, 'M' = 12 cpi, 'C' = 17 cpi |
| 11 | 1 | ' ' or '1', '2', '3', '4', or '5' Priority<br>(see the following Notes) |

**Notes:**

1. The priority number in Reply PARMLIST corresponds to the number assigned to a LU 1 session, and defines the order of service for print requests. Priority 1 corresponds to the LU 1 session currently in process.
2. If the adapter is not installed or the LU activity is suspended, then the priority field is returned with a blank which means the specified LU has *no* priority.

After execution, the following 3287 emulator status codes are returned in Reply DATA. See "Status codes" on page 515 for an explanation of these codes.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | C1, C2, C3,.. or C8 Communication status |
| 2 | 2 | NC, TS, or SS Pending communication function status |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 4 | 2 | P9 or PA Emulator attending status |
| 6 | 2 | NA or LA Application intervention status |
| 8 | 2 | NL, LB, or LC Listing control status |
| 10 | 2 | NS, S1, S2, or S3 Printing activity mode |

## End listing (EL function)

This function ends the listing currently in progress, only when session listing control status is LC (*wait for an application program to finish listing*).

The result of requesting the EL function is that an *end of listing* condition is simulated. A different LU can thereby access the same printer. The EL function can be used when:

**End listing control status** LC

| CPRB Field | Content/Description |
|---|---|
| Function code | EL |
| Request DATA length | 0 |
| Request PARMLIST length | 9 to 16 |
| Reply DATA length | 12 |
| Reply PARMLIST length | 12 |
| Replied DATA length | 12 |
| Replied PARMLIST length | 12 |

| Request PARMLIST Values when length is X'0009' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |

| Request PARMLIST Values when length is X'0010' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |
| 9 | 1 | '1', '2', or '3' Printer number identifier |
| 10 | 1 | 'N', 'M', or 'C' Print density mode<br>'N' = 10 cpi, 'M' = 12 cpi, 'C' = 17 cpi |
| 11 | 1 | ' ' or '1', '2', '3', '4', or '5' Priority<br>(see the following Notes) |

**Notes:**

1. The priority number in Reply PARMLIST corresponds to the number assigned to a LU 1 session, and defines the order of service for print requests. Priority 1 corresponds to the LU 1 session currently in process.
2. If the adapter is not installed or the LU activity is suspended, then the priority field is returned with a blank which means the specified LU has *no* priority.

After execution, the following 3287 emulator status codes are returned in Reply DATA. See "Status codes" on page 515 for an explanation of these codes.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | C1, C2, C3, .. or C8 Communication status |
| 2 | 2 | NC, TS, or SS Pending communication function status |
| 4 | 2 | P9 or PA Emulator attending status |
| 6 | 2 | NA or LA Application intervention status |
| 8 | 2 | NL, LB, or LC Listing control status |
| 10 | 2 | NS, S1, S2, or S3 Printing activity mode |

## Get status (GS function)

This function obtains the current status of the LU 1 session and printer assigned to a specified 3287 printer emulator. (Other uses of this function in the local resource manager are described on pages 520, 541, and 547.)

| CPRB Field | Content/Description |
|---|---|
| Function code | GS |
| Request DATA length | 0 |
| Request PARMLIST length | 9 to 16 |
| Reply DATA length | 12 |
| Reply PARMLIST length | 12 |
| Replied DATA length | 12 |
| Replied PARMLIST length | 12 |

| Request PARMLIST Values when length is X'0009' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |

| Request PARMLIST Values when length is X'0010' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |
| 9 | 1 | '1', '2', or '3' Printer number identifier |
| 10 | 1 | 'N', 'M', or 'C' Print density mode<br>'N' = 10 cpi, 'M' = 12 cpi, 'C' = 17 cpi |
| 11 | 1 | ' ' or '1', '2', '3', '4', or '5' Priority<br>(see the following Notes) |

**Notes:**

1. The priority number in Reply PARMLIST corresponds to the number assigned to a LU 1 session, and defines the order of service for print requests. Priority 1 corresponds to the LU 1 session currently in process.
2. If the adapter is not installed or the LU activity is suspended, then the priority field is returned with a blank which means the specified LU has *no* priority.

After execution, the following 3287 emulator status codes are returned in Reply DATA. See "Status codes" on page 515 for an explanation of these codes.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | C1, C2, C3, .. or C8 Communication status |
| 2 | 2 | NC, TS, or SS Pending communication function status |
| 4 | 2 | P9 or PA Emulator attending status |
| 6 | 2 | NA or LA Application intervention status |
| 8 | 2 | NL, LB, or LC Listing control status |
| 10 | 2 | NS, S1, S2, or S3 Printing activity mode |

## Initiate session (IS function)

This function causes the host computer to reestablish the specified LU 1 session, after a *power on* notification has been received. This function can be requested after a TS function.

The IS function can be used when:

**Communication status** C2, C5, C6, C7, or C8

**Pending communication function status**
NC

| CPRB Field | Content/Description |
|---|---|
| Function code | IS |
| Request DATA length | 0 |
| Request PARMLIST length | 9 to 16 |
| Reply DATA length | 12 |
| Reply PARMLIST length | 12 |
| Replied DATA length | 12 |
| Replied PARMLIST length | 12 |

| Request PARMLIST Values when length is X'0009' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |

| Request PARMLIST Values when length is X'0010' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |
| 9 | 1 | '1', '2', or '3' Printer number identifier |
| 10 | 1 | 'N', 'M', or 'C' Print density mode<br>'N' = 10 cpi, 'M' = 12 cpi, 'C' = 17 cpi |
| 11 | 1 | ' ' or '1', '2', '3', '4', or '5' Priority<br>(see the following Notes) |

**Notes:**

1. The priority number in Reply PARMLIST corresponds to the number assigned to a LU 1 session, and defines the order of service for print requests. Priority 1 corresponds to the LU 1 session currently in process.
2. If the adapter is not installed or the LU activity is suspended, then the priority field is returned with a blank which means the specified LU has *no* priority.

After execution, the following 3287 emulator status codes are returned in Reply DATA. See “Status codes” on page 515 for an explanation of these codes.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | C1, C2, C3, .. or C8 Communication status |
| 2 | 2 | NC, TS, or SS Pending communication function status |
| 4 | 2 | P9 or PA Emulator attending status |
| 6 | 2 | NA or LA Application intervention status |
| 8 | 2 | NL, LB, or LC Listing control status |
| 10 | 2 | NS, S1, S2, or S3 Printing activity mode |

## Suspend service (SS function)

This function stops the operation of the specified LU 1 session.

The function mode is supplied in Request DATA and can take one of the following values:

- ’ ’ (no force). If a printing process is active then the session is suspended only when the listing has been completed. If no printing process is active the LU operation is suspended immediately.
- ’F’ (force) causes the LU operation to be suspended immediately.

The LU operation activity can be resumed by using the AS function.

Points to consider concerning the compatibility between function and communication status, are the following. The SS function can be used when:

**Printing activity mode** S3 (when mode is ’F’) or NS

**Pending communication function status**
SS (when mode is ’F’) or NC

| CPRB Field | Content/Description |
|---|---|
| Function code | SS |
| Request DATA length | 1 |
| Request PARMLIST length | 9 to 16 |
| Reply DATA length | 12 |
| Reply PARMLIST length | 12 |

| CPRB Field | Content/Description |
|---|---|
| Replied DATA length | 12 |
| Replied PARMLIST length | 12 |

| Request PARMLIST Values when length is X'0009' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |

| Request PARMLIST Values when length is X'0010' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Function mode<br>**'F'** force<br>**' '** do not force |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |
| 9 | 1 | '1', '2', or '3' Printer number identifier |
| 10 | 1 | 'N', 'M', or 'C' Print density mode<br>'N' = 10 cpi, 'M' = 12 cpi, 'C' = 17 cpi |
| 11 | 1 | ' ' or '1', '2', '3', '4', or '5' Priority<br>(see the following Notes) |

**Notes:**

1. The priority number in Reply PARMLIST corresponds to the number assigned to a LU 1 session, and defines the order of service for print requests. Priority 1 corresponds to the LU 1 session currently in process.
2. If the adapter is not installed or the LU activity is suspended, then the priority field is returned with a blank which means the specified LU has *no* priority.

After execution, the following 3287 emulator status codes are returned in Reply DATA. See "Status codes" on page 515 for an explanation of these codes.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | C1, C2, C3, .. or C8 Communication status |
| 2 | 2 | NC, TS, or SS Pending communication function status |
| 4 | 2 | P9 or PA Emulator attending status |
| 6 | 2 | NA or LA Application intervention status |
| 8 | 2 | NL, LB, or LC Listing control status |
| 10 | 2 | NS, S1, S2, or S3 Printing activity mode |

## Terminate session (TS function)

This function causes the host computer to terminate the specified LU 1 session, after a *power off* notification and *UNBIND* have been received.

The function mode is supplied in Request DATA and can take one of the following values:

- ' ' (no force). If a printing process is active then the session is terminated only when the listing has been completed. If no printing process is active the session is terminated immediately.
- 'F' (force) causes the session to end immediately.

The TS function can be used when:

**Communication status** C3, C4, or C6,

**Pending communication function status**
TS (with function mode 'F') or NC

| CPRB Field | Content/Description |
|---|---|
| Function code | TS |
| Request DATA length | 1 |
| Request PARMLIST length | 9 to 16 |
| Reply DATA length | 12 |
| Reply PARMLIST length | 12 |
| Replied DATA length | 12 |
| Replied PARMLIST length | 12 |

| Request PARMLIST Values when length is X'0009' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |

| Request PARMLIST Values when length is X'0010' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'EMU3287 ' Service identification name (left-justified and padded with blanks) |
| 8 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Function mode<br>**'F'** force<br>**' '** do not force |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |
| 8 | 1 | '1', '2', '3', '4', or '5' Emulator LU 1 session number identifier |
| 9 | 1 | '1', '2', or '3' Printer number identifier |
| 10 | 1 | 'N', 'M', or 'C' Print density mode<br>'N' = 10 cpi, 'M' = 12 cpi, 'C' = 17 cpi |
| 11 | 1 | ' ' or '1', '2', '3', '4', or '5' Priority<br>(see the following Notes) |

**Notes:**

1. The priority number in Reply PARMLIST corresponds to the number assigned to a LU 1 session, and defines the order of service for print requests. Priority 1 corresponds to the LU 1 session currently in process.
2. If the adapter is not installed or the LU activity is suspended, then the priority field is returned with a blank which means the specified LU has *no* priority.

After execution, the following 3287 emulator status codes are returned in Reply DATA. See "Status codes" on page 515 for an explanation of these codes.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | C1, C2, C3, .. or C8 Communication status |
| 2 | 2 | NC, TS, or SS Pending communication function status |
| 4 | 2 | P9 or PA Emulator attending status |
| 6 | 2 | NA or LA Application intervention status |
| 8 | 2 | NL, LB, or LC Listing control status |
| 10 | 2 | NS, S1, S2, or S3 Printing activity mode |

## Functions relating to the printer manager server

The emulator is described in Chapter 11, “Printer manager server” on page 199. The local resource manager functions are listed in alphabetical order of function code.

## Get status (GS function)

This function obtains the status of the specified printer, together with the following information:

- Identification names of the resources that have been assigned to be managed by PRTMGR
- Printer use type: *exclusive* or *alternate*
- Identification name of the resource using the printer
- Identification name of the resource pending for exclusive use

(Other uses of this function in the local resource manager are described on pages 520, 534, and 547.)

| CPRB Field | Content/Description |
|---|---|
| Function code | GS |
| Request DATA length | 0 |
| Request PARMLIST length | 9 |
| Reply DATA length | 19 |
| Reply PARMLIST length | 40 |
| Replied DATA length | 19 |
| Replied PARMLIST length | 8 to 40 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | ’PRTMGR ’ Resource identification name (left-justified and padded with blanks) |
| 8 | 1 | ’1’, ’2’, or ’3’ Printer number identifier |

After execution, the identification names of the resources that are assigned to printer manager, are returned in Reply PARMLIST.

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | **Note:** The reference field is filled by one of the following resource names:<br>**'PR47X2 '** financial printer server<br>**'EMU3287 '** 3287 printer emulator<br>**'cccccccc'** User0 printer server<br>**'cccccccc'** User1 printer server<br>**'cccccccc'** User2 printer server<br>**'PRINT '** DOS print command |

The maximum number of resources assigned is six. Therefore the maximum Reply PARMLIST length is 40 bytes. The Reply PARMLIST can contain one 8 byte length entry for each resource assigned. The 'PRINT ' resource identifies any printing not originated by LANDP resources. 'PRINT ' resource is always assigned to a printer.

After execution, the status codes available for printer manager and other printer use information are returned in Reply DATA. See "Status codes" on page 515 for an explanation of these codes.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Printer/Port status<br><br>PB, PC, PD, PE, PF, PG, or PH |
| 2 | 1 | Printer use type<br>**'E'** exclusive<br>**'A'** alternate |
| 3 | 8 | Resource name using the printer<br><br>'PR47X2 ', 'EMU3287 ', User server names, 'PRINT ', or ' ' (if port free in alternate printer use type) |
| 11 | 8 | Resource name pending for exclusive use<br><br>'PR47X2 ', 'EMU3287 ', User server names, 'PRINT ' or ' ' (if no resource name is pending for exclusive use) |

## Set alternate mode (SA function)

This function sets the printer manager operation to *alternate*. The use of the specified port will then be alternate printer use type.

The SA function can be used when:

**Printer/Port status** Any, except PC

| CPRB Field | Content/Description |
|---|---|
| Function code | SA |
| Request DATA length | 0 |
| Request PARMLIST length | 9 |
| Reply DATA length | 19 |
| Reply PARMLIST length | 40 |
| Replied DATA length | 19 |
| Replied PARMLIST length | 8 to 40 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'PRTMGR ' Resource identification name (left-justified and padded with blanks) |
| 8 | 1 | '1', '2', or '3' Printer number identifier |

After execution, the identification names of the resources that are assigned to printer manager, are returned in Reply PARMLIST.

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | List of resources assigned to the printer.<br>**Note:** The reference field is filled by one of the following resource names:<br>**'PR47X2 '** financial printer server<br>**'EMU3287 '** 3287 printer emulator<br>**'cccccccc'** User0 printer server<br>**'cccccccc'** User1 printer server<br>**'cccccccc'** User2 printer server<br>**'PRINT '** DOS print command |

The maximum number of resources assigned is six. Therefore, the maximum Reply PARMLIST length is 40 bytes. The Reply PARMLIST can contain one 8 byte length entry for each resource assigned. The 'PRINT ' resource identifies any printing not originated by LANDP resources. 'PRINT ' resource is always assigned to a printer.

After execution, the status codes available for printer manager and other printer use information are returned in Reply DATA. See "Status codes" on page 515 for an explanation of these codes.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Printer/Port status<br><br>PB, PC, PD, PE, PF, PG, or PH |
| 2 | 1 | Printer use type<br>**'E'** exclusive<br>**'A'** alternate |
| 3 | 8 | Resource name using the printer<br><br>'PR47X2 ', 'EMU3287 ', User server names, 'PRINT ', or '  ' (if port free in alternate printer use type) |
| 11 | 8 | Resource name pending for exclusive use<br><br>'PR47X2 ', 'EMU3287 ', User server names, 'PRINT ' or '  ' (if no resource name is pending for exclusive use) |

## Set exclusive mode (SE function)

This function sets the printer manager operation to *exclusive*. The specified port is then exclusively used by the resource specified by the resource name parameter. The resource must be selected from the list of assigned resources returned in Reply PARMLIST. The SE function can be used when:

**Printer/Port status** Any, except PC

| CPRB Field | Content/Description |
|---|---|
| Function code | SE |
| Request DATA length | 0 |
| Request PARMLIST length | 9 |
| Reply DATA length | 19 |
| Reply PARMLIST length | 40 |
| Replied DATA length | 19 |
| Replied PARMLIST length | 8 to 40 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'PRTMGR ' Resource identification name (left-justified and padded with blanks) |
| 8 | 1 | '1', '2', or '3' Printer number identifier |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'PR47X2 ', 'EMU3287 ', User server names, or 'PRINT ' (Resource name that will use the printer in exclusive mode) |

After execution, the identification names of the resources that are assigned to printer manager are returned in the Reply PARMLIST.

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | List of resources assigned to the printer.<br>**Note:** The reference field is filled by one of the following resource names:<br>**'PR47X2 '** financial printer server<br>**'EMU3287 '** 3287 printer emulator<br>**'cccccccc'** User0 printer server<br>**'cccccccc'** User1 printer server<br>**'cccccccc'** User2 printer server<br>**'PRINT '** DOS print command |

The maximum number of resources assigned is six. Therefore the maximum Reply PARMLIST length is 40 bytes. The Reply PARMLIST can contain one 8 byte length entry for each resource assigned. The 'PRINT ' resource identifies any printing not originated by LANDP resources. 'PRINT ' resource is always assigned to a printer.

After execution, the status codes available for printer manager and other printer use information are returned in Reply DATA. See “Status codes” on page 515 for an explanation of these codes.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | Printer/Port status<br><br>PB, PC, PD, PE, PF, PG, or PH |
| 2 | 1 | Printer use type<br>**'E'** exclusive<br>**'A'** alternate |
| 3 | 8 | Resource name using the printer<br><br>'PR47X2 ', 'EMU3287 ', User server names, 'PRINT ', or ' ' (if port free in alternate printer use type) |
| 11 | 8 | Resource name pending for exclusive use<br><br>'PR47X2 ', 'EMU3287 ', User server names, 'PRINT ', or ' ' (if no resource name is pending for exclusive use) |

## Functions relating to the banking printer program (BPP)

The banking printer program is a part of the IBM Financial Branch System Integrator. The local resource manager functions are listed in alphabetical order of function code.

### Cancel pending functions (CC function)

This function cancels pending functions. For BPP the pending functions can be LP or HP.

| CPRB Field | Content/Description |
|---|---|
| Function code | CC |
| Request DATA length | 0 |
| Request PARMLIST length | 9 to 16 |
| Reply DATA length | 10 |
| Reply PARMLIST length | 9 |
| Replied DATA length | 10 |
| Replied PARMLIST length | 9 |

| Request PARMLIST Values when length is X'0009' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'BPP ' Service identification name (left-justified and padded with blanks) |
| 8 | 1 | '1', '2', '3', or '4' Printer LU 0 session or port number |

| Request PARMLIST Values when length is X'0010' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'BPP ' Service identification name (left-justified and padded with blanks) |
| 8 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |
| 8 | 1 | '1', '2', '3', or '4' Printer LU 0 session or port number |

After execution, the following BPP status codes are returned in Reply DATA. See "Status codes" on page 515 for an explanation of these codes.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | C1, C2, C3, .. or C8 Communication status |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 2 | P1, P2, P3, P4, or P5 Printer attending mode status |
| 4 | 2 | NP, LP, or HP Pending printer attending mode status |
| 6 | 2 | P9 or PA Emulator attending status |
| 8 | 2 | PB, PC, PD, PE, PF, PG, PH, or PI Printer/Port status |

## Get status (GS function)

This function obtains the current communication and printer status of the LU 0 session assigned to a specified BPP printer. (Other uses of this function in the local resource manager are described on pages 520, 534, and 541.)

| CPRB Field | Content/Description |
|---|---|
| Function code | GS |
| Request DATA length | 0 |
| Request PARMLIST length | 9 to 16 |
| Reply DATA length | 10 |
| Reply PARMLIST length | 9 |
| Replied DATA length | 10 |
| Replied PARMLIST length | 9 |

| Request PARMLIST Values when length is X'0009' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'BPP ' Service identification name (left-justified and padded with blanks) |
| 8 | 1 | '1', '2', '3', or '4' Printer LU 0 session or port number |

| Request PARMLIST Values when length is X'0010' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'BPP ' Service identification name (left-justified and padded with blanks) |
| 8 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |
| 8 | 1 | '1', '2', '3', or '4' Printer LU 0 session or port number |

After execution, the following BPP status codes are returned in Reply DATA. See “Status codes” on page 515 for an explanation of these codes.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | C1, C2, C3, .. or C8 Communication status |
| 2 | 2 | P1, P2, P3, P4, or P5 Printer attending mode status |
| 4 | 2 | NP, LP, or HP Pending printer attending mode status |
| 6 | 2 | P9 or PA Emulator attending status |
| 8 | 2 | PB, PC, PD, PE, PF, PG, PH, or PI Printer/Port status |

## Change printer to remote mode (HP function)

This function sets the printer to remote mode.

Function mode ’F’ can be used together with the HP function code. If the printer is in local mode the function is processed:

- For function mode ’ ’ (not force): when the current transaction has ended.
- For function mode ’F’ (force): immediately (any in process transaction are interrupted).

The HP function can be used when:

**Printer attending mode status** P3, P4, or P5

**Pending printer attending mode status**
HP (with function mode ’F’) or NP

| CPRB Field | Content/Description |
|---|---|
| Function code | HP |
| Request DATA length | 1 |
| Request PARMLIST length | 9 to 16 |
| Reply DATA length | 10 |
| Reply PARMLIST length | 9 |
| Replied DATA length | 10 |
| Replied PARMLIST length | 9 |

| Request PARMLIST Values when length is X'0009' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | ’BPP ’ Service identification name (left-justified and padded with blanks) |
| 8 | 1 | ’1’, ’2’, ’3’, or ’4’ Printer LU 0 session or port number |

| Request PARMLIST Values when length is X'0010' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'BPP ' Service identification name (left-justified and padded with blanks) |
| 8 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Function mode:<br>**' '** do not force<br>**'F'** force |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |
| 8 | 1 | '1', '2', '3', or '4' Printer LU 0 session or port number |

After execution, the following BPP status codes are returned in Reply DATA. See "Status codes" on page 515 for an explanation of these codes.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | C1, C2, C3, .. or C8 Communication status |
| 2 | 2 | P1, P2, P3, P4, or P5 Printer attending mode status |
| 4 | 2 | NP, LP, or HP Pending printer attending mode status |
| 6 | 2 | P9 or PA Emulator attending status |
| 8 | 2 | PB, PC, PD, PE, PF, PG, PH, or PI Printer/Port status |

## Change printer to local mode (LP function)

This function changes the printer to local mode.

Function mode 'F' can be used together with the LP function code.

This command allows local printing until the next HP command takes effect.

The LP function can be used when:

**Printer attending mode status** P1, P2, or P5

**Pending printer attending mode status**
LP (with function mode 'F') or NP

| CPRB Field | Content/Description |
|---|---|
| Function code | LP |
| Request DATA length | 1 |
| Request PARMLIST length | 9 to 16 |
| Reply DATA length | 10 |
| Reply PARMLIST length | 9 |
| Replied DATA length | 10 |
| Replied PARMLIST length | 9 |

| Request PARMLIST Values when length is X'0009' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'BPP ' Service identification name (left-justified and padded with blanks) |
| 8 | 1 | '1', '2', '3', or '4' Printer LU 0 session or port number |

| Request PARMLIST Values when length is X'0010' | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | 'BPP ' Service identification name (left-justified and padded with blanks) |
| 8 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Function mode:<br>**' '** do not force<br>**'F'** force |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Host/application identifier (any valid ASCII string of 8 characters) |
| 8 | 1 | '1', '2', '3', or '4' Printer LU 0 session or port number |

After execution, the following BPP status codes are returned in Reply DATA. See "Status codes" on page 515 for an explanation of these codes.

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | C1, C2, C3, .. or C8 Communication status |
| 2 | 2 | P1, P2, P3, P4, or P5 Printer attending mode status |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 4 | 2 | NP, LP, or HP Pending printer attending mode status |
| 6 | 2 | P9 or PA Emulator attending status |
| 8 | 2 | PB, PC, PD, PE, PF, PG, PH, or PI Printer/Port status |

# Part 7. Facilities and utilities

The LANDP family provides several utility programs and other facilities, some of which have an application programming interface. These are described in this part of the book.

**Chapter 23, "Running LANDP for DOS applications in OS/2 or Windows workstations" on page 555**
This chapter describes how you can run LANDP for DOS applications in a LANDP for OS/2 or LANDP for Windows workstation, including substituting server names, specifying asynchronous events, and automatic session release for the SNA server. It also explains how to write user exits that perform pre-invocation and post-invocation processing.

**Chapter 24, "Running LANDP for DOS applications in Linux workstations" on page 563**
This chapter describes how you can run LANDP for DOS applications in a LANDP for Linux workstation. It describes LANDP Linux DosBox support and how to use DOSEMU.

**Chapter 25, "ASCII-EBCDIC translation" on page 569**
This chapter presents the functions provided by the LANDP for DOS ASCII-EBCDIC translation server for servicing requests originating in applications or other servers.

**Chapter 26, "IBM 4707 monochrome display" on page 575**
This chapter describes how you can switch between the display modes of the IBM 4707 using subroutines provided by LANDP for DOS.

**Chapter 27, "Object postbox server and batch machines" on page 577**
This chapter describes how you can send and receive messages and data to and from other users in the LANDP workgroup, and how you can use this facility to execute batch programs.

© Copyright IBM Corp. 1992, 2002

# Chapter 23. Running LANDP for DOS applications in OS/2 or Windows workstations

The LANDP multiple virtual DOS machine (MVDM) relay is provided to enable **LANDP for DOS** applications and emulators to run in LANDP for OS/2 or LANDP for Windows workstations. MVDM enables the LANDP for OS/2 or LANDP for Windows supervisor to process requests from LANDP for DOS clients (not servers) that are running on the OS/2 or Windows virtual DOS machine.

For information on the virtual DOS machine, refer to the OS/2 or Windows documentation.

The MVDM relay sends LANDP for DOS application requests to the LANDP for OS/2 or LANDP for Windows supervisor and the corresponding replies from the LANDP for OS/2 or LANDP for Windows supervisor to the application, running in a virtual DOS machine (VDM). The MVDM relay translates INT X'7F' API requests into Common API requests.

Compatibility between LANDP for DOS and LANDP for OS/2 or LANDP for Windows processing is achieved at the CPRB server name and function identification level. Before sending a CPRB or replying to the application running in a virtual DOS machine, the MVDM relay, if necessary, changes the LANDP for OS/2 or LANDP for Windows reply CPRB fields and formats them so that the LANDP for DOS application receives the CPRB as if the request was processed under LANDP for DOS.

User servers, LANDP servers, and any other LANDP component loaded through a LOADER statement cannot run in a virtual DOS machine.

## How the MVDM relay works on LANDP for OS/2

The MVDM relay for LANDP for OS/2 is made up of three programs:

- EHCVDMVD.SYS — a virtual device driver program
- EHCBOXM.EXE — a monitor program
- EHCVDMGR.EXE — the virtual DOS machine manager server

The device driver program enables data interchange between the virtual DOS machine INT X'7F' routine and the MVDM relay monitor program.

The device driver traps LANDP for DOS application calls and pass the requests to the monitor program. It also passes the corresponding replies from the monitor program to the LANDP for DOS application. The application requests are completed when the application regains control.

The monitor program translates the LANDP for DOS application requests into LANDP for OS/2 sends the requests to the LANDP for OS/2 supervisor, and returns the replies to the device driver program.

© Copyright IBM Corp. 1992, 2002

The virtual DOS machine manager server, EHCVDMGR.EXE, is a LANDP for OS/2 server that controls processing of monitor programs. Each monitor program is created:

- The first time you issue a LANDP for DOS function call in the virtual DOS machine session
- The first time you issue a LANDP for DOS function call after an ES function is performed in this virtual DOS machine session

When you unload the virtual DOS machine manager server, the monitor programs that were controlled by this virtual DOS machine manager server are unloaded as well.

If you close a virtual DOS machine session that was associated with a monitor program, it is unloaded.

If your application issues an EJ function call to the supervisor from a virtual DOS machine, the monitor program associated to the virtual DOS machine session is released and is used again when this virtual DOS machine session issues LANDP requests.

If your application issues an ES function call to the supervisor from a virtual DOS machine, the monitor program associated to the virtual DOS machine session is unloaded.

## How the MVDM relay works on LANDP for Windows

The MVDM relay for LANDP for Windows is made up of four programs:

- EHCVDSPV.COM — interrupt 7F intercept program
- EHCVDVXD.DLL — virtual device driver program
- EHCVDMIR.EXE — monitor program
- EHCVDMGR.EXE — the MVDM relay manager

Interrupt 7F is intercepted by a terminate and stay resident (TSR) program (EHCVDSPV.COM) that must be loaded before running any LANDP for DOS applications.

When the application makes a call, the request is validated and passed to the virtual device driver program (EHCVDVXD.DLL).

On the first call from any given virtual DOS machine, the virtual device driver signals the MVDM relay manager program (EHCVDMGR.EXE) to start the monitor program (EHCVDMIR.EXE) for this virtual DOS machine.

Using shared memory, the request is passed to LANDP for Windows for processing. The virtual DOS machine is suspended.

On completion, the results are passed back to the virtual DOS machine, which resumes processing.

## Sample files

Two sample files are provided by LANDP that help you write a configuration file and a user exit routine for LANDP for OS/2 or LANDP for Windows:

- EHCBOXS.CFG
- EHCEXITS.C

The configuration file tailors the behavior of the MVDM relay in order to enable LANDP for DOS applications to run unchanged in the virtual DOS machines.

## Configuration file

If necessary, a configuration file can be created. In the configuration file you specify requirements about:

- Server name substitution
- Asynchronous events
- User exits
- Session release

A sample configuration file, named EHCBOXS.CFG, is provided. You can follow this example to write your own configuration file.

The *EHCBOXS.CFG* file is shown below:

```
* LANDP BRIDGE Configuration File *
* -------------------------------- *
* Server name substitution
* <-from-> <--to-->
S DOSNAME  NEWNAME
* Asynchronous events
* !!! "KBSPV     " should NOT be specified !!!
A CPSNA##
A CPX25NAT##
A RDMSRE4717
A WRMSRE4717
A RDPINP4718
A T1SPV
A T2SPV
A T3SPV
A T4SPV
A T5SPV
A T6SPV
A T7SPV
A T8SPV
* User exits
U EHCDLLS  EHCEXITS
* Automatic session connect
C CONNECT Y
* Session release after close
R RELEASE Y
* End of Configuration File
```

The configuration file is an ASCII file, and it is loaded the first time the MVDM relay is invoked. Each record in the file is an entry of the specification tables, which are created by the MVDM relay from the configuration file information. They are described in the following sections.

If no configuration file is specified in the installation statement of the monitor program, the MVDM relay uses the default configuration file, EHCBOXS.CFG. If the default configuration file is not found, the MVDM relay generates default entries for the specification tables.

If a configuration file is specified in that installation statement, but it cannot be found or it includes not valid records, a return code is displayed and the monitor program ends.

## Server name substitution

The server name substitution records are identified by the record ID 'S'. Then, using the 'S' records, the MVDM relay builds the server name substitution table. The record structure is as follows:

| Offset | Length | Content |
|---|---|---|
| 0 | 1 | Record ID: 'S' = Server Name Substitution |
| 2 | 8 | Name of the server substituted from (the LANDP for DOS server) |
| 11 | 8 | Name of the server substituted to (the LANDP for OS/2 or LANDP for Windowsserver) |

The server names are accepted without validation. The maximum number of entries is 16. If there are more than 16 entries, the excess entries are discarded.

If a server name substitution record has been specified when the MVDM relay receives a LANDP for DOS application request, it looks in the server name substitution table for the server name in the CPRB. If found, the matching server name is written in the CPRB field.

The MVDM relay enables substitution of a server for an equivalent one with a different name, including SNA session identification changes. The called function support and the function request interface are not verified.

## Asynchronous events

This section applies only to FBSS (DOS) applications that use the FBSS supervisor local functions EX, SA, and RA. For information about these functions refer to the *IBM Financial Branch System Services Version 2 Release 2.1 Application Programming* book.

The asynchronous event records are identified by the record ID 'A'. Then, using the 'A' records, the MVDM relay builds the asynchronous event table. The record structure is as follows:

| Offset | Length | Content |
|---|---|---|
| 0 | 1 | Record ID: 'A' = Asynchronous Event |
| 2 | 2 | Event ID |
| 4 | 11 | Event server |

The maximum number of entries is 64. The records must be written in descending dispatch priority sequence. It is not necessary to specify the keyboard event "KBSPV".

When the MVDM relay receives an RA function call, the specified asynchronous events are *added* to the asynchronous event table. When the MVDM relay receives an SA function call, the specified asynchronous events are *deleted* from the asynchronous event table. When the MVDM relay receives an EX function call, a LANDP WM function call replaces it. The asynchronous events associated with the WM function call are those listed in the asynchronous event table.

If neither an RA nor an SA function call is issued before the EX function call, the current asynchronous event table is used for the WM function call set by the MVDM relay. A WM function call with no associated asynchronous event involves the application exiting an idle state after any asynchronous event.

If no configuration file exists, or if no asynchronous event record is included in the configuration file, the MVDM relay generates a default asynchronous event table to include the following:

- CP entries
  - CPSNA##
  - CPX25NAT##
- 4717 MSR/E entries
  - RDMSRE4717
  - WRMSRE4717
- 4718 PIN Pad entries
  - RDPINP4718
- Timer entries
  - T1SPV
  - T2SPV
  - T3SPV
  - T4SPV
  - T5SPV
  - T6SPV
  - T7SPV
  - T8SPV

## User exits

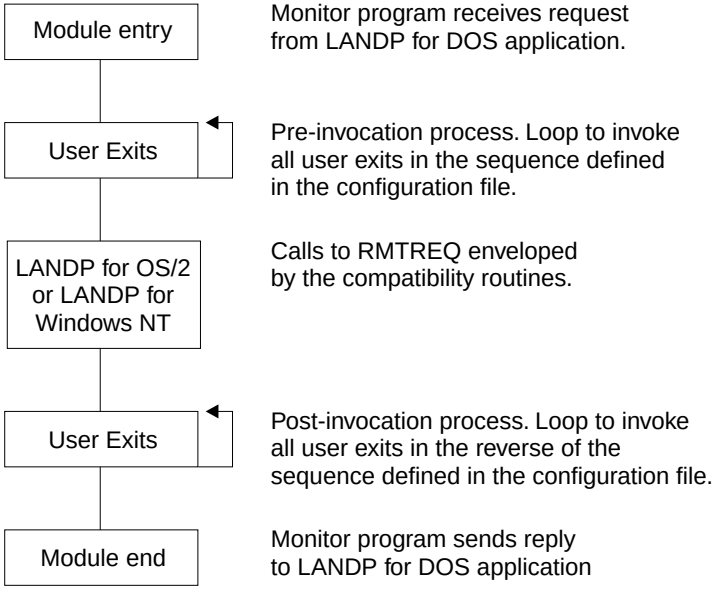
A *user exit* is a routine written by the user for some specific purpose. Those routines run before the MVDM relay calls the LANDP for OS/2 and Windows supervisor. After calling it, they run again. They can modify the CPRB fields, and the parameter and data areas. The MVDM relay runs the user exits automatically.

The user exit records are identified by the record ID 'U'. Then, using the 'U' records, the MVDM relay builds the user exit table. The record structure is as follows:

| Offset | Length | Content |
|---|---|---|
| 0 | 1 | Record ID: 'U' = User Exit |
| 2 | 8 | File name of the DLL module with the exit routine. File type is assumed to be DLL |
| 11 | 8 | Name of the exit routine |

The maximum number of entries is 4. Each record corresponds to a user exit routine to be called. The exit routines are called in the same sequence as they are defined in the configuration file for pre-invocation process, and in reverse order for post-invocation process.

When a LANDP for DOS application request is being processed, user exits are invoked. The user exits are invoked at pre-invocation and post-invocation time, as the diagram below shows.




The same user exit routine is called at pre-invocation time and at post-invocation time. A parameter is passed to the routine to indicate this. If no configuration file exists, or no user exit record is included in the configuration file, the MVDM relay generates a default user exit table to include a default user exit routine name, EHCEXITS, and the corresponding default DLL name, EHCDLLS. If the DLL is not available, or the user exit routine is not found, the MVDM relay does not run any user exit.

**To write a user exit routine**, follow the prototype below:

```
EXPENTRY ROUTINENAME (CPRBP cprbptr,
                      int instance,
                      int position,
                      void far * reserved);
```

where:

**ROUTINENAME** Is the name of the user exit routine. It returns one of the following integer values:
- **0** Correct, continue.
- **1** Skip rest of user exits in the list. If returned during pre-invocation, processing resumes at the point after the corresponding post-invocation user exit. If returned during post-invocation, subsequent user exits in the list to be called are bypassed.

**cprbptr** Is the far pointer to the request CPRB.

**instance** Has one of the following integer values:
- **0** Pre-invocation. The routine has been called before calling RMTREQ.
- **1** Post-invocation. The routine has been called after calling RMTREQ.

It is set by the MVDM relay.

**position** Is an integer that indicates the position of the current user exit in the list of user exits to be called. It is set by the MVDM relay, starting from 0. That is, the first routine to be called receives a zero value.

The user exit routines must be declared as functions that return an integer value. A zero return value means that the status is correct and that the process can continue.

A return value of **1** at pre-invocation time means that the user exit routine has formatted the CPRB as the reply CPRB. Hence, subsequent user exits and LANDP for OS/2 or Windows invocation must be bypassed, and the process must go to the point after the corresponding post-invocation user exit.

**To compile the user exit routine**, use the compiler that corresponds to the programming language you are working with.

**To link the user exit routine**, create a .DEF file where the routine is declared as EXPORT in order to include the user exit routine in a DLL. Refer to your *OS/2 Application Design Guide* or Windows documentation for an explanation on how to build DLLs. You can build one DLL for each routine, or include more than one routine in the same DLL.

**To make the DLL accessible by the MVDM relay**, specify the path for the files with extension DLL in:
- The LIBPATH statement of the CONFIG.SYS file (for OS/2)
- The PATH environment variable (for Windows)

Also, if the name of the DLL or the name of the user exit routine is different from the default name, the DLL must be specified in the configuration file and the monitor

program must be loaded with the name of the configuration file. If the MVDM relay does not find the DLL or the user exit routine specified in the configuration file, a return code is displayed and the monitor program ends.

The MVDM relay provides a sample user exit routine, named EHCEXITS.C. You can follow this example to write your own user exit routine.

## Session release

The session release record is identified by the record ID 'R'. The record structure is as follows:

| Offset | Length | Content |
|---|---|---|
| 0 | 1 | Record ID: 'R' = Session Release |
| 2 | 7 | RELEASE |
| 10 | 1 | 'Y' (Yes) or 'N' (No)<br>The default is 'Y'. |

When the LANDP for DOS application uses the SNA server, and you specify 'Y' in the session release record after the application calls the close (CL) function of the SNA server, the MVDM relay automatically generates a call to the release (RL) function of that server. If you specify 'N' in the session release record, the RL function call is not generated.

If the application already issues an RL function call, to avoid issuing two consecutive RL function calls to the SNA server, you can specify 'N' in the session release record, so that the MVDM relay does not generate a second RL function call.

## Auto connect

The auto-connect record is identified by the record ID 'C'. The record structure is as follows:

| Offset | Length | Content |
|---|---|---|
| 0 | 1 | Record ID: 'C' = Auto connect |
| 2 | 7 | CONNECT |
| 10 | 1 | 'Y' (Yes) or 'N' (No)<br>The default is 'Y'. |

When the LANDP for DOS application uses the SNA server, and you specify 'Y' in the auto-connect record, if a call to SNA returns P0 (in other words, there has been no previous CN call) the MVDM relay automatically issues a CN call, with default parameters, on behalf of the application.

If you specify 'N' in the auto-connect record, the CN function call is not generated.

If no auto-connect record is specified, the MVDM generates the CN call, if required.

# Chapter 24. Running LANDP for DOS applications in Linux workstations

The LANDP Linux DosBox support is provided to enable **LANDP for DOS** applications and emulators to run in LANDP for Linux workstations.

The LANDP Linux DosBox support enables the LANDP for Linux supervisor to process requests from LANDP for DOS clients (not servers) that are running on DOS under DOSEMU. For information on DOSEMU, refer to the documentation provided by the DOSEMU support group on their website: **www.dosemu.org**.

The DosBox relay sends LANDP for DOS application requests to the LANDP for Linux supervisor, and the corresponding replies from the LANDP for Linux supervisor to the application, running in a DOS under DOSEMU.

The DosBox relay translates INT X'7F' API requests into Common API requests.

Compatibility between LANDP for DOS and LANDP for Linux is achieved at the CPRB server name and function identification level.

Before sending a CPRB to the Linux supervisor, or before replying to the DOS application, the DosBox relay changes the CPRB format so that they receive the CPRB in the format they require.

## How the DosBox relay works on LANDP for Linux

The DosBox relay for LANDP for Linux is made up of four programs:

- EHCVDSPV.COM — interrupt X'7F' intercept program
- ehcplug.c — the main DOSEMU LANDP plugin (source code)
- ehcvdmir — monitor program
- ehcvdmgr — the DosBox relay manager

Interrupt X'7F' is intercepted by a terminate and stay resident (TSR) program, EHCVDSPV.COM, that must be loaded before you run any LANDP for DOS applications. Refer to the *LANDP Installation and Customization* book for details of this.

When the application makes a call, the request is validated and passed to the LANDP DOSEMU plugin. The plugin is provided as source code, because binary plugins are prohibited by DOSEMU for security reasons.

The plugin code can be found in directory EHCL600\DOSEMU\PLUGIN after installation. The include files that you need should be unzipped from ehcplug.zip.

**You are responsible for obtaining the DOSEMU source code from the DOSEMU website and recompiling it with the LANDP plugin included. For instructions on how to do this, refer to the DOSEMU documentation.**

© Copyright IBM Corp. 1992, 2002

On the first call from any DosBox, the LANDP DOSEMU plugin signals the DosBox relay manager to start the monitor program for this DosBox.

Using shared memory, the request is passed to the LANDP for Linux workstation for processing. Execution in the DosBox is suspended.

On completion, the results are passed back to the DosBox, which resumes processing.

## Sample files

Two sample files are provided by LANDP that help you write a configuration file and a user exit routine for LANDP for Linux:

- EHCBOXS.CFG
- EHCEXITS.C

The configuration file tailors the behavior of the MVDM relay in order to enable LANDP for DOS applications to run unchanged in the virtual DOS machines.

## Configuration file

If necessary, a configuration file can be created. In the configuration file you specify requirements about:

- Server name substitution
- Asynchronous events
- Session release

A sample configuration file, named *EHCBOXS.CFG*, is provided. You can follow this example to write your own configuration file.

The *ehcboxs.cfg* file is shown below:

```
* LANDP BRIDGE Configuration File *
* -------------------------------- *
* Server name substitution
* <-from-> <--to-->
S DOSNAME  NEWNAME
* Asynchronous events
* !!! "KBSPV     " should NOT be specified !!!
A CPSNA##
A CPX25NAT##
A RDMSRE4717
A WRMSRE4717
A RDPINP4718
A T1SPV
A T2SPV
A T3SPV
A T4SPV
A T5SPV
A T6SPV
A T7SPV
A T8SPV
* User exits
```

```
U EHCDLLS  EHCEXITS
* Automatic session connect
C CONNECT Y
* Session release after close
R RELEASE Y
* End of Configuration File
```

The configuration file is an ASCII file, and it is loaded the first time the MVDM relay is invoked. Each record in the file is an entry of the specification tables, which are created by the MVDM relay from the configuration file information. They are described in the following sections.

If no configuration file is specified in the installation statement of the monitor program, the MVDM relay uses the default configuration file, EHCBOXS.CFG. If the default configuration file is not found, the MVDM relay generates default entries for the specification tables.

If a configuration file is specified in that installation statement, but it cannot be found or it includes not valid records, a return code is displayed and the monitor program ends.

## Server name substitution

The server name substitution records are identified by the record ID 'S'. Then, using the 'S' records, the MVDM relay builds the server name substitution table. The record structure is as follows:

| Offset | Length | Content |
|---|---|---|
| 0 | 1 | Record ID: 'S' = Server Name Substitution |
| 2 | 8 | Name of the server substituted from (the LANDP for DOS server) |
| 11 | 8 | Name of the server substituted to (the LANDP for Linux server) |

The server names are accepted without validation. The maximum number of entries is 16. If there are more than 16 entries, the excess entries are discarded.

If a server name substitution record has been specified when the MVDM relay receives a LANDP for DOS application request, it looks in the server name substitution table for the server name in the CPRB. If found, the matching server name is written in the CPRB field.

The MVDM relay enables substitution of a server for an equivalent one with a different name, including SNA session identification changes. The called function support and the function request interface are not verified.

## Asynchronous events

This section applies only to FBSS (DOS) applications that use the FBSS supervisor local functions EX, SA, and RA. For information about these functions refer to the *IBM Financial Branch System Services Version 2 Release 2.1 Application Programming* book.

The asynchronous event records are identified by the record ID 'A'. Then, using the 'A' records, the MVDM relay builds the asynchronous event table. The record structure is as follows:

| Offset | Length | Content |
|---|---|---|
| 0 | 1 | Record ID: 'A' = Asynchronous Event |
| 2 | 2 | Event ID |
| 4 | 11 | Event server |

The maximum number of entries is 64. The records must be written in descending dispatch priority sequence. It is not necessary to specify the keyboard event "KBSPV".

When the MVDM relay receives an RA function call, the specified asynchronous events are *added* to the asynchronous event table. When the MVDM relay receives an SA function call, the specified asynchronous events are *deleted* from the asynchronous event table. When the MVDM relay receives an EX function call, a LANDP WM function call replaces it. The asynchronous events associated with the WM function call are those listed in the asynchronous event table.

If neither an RA nor an SA function call is issued before the EX function call, the current asynchronous event table is used for the WM function call set by the MVDM relay. A WM function call with no associated asynchronous event involves the application exiting an idle state after any asynchronous event.

If no configuration file exists, or if no asynchronous event record is included in the configuration file, the MVDM relay generates a default asynchronous event table to include the following:

- CP entries
  - CPSNA##
  - CPX25NAT##
- 4717 MSR/E entries
  - RDMSRE4717
  - WRMSRE4717
- 4718 PIN Pad entries
  - RDPINP4718
- Timer entries
  - T1SPV
  - T2SPV
  - T3SPV
  - T4SPV
  - T5SPV
  - T6SPV
  - T7SPV
  - T8SPV

## Session release

The session release record is identified by the record ID 'R'. The record structure is as follows:

| Offset | Length | Content |
|---|---|---|
| 0 | 1 | Record ID: 'R' = Session Release |
| 2 | 7 | RELEASE |
| 10 | 1 | 'Y' (Yes) or 'N' (No)<br>The default is 'Y'. |

When the LANDP for DOS application uses the SNA server, and you specify 'Y' in the session release record after the application calls the close (CL) function of the SNA server, the MVDM relay automatically generates a call to the release (RL) function of that server. If you specify 'N' in the session release record, the RL function call is not generated.

If the application already issues an RL function call, to avoid issuing two consecutive RL function calls to the SNA server, you can specify 'N' in the session release record, so that the MVDM relay does not generate a second RL function call.

## Auto connect

The auto-connect record is identified by the record ID 'C'. The record structure is as follows:

| Offset | Length | Content |
|---|---|---|
| 0 | 1 | Record ID: 'C' = Auto connect |
| 2 | 7 | CONNECT |
| 10 | 1 | 'Y' (Yes) or 'N' (No)<br>The default is 'Y'. |

When the LANDP for DOS application uses the SNA server, and you specify 'Y' in the auto-connect record, if a call to SNA returns P0 (in other words, there has been no previous CN call) the MVDM relay automatically issues a CN call, with default parameters, on behalf of the application.

If you specify 'N' in the auto-connect record, the CN function call is not generated.

If no auto-connect record is specified, the MVDM generates the CN call, if required.

# Chapter 25. ASCII-EBCDIC translation

This chapter provides information on how the **LANDP for DOS** ASCII–EBCDIC translation server handles service requests. It also describes two translation routines (see page 573) that you can use in a LANDP for DOS or LANDP for OS/2 application program to translate between ASCII and EBCDIC.

This chapter also provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields. If you need more information about the CPRB fields, refer to Appendix A, “Connectivity programming request block” on page 669.

*Table 44. Function Codes used in the ASCII–EBCDIC Translation Server*

| Function code | Description | Env. | Page |
|---|---|---|---|
| AE | ASCII to EBCDIC translation | 0--- | 571 |
| EA | EBCDIC to ASCII translation | 0--- | 572 |

The first and second columns give the function code and the name of the function. The third column shows the operating environment of the function. “0---” means that it is available from LANDP for DOS servers. The last column refers to the page where you can find the function described.

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | 26 | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 26 | 2 | 26 | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | EHCDBTR | Server name |

The following fields are variable and are discussed in each function request description:
- Function code
- Request DATA length
- Reply DATA length

© Copyright IBM Corp. 1992, 2002

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 44 | 2 | | Replied PARMLIST length |
| 46 | 2 | | Replied DATA length |

If the request was successful, the *router return code* and the *server return code* are both X'00000000'. In all other cases, see the appropriate section in the *LANDP Problem Determination* book to see if you should take any action. The return values in Reply PARMLIST and DATA should be ignored if there is an error.

The following fields are variable and are discussed in each function request description:
- Replied PARMLIST length
- Replied DATA length

**PARMLIST:** The Request PARMLIST and Reply PARMLIST fields for the ASCII-EBCDIC translation server are:

| Offset | Length | Content |
|---|---|---|
| 0 | 1 | Flag0 = 'D'. Set by the application in the EA function. It indicates that the source string begins with a DBCS character. |
| 1 | 1 | Flag1 = 'S'. Set by the application in either the EA or the AE function to indicate that the target string has shift-out/shift-in (SO/SI) characters. |
| 2 | 1 | Flag2 = 'S'. Set by the application in either the EA or the AE function to indicate that the source string has SO/SI characters. |
| 3 | 1 | Flag3 = 'D' or 'S'. Returned by the server to indicate whether the last translated character was a DBCS one ('D') or any other type of character ('S'). |
| 4 | 2 | Length. Returned by the server. It specifies the length of the translated string or the necessary length to translate the complete source string. |
| **Note:** Flags not used must be filled with blanks. | | |

**DATA:** Request DATA and Reply DATA contain data to be sent to or received from the server. The use of these areas is explained in the description of each function request.

## Double-byte character sets

If a short buffer condition occurs due to a Reply DATA length not large enough to hold the complete target string, the server checks the last byte of the target string. If this byte is a half double-byte character set (DBCS) character, the server replaces the half character with a null.

If the last character of the source string is a half DBCS character, flag3 equals 'D' and the target string contains a null in the half DBCS character place.

If the source string contains one of the following:

- A single-byte character set (SBCS) character that does not have a corresponding character in the target code page
- A DBCS character that does not have a corresponding character in the target code page (undefined)
- A DBCS character, within a SO/SI bracket, that is out of the valid code range for DBCS characters (invalid)

the server translates these characters into the substitution characters with the code points shown in the following table:

| Language | Host DBCS | PC DBCS | Host SBCS | PC SBCS |
|---|---|---|---|---|
| Korean | X'FEFE' | X'AFFE' | X'3F' | X'7F' |
| T-Chinese (undefined) | X'FEFD' | X'FCFB' | X'3F' | X'7F' |
| T-Chinese (invalid) | X'FEFE' | X'FCFC' | | |
| S-Chinese | X'FEFE' | X'FCFC' | X'3F' | X'7F' |

## ASCII-EBCDIC translation server request reference

Two functions are provided by this server:

- ASCII-to-EBCDIC translation (AE function)
- EBCDIC-to-ASCII translation (EA function)

## ASCII-EBCDIC translation (AE function)

This function translates an ASCII string into an EBCDIC string.

| CPRB Field | Content/Description |
|---|---|
| Function code | AE |
| Request DATA length | Length of the source string |
| Reply DATA length | Expected length of the target string |
| Replied DATA length | Actual length of the target string |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 1 | 1 | Flag1 |
| 2 | 1 | Flag2 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 to 4096 | ASCII string to be passed to the server |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 3 | 1 | Flag3 |
| 4 | 2 | Length |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 to 6827 | EBCDIC string generated by the server |

## EBCDIC-ASCII translation (EA function)

This function translates an EBCDIC string into an ASCII string.

| CPRB Field | Content/Description |
|---|---|
| Function code | EA |
| Request DATA length | Length of the source string |
| Request PARMLIST length | 26 |
| Reply DATA length | Expected length of the target string |
| Reply PARMLIST length | 26 |
| Replied DATA length | Actual length of the target string |
| Replied PARMLIST length | |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag0 |
| 1 | 1 | Flag1 |
| 2 | 1 | Flag2 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 to 4096 | EBCDIC string to be passed to the server |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 3 | 1 | Flag3 |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 4 | 2 | Length |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 0 to 4096 | ASCII string generated by the server |

## Translation routines between ASCII and EBCDIC

Translation routines enable an application to translate between ASCII and EBCDIC. They are supplied, as part of LANDP for DOS, LANDP for OS/2, and LANDP for Windows, as object modules and the application must be link-edited with them.

The routines provided perform translation from the IBM PC ASCII code page CP 850 to the IBM EBCDIC MLP code page CP 500. However, they can be modified to meet any specific need. To change the routines, the sources for them are also provided.

## Routines _AE and _EA

The routines _AE and _EA reside in the object modules CAE.OBJ and CEA.OBJ, respectively. The corresponding source modules are CAE.ASM and CEA.ASM (CAE.C and CEA.C for LANDP for Windows).

Those routines can be used from C and COBOL, although any programming language that can call a routine using the C language calling convention can be used.

The routine _AE is used to translate from ASCII to EBCDIC, and the routine _EA is used to translate from EBCDIC to ASCII. Both routines support two parameters.

The syntax for a C program (for example, VisualAge C++) is:

```
AE (data_addr, length);
EA (data_addr, length);
```

where:

**data_addr** is the *far* address of the data to be translated. The routines translate the data where it resides. Thus, the same area is used for the translated data after the routine completes.

**length** is the length of the data to be translated. The maximum amount of data which can be translated is 32KB.

**Note:** When writing a C program, do not include the underscore in the name of the routines, because the C compiler automatically appends it to the function name as it appears in the source.

The syntax for a **VisualAge COBOL** program is:

```
CALL "AE" USING
    BY REFERENCE DATA_ADDR
    BY VALUE LENGTH
END-CALL
```

or:

```
CALL "EA" USING
    BY REFERENCE DATA_ADDR
    BY VALUE LENGTH
END-CALL
```

where:

**DATA_ADDR** is the *far* address of the data to be translated. The routines translate the data where it resides. Thus, the same area is used for the translated data after the routine completes.

**LENGTH** is the length of the data to be translated. The maximum amount of data which can be translated is 32KB.

## Routines AE and EA

These routines are equivalent to the _AE and _EA routines, using the same code pages for the translation process. They are **not** supplied for LANDP for Windows. They support Pascal/2 and MASM/2, and any programming language that can meet the Pascal calling convention.

The syntax for a **Pascal/2** program is:

```
AE (data_addr, length);
EA (data_addr, length);
```

where:

**data_addr** is the *near* address of the data to be translated. The routines expect that the data resides in the default data segment at the time of the call. If not, a far pointer should be specified to access the data, and you should use the routines _AE and _EA.

**length** is the length of the data to be translated, with a maximum of 32KB.

# Chapter 26. IBM 4707 monochrome display

The IBM 4707 monochrome display is a compact 9-inch monochrome display. Because of its small size, it is suitable when space is limited.

The 4707 display provides hardware support for the three following display modes:

- *2000-mode*: displays 25 rows of 80 characters.
- *1000-mode*: displays 25 rows of 40 characters. The displayed characters are *double width* compared to the displayed characters in 2000-mode. The character height is the same in both modes.
- *480-mode*: displays 12 rows of 40 characters. The displayed characters are *double width* and *double height* compared to the displayed characters in 2000-mode. Because of the large character size, the characters fill the entire surface when in 480-mode.

**LANDP for DOS** provides a subroutine that enables switching between these three modes. The display mode can also be changed using IBM DOS commands. These are described in the *LANDP Servers and System Management* book.

A device driver is provided with the IBM 4707 monochrome display. Using the device driver is optional. If it is not installed, the 4707 display always operates in 2000-mode.

If the device driver is installed, it can be in two states: enabled and disabled. If the device driver is disabled, the 4707 display operates as if the device driver is not installed.

To be able to change 4707 display modes, the video adapter card must be one of the following:

- EGA (enhanced graphics adapter)
- VGA (video graphics array)
- MCGA (multicolor graphics array)
- Any other adapter that can emulate the subset of the necessary EGA functions

When you write applications for a 4707 display operating in either 480-mode or 1000-mode, you must take into account the number of available rows and columns. When the 4707 display operates in 1000-mode, all requests to write a character in a position to the right of the 40th column are ignored. When the 4707 display operates in 480-mode, all requests to write a character in a position below the 12th row or to the right of the 40th column are ignored.

## Changing display mode using the IBM LANDP-supplied routine

If the 4707 display device driver is installed, it is automatically activated when the LANDP program is loaded. Then the device driver status can be set to either enabled or disabled.

© Copyright IBM Corp. 1992, 2002

LANDP for DOS provides a subroutine that enables switching between the three modes available. The subroutine is invoked from an application using one of the following calls:

**CALL MODE480,x**
Is used for programming languages passing the parameters by value—for example, Macro Assembler and C.

**CALL BMODE480,x**
Is used for programming languages passing the parameters by reference—for example, BASIC.

**CALL '_mode480' using by value xx.**
Is used for COBOL/2.

**Notes:**

1. A field of 2 bytes must be defined in DATA DIVISION:

   ```
   01 xx  pic  x(2)
   ```

2. If you want to switch to 480-mode:

   ```
   move 'aa' to xx.
   ```

   or:

   ```
   move '21' to xx.
   ```

3. If you want to switch back to the previous mode:

   ```
   move low-value to xx.
   ```

The LANDP subroutine must be linked to the application using the MODE480.OBJ file, which is also supplied with LANDP for DOS.

Calling the subroutine with x ≠ X'00' results in the following sequence of events:
1. The current display state is saved
2. The device driver is enabled
3. The 480-mode is entered

Calling the subroutine with x = X'00' results in restoring the saved display state, no matter what the device driver state is. Thus, it is possible to switch between 1000-mode and 480-mode and to switch between 2000-mode and 480-mode.

If the device driver is not installed, calling the subroutine with x ≠ X'00' results in changing to 1000-mode.

If the video adapter installed is not EGA, VGA, MCGA, or an adapter emulating the necessary EGA function set, or if the LANDP program is not loaded, the results obtained when using the subroutine vary depending on the hardware installed and the device driver state.

# Chapter 27. Object postbox server and batch machines

The object postbox server allows an application program to store and retrieve messages in common queues. The messages are accessible by all LANDP workstations. Data integrity and recoverability is handled through the shared file server, which stores the data. Messages are compressed and decompressed as required.

The object postbox server organizes data messages as queues owned by single users. The user can be either a physical LANDP workstation, the batch machine loader server (see “Batch machines” on page 587), or a user defined using the system manager server. Only the owner of a data queue can read the data stored in that queue. Any user can send new object messages to any known queue.

Queues contain several entries, each consisting of a message. A message consists of two parts: a header, which defines the characteristics of the message, and the message data.

The server operates in the **LANDP for OS/2** environment.

*Table 45. Function Codes used in the Object Post-Box Server*

| Function code | Description | Env. | Page |
|---|---|---|---|
| BO | Backout operation | -2- | 580 |
| LI | List input queue | -2- | 580 |
| LO | List output queue | -2- | 581 |
| RD | Register device and check status | -2- | 583 |
| RM | Read message | -2- | 584 |
| SM | Send message and message header | -2- | 584 |
| WM | Write message data | -2- | 585 |

The first and second columns give the function code and the name of the function. The third column shows the operating environment of the function. “-2-” means that it is available from LANDP for OS/2 servers. The last column refers to the page where you can find the function described.

## Server characteristics

Object postbox transactions fall into the following categories:

**Send transactions:** The first function request is SM (send message and message header), which initiates a sequence. This is followed by several WM (write message data) functions, the last of which should generate an ED return code which indicates that the transaction closed successfully.

**Read transactions:** A sequence of RM (read message) function requests, which ends when the server returns the

© Copyright IBM Corp. 1992, 2002

code ED, can be used to read data from the queue. Three modes of reading are supported:

**Receive:** The message header is logged and the message is purged from the queue. If an acknowledgement is required, one is sent to the originator's queue.

**Peek:** The message is retrieved, but is not removed from the queue.

**Discard:** The message header is not logged, but the message is purged from the queue. If an acknowledgement is required, one is sent to the originator's queue.

**List transactions:** A sequence of LI (list input queue) or LO (list output queue) function requests gives information about what messages are stored in the application's input or output queues. An ED return code indicates that there is no more data in the queue.

**Backout operation:** A transaction can be terminated before it is finished, using the BO function. All transactions that are ended this way, and all transactions that are not successfully completed when the server or application is unloaded have all changes made backed out.

An application can have only one transaction open at a time. Also, there should be a valid user ID (known to the system manager server) logged in to the requesting application, unless the RD (register device and check status) function is used. This function can be used as a "fast query" method of knowing when to make use of the server's other functions.

The message header contains fields that identify:
- The user and LAN identifiers of the origin and destination of the message
- A date and time stamp, showing when the message was sent
- The type of message (a data message or an acknowledgement)
- Whether an acknowledgement is required
- Any additional data, other than the message itself

## Using the object postbox server

This section provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields. If you need more information about the CPRB fields, see Appendix A, "Connectivity programming request block" on page 669.

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | 6 | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 26 | 2 | 6 | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | OPBS | Server name |

The following fields are variable and are discussed in each function request description:
- Function code
- Request DATA length
- Reply DATA length

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |

If the request was successful, the *router return code* and the *server return code* are both X'00000000'. In all other cases, see the appropriate section in the *LANDP Problem Determination* book to see if you should take any action. The return values in Reply PARMLIST and DATA should be ignored if there is an error.

The following fields are variable and are discussed in each function request description:
- Replied PARMLIST length
- Replied DATA length

**PARMLIST:** The Request PARMLIST and Reply PARMLIST fields for the object postbox server are:

| Offset | Length | Content |
|---|---|---|
| 0 | 1 | Flag1 (more data flag) |
| 1 | 1 | Flag2 (qualifier flag) |
| 2 | 4 | Reference number |
| 6 | | (As required by each function) |
| **Note:** The meanings of the flags and other fields are explained in the description of each function. | | |

**DATA:** Request DATA and Reply DATA contain data to be sent to, or received from, the server. The use of these areas is explained in the description of each function call.

## Request reference

This section describes the functions that client programs can request from the object postbox server. They are listed in alphabetical order of function code.

### Backout operation (BO function)

This function allows the application to back out a read, write, or list transaction that is not yet terminated. The server rolls back any changes made to the queues by the requesting application transaction.

| CPRB Field | Content/Description |
|---|---|
| Function code | BO |
| Request DATA length | 0 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

### List input queue (LI function)

This function retrieves the message header of a pending message in the input queue of the requesting application. The retrieval can be done in time ascending order (FIFO) or time descending order (LIFO), through Flag1. Flag2 indicates whether the requested queue is the active reader queue or the messages received (log) queue.

A message number (a spool ID) indicates from which message the scan is to be done. Repeating this function provides the application with the following message in the queue:

| CPRB Field | Content/Description |
|---|---|
| Function code | LI |
| Request DATA length | 0 |
| Request PARMLIST length | 6 |
| Reply DATA length | 4096 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 26 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1. Forward or backward scan. Possible values are:<br>**'+'** Scan forwards (FIFO)<br>**'–'** Scan backwards (LIFO) |
| 1 | 1 | Flag2. Log or reader.<br>**'L'** Scan the log queue (of acknowledged messages received by other users)<br>**'R'** Scan the message queue |
| 2 | 4 | Reference number (spool ID to start the scan) |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Origin queue LAN ID |
| 8 | 8 | Origin queue user ID |
| 16 | 4 | Reference number (spool ID) |
| 20 | 8 | Destination queue LAN ID |
| 28 | 8 | Destination queue user ID |
| 36 | 1 | Hour |
| 37 | 1 | Minute |
| 38 | 1 | Second |
| 39 | 1 | Reserved |
| 40 | 1 | Day |
| 41 | 1 | Month |
| 42 | 2 | Year |
| 44 | 4 | Message data size |
| 48 | 1 | Message type<br>**'H'** Hold on (message is locked by another transaction)<br>**'R'** Reader message<br>**'L'** Log message |
| 49 | 1 | Acknowledgement<br>**'A'** Acknowledgement required (for a reader message)<br>**' '** Acknowledgement not required (for a reader message)<br>**'R'** Received message acknowledgement (for a log message)<br>**'D'** Discarded acknowledgement (for a log message) |
| 50 | 1 | Additional message header information |

## List output queue (LO function)

This function retrieves the message header of a pending message in the output queue of the requesting application. The retrieval can be done in time ascending order (FIFO) or time descending order (LIFO), through Flag1. Flag2 indicates whether the requested queue is the active reader queue or the messages sent and received (log) queue.

A message number (a spool ID) indicates from which message the scan is to be done. Repeating this function provides the application with the following message in the queue:

| CPRB Field | Content/Description |
|---|---|
| Function code | LO |
| Request DATA length | 0 |
| Request PARMLIST length | 6 |
| Reply DATA length | 4096 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 26 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1. Forward or backward scan. Possible values are:<br>**'+'** Scan forwards (FIFO)<br>**'–'** Scan backwards (LIFO) |
| 1 | 1 | Flag2. Log or reader.<br>**'L'** Scan the log queue (of acknowledged messages received by other users)<br>**'R'** Scan the message queue |
| 2 | 4 | Reference number (spool ID to start the scan) |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Origin queue LAN ID |
| 8 | 8 | Origin queue user ID |
| 16 | 4 | Reference number (spool ID) |
| 20 | 8 | Destination queue LAN ID |
| 28 | 8 | Destination queue user ID |
| 36 | 1 | Hour |
| 37 | 1 | Minute |
| 38 | 1 | Second |
| 39 | 1 | Reserved |
| 40 | 1 | Day |
| 41 | 1 | Month |
| 42 | 2 | Year |
| 44 | 4 | Message data size |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 48 | 1 | Message type<br>**'H'** Hold on (message is locked by another transaction)<br>**'R'** Reader message<br>**'L'** Log message |
| 49 | 1 | Acknowledgement<br>**'A'** Acknowledgement required (for a reader message)<br>**' '** Acknowledgement not required (for a reader message)<br>**'R'** Received message acknowledgement (for a log message)<br>**'D'** Discarded acknowledgement (for a log message) |
| 50 | 1 | Additional message header information |

## Register device and check status (RD function)

This function allows a physical user (one not defined through the system manager server) to establish and maintain its own queues. The user identifies itself through the Request DATA field, and receives an indicator of whether there are messages pending.

This function can be used as a “fast query” method of knowing when to make use of the server’s other functions.

| CPRB Field | Content/Description |
|---|---|
| Function code | RD |
| Request DATA length | 8 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 1 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 1 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 8 | Queue owner name (pseudo user ID) |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1<br>**'+'** There are messages pending<br>**' '** There are no messages pending |

## Read message (RM function)

This function retrieves a message from its queue. The message number (spool ID) must be specified, and Flag2 must be set to show whether the message is to be received, peeked, or discarded.

The server indicates, in Flag1, whether more data blocks remain to be read. Return code ED indicates that the transaction is complete.

| CPRB Field | Content/Description |
|---|---|
| Function code | RM |
| Request DATA length | 0 |
| Request PARMLIST length | 6 |
| Reply DATA length | 4096 |
| Reply PARMLIST length | 1 |
| Replied DATA length | 26 |
| Replied PARMLIST length | 1 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 1 | 1 | Flag2. Message disposal flag<br>**'D'** Discard<br>**'P'** Peek<br>**'R'** Receive |
| 2 | 4 | Spool ID to receive |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 1 | 1 | Flag1. More data flag<br>**'+'** More data<br>**' '** No more data |

| Reply DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Message data block |

## Send message and message header (SM function)

This function starts a send transaction. It indicates the destination user IDs that are to receive the message that follows, whether an acknowledgement is required, and any other data to add to the message header.

The SM function is followed by one or more WM function requests that contain the message data to be sent. It returns the spool ID of the message.

| CPRB Field | Content/Description |
|---|---|
| Function code | SM |
| Request DATA length | Up to 2048 |
| Request PARMLIST length | 6+(16×N)<br>N is the number of destination user IDs |
| Reply DATA length | 0 |
| Reply PARMLIST length | 6 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 6 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 1 | 1 | Flag2. Acknowledgement<br>**'A'** Acknowledgement required<br>**' '** Acknowledgement not required |
| 2 | 4 | Number of user IDs (N) |
| 6 | 8 | User ID 1 (destination queue user ID) |
| 14 | 8 | LAN ID 1 (destination queue LAN ID), see note |
| ...and so on until... | | |
| | 8 | User ID N |
| | 8 | LAN ID N |
| **Note:** LAN IDs other than the current one are not supported | | |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Additional message header information |

| Reply PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 2 | 4 | Message number (spool ID) |

## Write message data (WM function)

This function is used to pass message data to the server. It is repeated as necessary to supply successive blocks of message data. It is preceded by an SM request, which sets up the message header and starts the transaction.

After the last block of data is sent (shown by flag1 being ' '), an ED return code indicates that the transaction has ended, and the message is sent to the destination queues.

| CPRB Field | Content/Description |
|---|---|
| Function code | WM |
| Request DATA length | 0 |
| Request PARMLIST length | 6 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

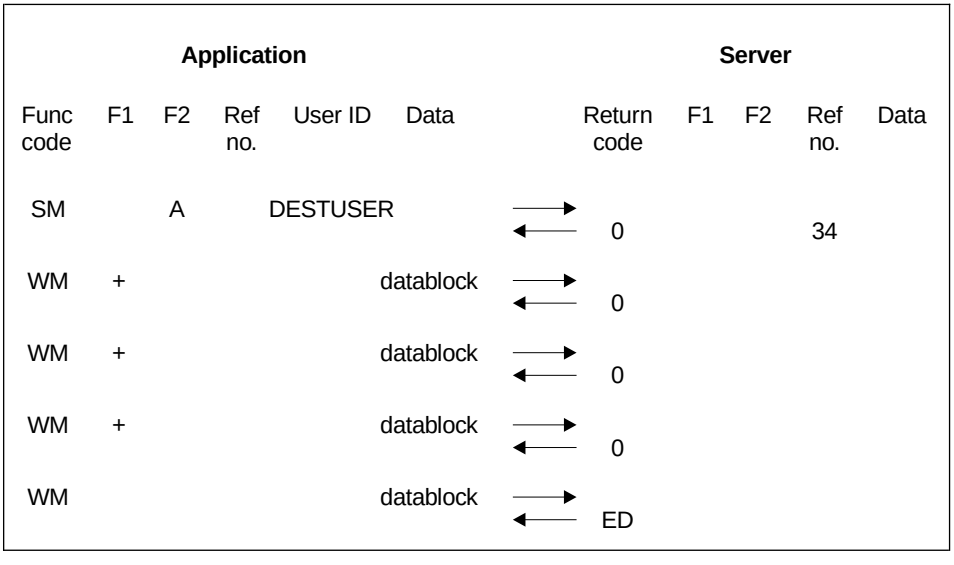
| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | Flag1.<br>**'+'** More data follows<br>**' '** No more data follows |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | | Message data block |

## Examples of use

The following examples show typical sequences of data flow.

- To send a multiblock message, which is assigned spool ID 34 by the server:



- To retrieve a multiblock message, knowing its assigned spool ID (34):

| Application | | | | | | | Server | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Func code | F1 | F2 | Ref no. | User ID | Data | | Return code | F1 | F2 | Ref no. | Data |
| RM | | R | 34 | | | → | | | | | |
| | | | | | | ← | 0 | + | | | datablock |
| RM | | | | | | → | | | | | |
| | | | | | | ← | 0 | + | | | datablock |
| RM | | | | | | → | | | | | |
| | | | | | | ← | 0 | + | | | datablock |
| RM | | | | | | → | | | | | |
| | | | | | | ← | 0 | | | | datablock |
| RM | | | | | | → | | | | | |
| | | | | | | ← | ED | | | | |

- To retrieve all message headers from the input queue, searching forwards from a given spool ID (34):

| Application | | | | | | | Server | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Func code | F1 | F2 | Ref no. | User ID | Data | | Return code | F1 | F2 | Ref no. | Data |
| LI | + | R | 34 | | | → | | | | | |
| | | | | | | ← | 0 | | | | msg hdr |
| LI | | | | | | → | | | | | |
| | | | | | | ← | 0 | | | | msg hdr |
| LI | | | | | | → | | | | | |
| | | | | | | ← | 0 | | | | msg hdr |
| LI | | | | | | → | | | | | |
| | | | | | | ← | ED | | | | |

## Batch machines

The batch machine loader server (BMLS) is a program that works in conjunction with the object postbox server. It allows a program to act as an automated machine that handles one of the object postbox server message queues. It allows you to:

- Integrate applications that deal with input and output files

- Unload work from a client machine into a server machine (which may be more powerful)
- Process batch jobs asynchronously and unattended, at a later time
- Concentrate jobs into a central place, eliminating the need to have several licenses for the same program.

If you want to process jobs in batch mode, you must write a batch or executable program that performs the unattended processing. This program must be in an accessible path as specified when loading the BMLS program. For information about the server loading statement, refer to the *LANDP Installation and Customization* book.

The batch machine loader server reads its message queue, and when it finds a pending message, it calls the specified program, using the operating system command processor.

The program you write must follow some rules in parameter passing and queue handling:

- Parameters are passed as follows:
  - %1 is the message originator user ID.
  - %2 is the file name of the message. If the message has no file name associated, it will be 'NUL'.
  - %3 is the file extension of the message. This parameter includes also the dot that precedes the file extension (thus, %3=.EXE).
  - %4 and following are any additional parameters found in the message header. These parameters are passed to your program with the `/comment` parameter of the MAIL program or from the batch machine operator **Command & parameters** entry field.
- Your program must purge messages from the message queue. If it does not purge a message, it keeps on receiving the same message. For that purpose, your program can use the commands supplied with the MAIL program.
- Your program can display output on screen only for debugging purposes. You must load the server with `START BMLS ...` instead of `LOADER BMLS ...` in order to see the messages displayed.
- Your program cannot request input from the keyboard.

## Building your programs for batch processing

LANDP provides the MAIL program commands as an aid in building your programs for batch processing. For information on the MAIL program commands, refer to the *LANDP Servers and System Management* book.

## Sample programs that use batch processing

This section presents two sample programs used for processing jobs in batch mode.

### Batch processing sample 1

The first program is a print spooler sample, SPOOL.CMD:

```
@echo off
mail receive %1\\%2%3 lpt1
```

The required server loading statement is

```
LOADER BMLS /N:SPOOLPR /P:SPOOL.CMD
```

You can send your requests to SPOOL.CMD in the following way:

```
MAIL SEND MYFILE.DOC SPOOLPR
```

The program receives the message file and sends it to the printer, purging it from the queue at the same time.

## Batch processing sample 2

This sample program compiles a program remotely. It waits for incoming message files until a file with .MAK extension arrives. After that, the batch program starts compiling and upon completion sends back an .EXE file to the originator userid. The program name is MAKEPROJ.CMD.

```
@echo off
rem ********************************************************
rem * BATMAKE remote batch compiler                        *
rem * Assumes:                                             *
rem * - MAIL accessible from FILESERV exec dir             *
rem * - PKUNZIP accessible from path directory             *
rem * - Receives:                                          *
rem *      - Either a X.ZIP file in SAVERAM format         *
rem *        with required files and a X.MAK included      *
rem *      - Or packages of files, being the last file     *
rem *        any .MAK file to invoke.                      *
rem * - The executed .MAK file can redirect commands       *
rem *   output to .INF files so that they are sent back    *
rem ********************************************************
if exist %1\. goto a
md %1
:a
cd %1
··\mail receive %1\\%2%3 >nul
if errorlevel 1 goto x
if %3.==.MAK. goto p
if %3.==.ZIP. goto c
goto b
:c
pkunzip %2%3
if exist %2.mak goto p
goto b
:p
call make /x make.inf %2.mak
if errorlevel 1 goto y
del make.inf
:y
··\mail send *.exe %1   >nul
:d
··\mail send *.inf %1   >nul
```

```
echo y|del *.*
cd..
rd %1
goto b
:x
echo BATMAKE: Could not receive %2%3 >batmake.inf
goto d
:b
```

The required server loading statement is

```
LOADER BMLS /N:BATMAKE /P:MAKEPROJ.CMD /D:\WORKDIR
```

You can send your requests to MAKEPROJ.CMD in the following way:

```
MAIL SEND FILE1.H BATMAKE
MAIL SEND FILE2.H BATMAKE
MAIL SEND FILE3.H BATMAKE
MAIL SEND FILE1.C BATMAKE
MAIL SEND FILE1.MAK BATMAKE
```

The program performs the following processes:

- It creates a directory with the name of the userid that sent the files.
- It receives each incoming file in the userid private directory. If it is a .ZIP file, unpacks it. If there is a .MAK file inside, the program builds the project.
- If there is no .MAK file, it stops processing because the .MAK file probably arrives later on.
- After building the project (invoking MAKE), it puts all the resulting data into a file, and sends back the compiled modules and the .INF file with the building messages. After that, it erases the working directory.

# Part 8. IBM LANDP emulators

**LANDP for DOS** provides two emulators:
- IBM 3270 emulator
- IBM 3287 emulator

LANDP for OS/2 and LANDP for Windows allow both emulators to run also in OS/2 and Windows VDMs (virtual DOS machines).

The LANDP 3270 emulator is used to operate a LANDP for DOS workstation as if it were an IBM 3270 Information Display System terminal. The emulator communicates with a host application program through SNA/SDLC, SNA/Token-Ring, or SNA/X.25, using SNA LU 2 protocols. It emulates a subset of the IBM 3274 Model 51C Control Unit functions.

Up to five 3270 emulators communicating with one or more host computers can be installed in one LANDP for DOS workstation.

The 3270 emulator can be accessed by an application program using the 3270 emulator high-level or low-level API.

The user can start the 3270 emulator with an application program or by using a hot key.

This part of the book has the following chapters:

**Chapter 28, "LANDP 3270 emulator API" on page 593**
This chapter presents the *low-level* application programming interface that allows application programs to request services from the LANDP for DOS 3270 Emulator.

**Chapter 29, "LANDP 3270 emulator high-level language API" on page 613**
This chapter presents the *high-level* application programming interface that allows application programs to request services from the LANDP for DOS 3270 Emulator.

**Chapter 30, "LANDP 3287 printer emulator API" on page 657**
The 3287 printer emulator uses the SNA LU 1 capabilities of the SNA/SDLC, SNA/Token-Ring, or SNA/X.25 server. One 3287 printer emulator supports up to five logical units (LU 1), communicating with one or more host computers.

3287 emulation is compatible with local use of the printer.

This chapter presents the Common Application Programming Interface function calls corresponding to the LANDP for DOS 3287 Printer Emulator.

© Copyright IBM Corp. 1992, 2002

# Chapter 28. LANDP 3270 emulator API

This emulator operates in the **LANDP for DOS** environment.

Two ways are available to interact with host computer application programs written for the IBM 3270 Information Display System environment:
1. The workstation is used as a 3270 terminal, through the 3270 emulator.
2. A workstation application program uses one of the 3270 emulator APIs.

This chapter describes the low-level 3270 emulator application programming interface. The high-level API is described in Chapter 29, “LANDP 3270 emulator high-level language API”.

Two functions of the local resource manager can be used with the 3270 emulator.

*Table 46. Function Codes used in the Local Resource Manager—functions used with the 3270 emulator low-level interface*

| Function code | Description | Env. | Page |
|---|---|---|---|
| EC | End connection | 0-- | 519 |
| GS | Get status | 0-- | 520 |

The first and second columns give the function code and the name of the function. The third column shows the operating environment of the function. “0--” means that it is available from LANDP for DOS servers. The last column refers to the page where you can find the function described.

## Characteristics of the low-level language API

The 3270 emulator API allows workstation application program interaction with host computer application programs written for the IBM 3270 Information Display System environment. A workstation application program using the 3270 emulator API can:

- Initiate and control the host computer application program access
- Use the display presentation space of the 3270 emulator
- Use most bit specifications of the operator information area (OIA)
- Enter key strokes representing commands and messages to the host computer

### Invocation of the API

All calls to the 3270 emulator API are initiated by the workstation application program following the steps below:
1. The system registers are loaded with the service specific values.
2. The parameters are set to specified values.
3. The software interrupt X'7A' is issued.

All service requests are processed synchronously. That is, once processing has started, control is not returned until the call is completed.

© Copyright IBM Corp. 1992, 2002

The values to be inserted in the system registers and in the parameter lists are discussed for each function.

## API services

The 3270 emulator application programming services needed to control access to 3270 host computer application programs are grouped into *gates*. These services are available at each gate:

**SESSMGR**
These session information services:
- Query session ID
- Query session parameters
- Query session cursor

**KEYBOARD**
These keyboard services:
- Connect to emulator keyboard
- Disconnect from emulator keyboard
- Enable input
- Disable input
- Write keystrokes

**COPY** This copy service:
- Copies a string

**OIAM** This operator information area service:
- Reads the operator information area

The 3270 emulator API provides two kinds of return codes: system return codes and service return codes.

The system return codes are common to all service calls, and they are always returned in the CL register. Those return codes are presented in the table below.

| **System Return Codes** | |
|---|---|
| **Code** | **Explanation** |
| X'00' | Call accepted |
| X'05' | Invalid value specified in the DX register |
| X'07' | Invalid value specified in the BH register |
| X'08' | Invalid value specified in the BL register |
| X'2F' | Invalid value specified in the AH register |
| X'34' | Invalid value specified in the AL register |

The service return codes are always returned in the first byte of the parameter list. A X'00' return code means successful completion. Other return codes are presented in each service section.

## Query gate ID

When using the 3270 emulator API, the first step is to obtain the *gate ID*. The gate ID is a 2-byte identification number, used in all calls for a service available at that gate.

The call to obtain a gate ID is issued only once for each gate. To obtain a gate ID, you issue a call as described in “Invocation of the API” on page 593. The corresponding register values are shown in the following table:

| **Register values when calling:** | | | |
|---|---|---|---|
| **AH** | X'81' | **ES**<br>**DI** | Segment address of the parameter list<br>Offset address of the parameter list |
| **Register values on completion:** | | | |
| **BH**<br>**CH** | X'07'<br>X'12' | **CL** | System return codes<br>**X'00'** Successfully completed<br>**X'2E'** Invalid gate name<br>**X'2F'** Invalid AH register specified |
| | | **DX** | Gate ID |

The parameter list associated with this function is an 8-byte area containing the gate name in ASCII characters, padded to the right with blanks:

```
'SESSMGR '
'KEYBOARD'
'COPY    '
'OIAM    '
```

## Session information services

The session information service calls allow the application program to obtain the 3270 emulation display session ID, session parameters, and current session cursor position.

### Query session ID

The query session ID service obtains the session ID, which is a required parameter for most calls. A session can be specified by its short or its long name. By using this service, you can also obtain the IDs of all the host computer communication sessions.

| **Register values when calling:** | | | |
|---|---|---|---|
| **AH**<br>**AL**<br>**BH**<br>**BL** | X'09'<br>X'01'<br>X'80'<br>X'20' | **CX**<br>**DX**<br>**ES**<br>**DI** | X'0000'<br>Gate ID for SESSMGR<br>Segment address of the parameter list<br>Offset address of the parameter list |
| **Register values on completion:** | | | |
| **CH** | X'12' | **CL** | System return code. See table on page 594 |

The parameter list format is:

| Offset | Length | Contents when Calling | Contents on Completion |
|---|---|---|---|
| 0 | 1 | Must be zero | Service return code, see below |
| 1 | 1 | Must be zero | Function code (X'6B') |
| 2 | 1 | Option code | Unchanged |
| 3 | 1 | Data code | Unchanged |
| 4 | 2 | Offset address of name array | Unchanged |
| 6 | 2 | Segment address of name array | Unchanged |
| 8 | 8 | Session long name | Reserved |

The service return codes are:

| Service Return Codes | |
|---|---|
| **Code** | **Explanation** |
| X'03' | Specified long name not valid |
| X'09' | Data code not valid |
| X'0B' | Specified short name not valid |
| X'0C' | Byte 0 of the parameter list not zero when calling |
| X'0D' | Option code not valid |
| X'12' | Name array length not valid |

To obtain the session ID using the session short name, the parameter values must meet the following:
- The option code must be X'01'.
- The data code must be the 1-character ASCII name of the session—that is, the short name of the session. Accepted values are ’1’ to ’5’. For compatibility with previous releases, the character ’E’ is accepted and handled as the character ’1’.

**Note:** Here, the session long name is ignored.

To obtain the session ID using the session long name, the parameter values must satisfy the following criteria:
- The option code must be X'01'.
- The data code must be X'00'.
- Bytes 8 through 15 must contain the long name of the session, padded to the right with blanks.

To obtain the session ID for all the sessions, the parameter values must satisfy these criteria:
- The option code must be X'00'

- The data code must be X'02'

**Note:** Here, the session long name is ignored.

The name array format is:

| Offset | Length | Contents when Calling | Contents on Completion |
|---|---|---|---|
| 0 | 1 | Name array length | Unchanged |
| 1 | 1 | Not used | Number of sessions |
| 2 | 1 | Not used | Short name for session 1 |
| 3 | 1 | Not used | Type of session 1 (X'02') |
| 4 | 1 | Not used | Session ID 1 |
| 5 | 1 | Reserved | Reserved |
| 6 | 8 | Reserved | Session 1 long name |

**Notes:**

1. The name array format corresponding to offset 2 through 13 must be repeated for all the defined sessions.
2. If more than one session long name is present, it is stored in offset 14 through 21, 22 through 29, and so on.

The *name array length* is the number of bytes in the name array. It must be at least 14, and no more than 170. If this service is used to obtain the session ID for all the sessions, the *name array length* must be large enough for all of them to be returned.

Regarding the name array parameters on completion, note the following:

- The *number of sessions* contains the number of sessions returned.
- The *short name* for the session is the one-character ASCII name of the session.
- The *session ID* is used to identify the session, and must match the SNA session ID. It has to be used in any following request.
- The *session long name* is the eight-character ASCII name assigned to the session. The *session long name* is padded to the right with blanks, if necessary.

## Query session parameters

The query session parameters service returns parameters to be used in the copy string service.

| **Register values when calling:** | | | |
|---|---|---|---|
| **AH** | X'09' | **CX** | X'0000' |
| **AL** | X'02' | **DX** | Gate ID for SESSMGR |
| **BH** | X'80' | **ES** | Segment address of the parameter list |
| **BL** | X'20' | **DI** | Offset address of the parameter list |
| **Register values on completion:** | | | |
| **CH** | X'12' | **CL** | System return code. See table on page594 |

The parameter list format is:

| Offset | Length | Contents when Calling | Contents on Completion |
|---|---|---|---|
| 0 | 1 | Must be zero | Service return code, see below |
| 1 | 1 | Must be zero | Function code (X'6B') |
| 2 | 1 | Session ID | Unchanged |
| 3 | 1 | Reserved | Reserved |
| 4 | 1 | Not used | Session type. X'02' means host computer session |
| 5 | 1 | Not used | Session characteristics (X'00') |
| 6 | 1 | Not used | Rows |
| 7 | 1 | Not used | Columns |
| 8 | 2 | Reserved | Reserved |
| 10 | 2 | Reserved | Reserved |

The service return codes are:

| Service Return Codes | |
|---|---|
| **Code** | **Explanation** |
| X'02' | Specified session ID not valid |
| X'0C' | Byte 0 of the parameter list not zero when calling |

The session characteristics parameter, X'00', has the following meaning:

**Bit 7 (EAB) = 0** No extended attributes.

**Bit 6 (PSS) = 0** No programmed symbols.

You should use the row and column values returned by this call to compute the maximum offset to be specified on a copy string call.

## Query session cursor

The query session cursor service obtains the session *cursor type*, and the row and column addresses.

| Register values when calling: | | | |
|---|---|---|---|
| **AH** | X'09' | **CX** | X'00FF' |
| **AL** | X'0B' | **DX** | Gate ID for SESSMGR |
| **BH** | X'80' | **ES** | Segment address of the parameter list |
| **BL** | X'20' | **DI** | Offset address of the parameter list |
| **Register values on completion:** | | | |
| **CH** | X'12' | **CL** | System return code. See table on page 594 |

The parameter list format is:

| Offset | Length | Contents when Calling | Contents on Completion |
|---|---|---|---|
| 0 | 1 | Must be zero | Service return code, see below |
| 1 | 1 | Must be zero | Function code (X'6B') |
| 2 | 1 | Session ID | Unchanged |
| 3 | 1 | Not used | Cursor type |
| 4 | 1 | Not used | Row address |
| 5 | 1 | Not used | Column address |

The service return codes are:

| Service Return Codes | |
|---|---|
| **Code** | **Explanation** |
| X'02' | Specified session ID not valid |
| X'0C' | Byte 0 of the parameter list not zero when calling |

The cursor type can be one of the following:
**X'00'** Underscore cursor
**X'01'** Box cursor

The *row address* is the address of the row where the cursor is, relative to zero, in the presentation space.

The *column address* is the address of the column where the cursor is, relative to zero, in the presentation space.

## Keyboard services

The keyboard service calls allow the application program to perform the following:
- Connect to and disconnect from the emulator keyboard
- Enable and disable input
- Write keystrokes

### Connect to emulator keyboard

The connect to emulator keyboard service, along with the disconnect from emulator keyboard service, serializes access to the keyboard and copy services. It avoids interference with other application programs, and prevents the workstation operator from interfering.

The connect to emulator keyboard service must be called before calling any other keyboard service or any copy service. The return code from a connect to emulator keyboard service call must be X'00'. If not, you must not attempt to issue another keyboard service or a copy service call. You must reissue the connect to emulator keyboard service call until you receive a X'00' return code for successful completion.

| Register values when calling: | | | |
|---|---|---|---|
| **AH** | X'09' | **CX** | X'0000' |
| **AL** | X'01' | **DX** | Gate ID for KEYBOARD |
| **BH** | X'80' | **ES** | Segment address of the parameter list |
| **BL** | X'20' | **DI** | Offset address of the parameter list |
| **Register values on completion:** | | | |
| **CH** | X'12' | **CL** | System return code. See table on page 594 |

The parameter list format is:

| Offset | Length | Contents when Calling | Contents on Completion |
|---|---|---|---|
| 0 | 1 | Must be zero | Service return code, see below |
| 1 | 1 | Must be zero | Function code (X'62') |
| 2 | 1 | Session ID | Unchanged |
| 3 | 1 | Reserved | Reserved |
| 4 | 2 | Must be zero | Unchanged |
| 6 | 2 | Must be zero | Unchanged |
| 8 | 1 | Must be zero | Unchanged |
| 9 | 1 | Reserved | Reserved |

The service return codes are:

| Service Return Codes | |
|---|---|
| **Code** | **Explanation** |
| X'01' | Byte 8 of the parameter list not zero when calling |
| X'02' | Specified session ID not valid |
| X'04' | Session already connected for keyboard services |
| X'0C' | Byte 0 of the parameter list not zero when calling |

## Disconnect from emulator keyboard

The disconnect from emulator keyboard service, along with the connect to emulator keyboard service, serializes access to the keyboard and copy services. It avoids interference with other application programs.

The disconnect from emulator keyboard service must be called when a programming error occurs or after:

- Entering the application program keystroke data
- Finishing the keyboard services usage
- Finishing the copy services usage

| Register values when calling: |
|---|

| | | | |
|---|---|---|---|
| **AH** | X'09' | **CX** | X'0000' |
| **AL** | X'02' | **DX** | Gate ID for KEYBOARD |
| **BH** | X'80' | **ES** | Segment address of the parameter list |
| **BL** | X'20' | **DI** | Offset address of the parameter list |
| **Register values on completion:** | | | |
| **CH** | X'12' | **CL** | System return code. See table on page 594 |

The parameter list format is:

| Offset | Length | Contents when Calling | Contents on Completion |
|---|---|---|---|
| 0 | 1 | Must be zero | Service return code, see below |
| 1 | 1 | Must be zero | Function code (X'62') |
| 2 | 1 | Session ID | Unchanged |
| 3 | 1 | Reserved | Reserved |
| 4 | 2 | Must be zero | Unchanged |

The service return codes are:

| **Service Return Codes** | |
|---|---|
| **Code** | **Explanation** |
| X'02' | Specified session ID not valid |
| X'04' | Session not connected for keyboard services |
| X'0C' | Byte 0 of the parameter list not zero when calling |

## Write keystroke

The write keystroke service sends keystroke data. Either a single keystroke or a list of keystrokes can be sent. When sending a list of keystrokes, note the following:

- Up to 255 keystrokes can be present in a list.
- A rejected keystroke cancels the processing of the rest of the list. A keystroke is rejected either when it is not valid, or when it cannot be processed because of an input inhibited condition.
- An attention identifier (AID) key can be present in a list, only if it is the last item in the list. An AID key cancels the processing of the rest of the list.

  When used in a 3270 host computer session, the AID keys are those that cause immediate interaction with the host computer.
- A parameter on completion contains the number of keystrokes actually sent. It is also set if the service return code is X'10' or X'12'. In that case, the number includes the keystrokes sent until the first AID key was found, its corresponding keystroke also included, or until the first rejected key.

| **Register values when calling:** |
|---|

| | | | |
|---|---|---|---|
| **AH** | X'09' | **CX** | X'0000' |
| **AL** | X'04' | **DX** | Gate ID for KEYBOARD |
| **BH** | X'80' | **ES** | Segment address of the parameter list |
| **BL** | X'20' | **DI** | Offset address of the parameter list |
| **Register values on completion:** | | | |
| **CH** | X'12' | **CL** | System return code. See table on page 594 |

To send a single keystroke, you must adhere to the following parameter list format:

| Offset | Length | Contents when Calling | Contents on Completion |
|---|---|---|---|
| 0 | 1 | Must be zero | Service return code, see below |
| 1 | 1 | Must be zero | Function code (X'62') |
| 2 | 1 | Session ID | Unchanged |
| 3 | 1 | Reserved | Reserved |
| 4 | 2 | Must be zero | Unchanged |
| 6 | 1 | Send a single keystroke (X'20') | Unchanged |
| 7 | 1 | Not used | Number of keys sent |
| 8 | 1 | Key scan code | Unchanged |
| 9 | 1 | Shift state | Unchanged |

To send a list of keystrokes, you must adhere to the following parameter list format:

| Offset | Length | Contents when Calling | Contents on Completion |
|---|---|---|---|
| 0 | 1 | Must be zero | Service return code, see below |
| 1 | 1 | Must be zero | Function code (X'62') |
| 2 | 1 | Session ID | Unchanged |
| 3 | 1 | Reserved | Reserved |
| 4 | 2 | Must be zero | Unchanged |
| 6 | 1 | Send a list of keystrokes (X'30') | Unchanged |
| 7 | 1 | Not used | Number of keys sent |
| 8 | 2 | Offset address of a list of keystrokes | Unchanged |
| 10 | 2 | Segment address of a list of keystrokes | Unchanged |

The keystroke list format is:

| Offset | Length | Contents |
|---|---|---|
| 0 | 2 | Twice the number of keys to send (if odd, rounded down) |
| 2 | 1 | Scan code of first key |
| 3 | 1 | Shift state of first key |
| 4 | 1 | Scan code of second key |
| 5 | 1 | Shift state of second key |
| ... | | |
| 2n | 1 | Scan code of *n*th key |
| 2n+1 | 1 | Shift state of *n*th key |

Keystrokes that are sent are identified by a 2-byte value. The first byte is the *scan code* and the second byte is the *shift state*.

The *scan code* is a unique hexadecimal value that is assigned to each key supported by the 3270 emulator API. How this byte is interpreted depends on its shift state. If the *shift state* shows ASCII or 3270 Device Buffer Formats, the *scan code* is the code actually sent. If the keystroke to be sent is a local 3270 function (for example, TAB) or an AID key (for example, ENTER), the *scan code* is the scan code for the key that represents that function on a personal computer system U.S. keyboard. The *scan codes* supported are:

| Scan code | Base | Alt | Scan code | Base | Alt |
|---|---|---|---|---|---|
| X'67' | PA1 | Invalid | X'28' | PF17 | Invalid |
| X'6E' | PA2 | Invalid | X'30' | PF18 | Invalid |
| X'06' | Clear | Invalid | X'38' | PF19 | Invalid |
| X'58' | Enter | Invalid | X'40' | PF20 | Invalid |
| X'07' | PF1 | Invalid | X'48' | PF21 | Invalid |
| X'0F' | PF2 | Invalid | X'50' | PF22 | Invalid |
| X'17' | PF3 | Invalid | X'57' | PF23 | Invalid |
| X'1F' | PF4 | Invalid | X'5F' | PF24 | Invalid |
| X'27' | PF5 | Invalid | X'05' | Invalid | SysReq |
| X'2F' | PF6 | Invalid | X'0B' | Erase EOF | Erase Input |
| X'37' | PF7 | Invalid | X'63' | Cursor up | Invalid |
| X'3F' | PF8 | Invalid | X'60' | Cursor down | Invalid |
| X'47' | PF9 | Invalid | X'61' | Cursor left | Invalid |
| X'4F' | PF10 | Invalid | X'6A' | Cursor right | Invalid |
| X'56' | PF11 | Invalid | X'62' | Invalid | Home |
| X'5E' | PF12 | Invalid | X'0D' | Tab | Invalid |
| X'08' | PF13 | Invalid | X'64' | Back Tab | Invalid |
| X'10' | PF14 | Invalid | X'5A' | New Line | Invalid |
| X'18' | PF15 | Invalid | X'03' | Invalid | Cursor select |
| X'20' | PF16 | Invalid | X'11' | Reset | Invalid |

All keys in the left column and the PF keys of the right column are AID keys. `SysReq` and `Cursor select` can be AID keys. The *shift state* shows which functions of a given *scan code* are sent. The *shift state* format is:

**Bits 7, 6** Type of keystroke
- **B'00'** Scan code
- **B'01'** ASCII
- **B'10'** 3270 Device Buffer Formats
- **B'11'** Not valid

**Bits 5, 4, 3** Ignored

**Bit 2** Alt key

**Bits 1, 0** Ignored

The service return codes are:

| **Service Return Codes** | |
|---|---|
| **Code** | **Explanation** |
| X'01' | Option byte not valid |
| X'02' | Specified session ID not valid, or list of keystrokes longer than 255 bytes |
| X'04' | Session not connected for keyboard services |
| X'0C' | Byte 0 of the parameter list not zero when calling |
| X'10' | Process stopped because scan code not valid or input inhibited |
| X'12' | Successful completion: AID generated |

## Disable input

Disables the hot-key defined to enter a 3270 emulator. Pressing the hot-key results in a "beep" to inform the workstation operator that input is disabled.

| **Register values when calling:** | | | |
|---|---|---|---|
| **AH** | X'09' | **CX** | X'0000' |
| **AL** | X'05' | **DX** | Gate ID for KEYBOARD |
| **BH** | X'80' | **ES** | Segment address of the parameter list |
| **BL** | X'20' | **DI** | Offset address of the parameter list |
| **Register values on completion:** | | | |
| **CH** | X'12' | **CL** | System return code. See table on page 594 |

The parameter list format is:

| Offset | Length | Contents when Calling | Contents on Completion |
|---|---|---|---|
| 0 | 1 | Must be zero | Service return code, see below |
| 1 | 1 | Must be zero | Function code (X'62') |
| 2 | 1 | Session ID | Unchanged |

| Offset | Length | Contents when Calling | Contents on Completion |
|---|---|---|---|
| 3 | 1 | Reserved | Reserved |
| 4 | 2 | Must be zero | Unchanged |

The service return codes are:

| Service Return Codes | |
|---|---|
| **Code** | **Explanation** |
| X'02' | Specified session ID not valid |
| X'04' | Session not connected for keyboard services |
| X'0C' | Byte 0 of the parameter list not zero when calling |

## Enable input

Allows workstation operator input for a 3270 emulator session. Workstation operator input for a session is also enabled through a disconnect from emulator keyboard service call (see “Disconnect from emulator keyboard” on page 600), if the input is disabled because of a disable input service call (see “Disable input” on page 604).

| **Register values when calling:** | | | |
|---|---|---|---|
| **AH** | X'09' | **CX** | X'0000' |
| **AL** | X'06' | **DX** | Gate ID for KEYBOARD |
| **BH** | X'80' | **ES** | Segment address of the parameter list |
| **BL** | X'20' | **DI** | Offset address of the parameter list |
| **Register values on completion:** | | | |
| **CH** | X'12' | **CL** | System return code. See table on page 594 |

The parameter list format is:

| Offset | Length | Contents when Calling | Contents on Completion |
|---|---|---|---|
| 0 | 1 | Must be zero | Service return code, see below |
| 1 | 1 | Must be zero | Function code (X'62') |
| 2 | 1 | Session ID | Unchanged |
| 3 | 1 | Reserved | Reserved |
| 4 | 2 | Must be zero | Unchanged |

The service return codes are:

| Service Return Codes | |
|---|---|
| **Code** | **Explanation** |
| X'02' | Specified session ID not valid |

| Service Return Codes | |
|---|---|
| **Code** | **Explanation** |
| X'04' | Session not connected for keyboard services |
| X'0C' | Byte 0 of the parameter list not zero when calling |

## Copy service

The copy service calls allow the application program to copy strings.

### Copy string

The copy string service copies strings of data from the application program buffer to the presentation space, or from the presentation space to the application program buffer. The service is complete when the data is moved physically. This service does not move the cursor.

You should issue a connect to emulator keyboard service call (see “Connect to emulator keyboard” on page 599) to provide protection against another application program that uses the 3270 emulator API and thus, might be calling the copy string service at the same time.

| **Register values when calling:** | | | |
|---|---|---|---|
| **AH** | X'09' | **CX** | X'00FF' |
| **AL** | X'01' | **DX** | Gate ID for COPY |
| **BH** | X'80' | **ES** | Segment address of the parameter list |
| **BL** | X'20' | **DI** | Offset address of the parameter list |
| **Register values on completion:** | | | |
| **CH** | X'12' | **CL** | System return code. See table on page 594 |

To copy from the presentation space to an application program buffer, you should follow these steps:

1. Insert the offset and segment addresses of the application program buffer in the target buffer fields.
2. Assign value to copy mode:
   **X'00'** Field attributes not copied. The field attributes are set to blanks (X'10' for 3270 Device Buffer Formats, X'20' for ASCII format).
   **X'40'** Field attributes copied. The field attributes always range from X'C0' to X'FF'.
3. Insert values for source start offset and source end offset.

   If the start or end offsets are outside this range, or if the start offset is greater than the end offset, no copy is made and a nonzero return code is generated.
4. Insert value for target start offset.
5. Choose values for target session type:
   **X'02'** Host computer session type. 3270 Device Buffer Formats.
   **X'05'** Personal computer system session type. ASCII format.

Each character in the presentation space is transferred as two characters, the first is the ASCII character of the data and the second is zero.

The parameter list format is:

| Offset | Length | Contents when Calling | Contents on Completion |
|---|---|---|---|
| 0 | 1 | Must be zero | Service return code, see below |
| 1 | 1 | Must be zero | Function code (X'64') |
| 2 | 1 | Session ID | Unchanged |
| 3 | 1 | Reserved | Reserved |
| 4 | 2 | X'0000' | Unchanged |
| 6 | 2 | X'0000' | Unchanged |
| 8 | 1 | Source characteristics (X'00') | Unchanged |
| 9 | 1 | Source session type | Unchanged |
| 10 | 2 | Source start offset | Unchanged |
| 12 | 2 | Source end offset | Unchanged |
| 14 | 1 | X'00' | Unchanged |
| 15 | 1 | Reserved | Reserved |
| 16 | 2 | Offset address of target buffer | Unchanged |
| 18 | 2 | Segment address of target buffer | Unchanged |
| 20 | 1 | Target characteristics | Unchanged |
| 21 | 1 | Target session type | Unchanged |
| 22 | 2 | Target start offset | Unchanged |
| 24 | 1 | Copy mode | Unchanged |
| 25 | 2 | Reserved | Reserved |

To copy from an application program buffer to the presentation space, you should follow these steps:

1. Insert into the source buffer fields the offset and segment addresses of the application program buffer.
2. Insert values for the source start offset and the source end offset.

   If the source start offset is greater than the source end offset, no copy is made and a nonzero return code is generated.
3. Insert a value for the target start offset.

   If the start source and end source offsets cause the copy to extend from the start target offset beyond the end of the presentation space, the data is copied to the end of the presentation space and a code indicating truncation is returned.
4. Choose values for the source session type:

**X'02'** Host computer session type. 3270 Device Buffer Formats.
**X'05'** Personal computer system session type. ASCII format.

Each character in the application program buffer is composed of one data byte followed by one attribute byte. The attribute byte is ignored.

**Note:** 3270 field attributes in the presentation space cannot be directly modified by data in the application program buffer. The 3270 field attributes are indirectly modified because the copy of a character into the field corresponding to a certain field attribute causes the modified data tag indicator to set in that 3270 field attribute.

The parameter list format is:

| Offset | Length | Contents when Calling | Contents on Completion |
|---|---|---|---|
| 0 | 1 | Must be zero | Service return code, see below |
| 1 | 1 | Must be zero | Function code (X'64') |
| 2 | 1 | X'00' | Unchanged |
| 3 | 1 | Reserved | Reserved |
| 4 | 2 | Offset address of source buffer | Unchanged |
| 6 | 2 | Segment address of source buffer | Unchanged |
| 8 | 1 | Source characteristics (X'00') | Unchanged |
| 9 | 1 | Source session type | Unchanged |
| 10 | 2 | Source start offset | Unchanged |
| 12 | 2 | Source end offset | Unchanged |
| 14 | 1 | Session ID | Unchanged |
| 15 | 1 | Reserved | Reserved |
| 16 | 2 | X'0000' | Unchanged |
| 18 | 2 | X'0000' | Unchanged |
| 20 | 1 | Target characteristics | Unchanged |
| 21 | 1 | Target session type | Unchanged |
| 22 | 2 | Target start offset | Unchanged |
| 24 | 1 | X'00' | Unchanged |
| 25 | 2 | Reserved | Reserved |

The service return codes are:

| Service Return Codes | |
|---|---|
| **Code** | **Explanation** |
| X'02' | Specified session ID not valid, copy not performed |
| X'03' | Target window is input inhibited, copy not performed |
| X'06' | Missing or invalid source parameters, copy not performed |
| X'07' | Missing or invalid target parameters, copy not performed |
| X'09' | Data truncated, copy performed |
| X'0C' | Byte 0 of the parameter list not zero when calling, copy not performed |
| X'0E' | Target window is protected, copy performed for the unprotected locations |
| X'0F' | Copy of field attributes not allowed, copy not performed |

## Operator information area service

The operator information area service calls allow the application program to obtain the status of the operator information area (OIA).

### Read operator information area

The read operator information area service returns a string of bits that reflect the settings of the OIA input inhibited indicators for the session. These indicators represent the current input inhibited states. An application program can use this information to wait for the removal of all input inhibited conditions.

| **Register values when calling:** | | | |
|---|---|---|---|
| **AH** | X'09' | **CX** | X'00FF' |
| **AL** | X'02' | **DX** | Gate ID for OIAM |
| **BH** | X'80' | **ES** | Segment address of the parameter list |
| **BL** | X'20' | **DI** | Offset address of the parameter list |
| **Register values on completion:** | | | |
| **CH** | X'12' | **CL** | System return code. See table on page 594 |

The parameter list format is:

| Offset | Length | Contents when Calling | Contents on Completion |
|---|---|---|---|
| 0 | 1 | Must be zero | Service return code, see below |
| 1 | 1 | Must be zero | Function code (X'6D') |
| 2 | 1 | Session ID | Unchanged |
| 3 | 1 | Reserved | Reserved |
| 4 | 2 | Offset address of buffer | Unchanged |
| 6 | 2 | Segment address of buffer | Unchanged |
| 8 | 1 | OIA group (X'08') | Unchanged |

The service return codes are:

| Service Return Codes | |
|---|---|
| **Code** | **Explanation** |
| X'02' | Specified session ID not valid |
| X'0C' | Byte 0 of the parameter list not zero when calling |

The bit string returned by the read operator information area service is located in the buffer pointed by the address specified in bytes 4 through 7 in the calling parameter list. The buffer size must be 5 bytes. The input inhibited states are ordered so that the high order bits represent the indicators of higher priority. The bit values in the allocated buffer are:

| Byte | Bit | Value |
|---|---|---|
| 0 | 7-6<br>5<br>4<br>3<br>2<br>1<br>0 | Reserved<br>Machine check<br>Communications check<br>Program check<br>Retry<br>Device not working<br>Reserved |
| 1 | 7<br>6<br>5<br>4<br>3<br>2-1<br>0 | Reserved<br>Terminal wait<br>Reserved<br>Minus function<br>Too much entered<br>Reserved<br>Numeric field |
| 2 | 7-4<br>3<br>2-0 | Reserved<br>Wrong place<br>Reserved |
| 3 | 7-6<br>5<br>4-0 | Reserved<br>System wait<br>Reserved |
| 4 | 7<br>6<br>5-0 | Reserved<br>Input disabled by application program<br>Reserved |

## Example using 3270 emulator API

Here is an example of how to use the 3270 emulator API:

```
Verify_3270API_Availability
Obtain_Gate-IDs
Query_Session_ID_&_parameters;
Connect_to_Keyboard
  (protects against interference of a second application)
Disable_Operator_Input
  (provides protection against operator interference)
```

```
do { Copy_String_from_appl_to_3270
     Write_Keystrokes_Enter_Action
     do {
          do { Read_OIA
               if NOT input_inhibited in OIA
                  then { Copy_String_from_3270_to_appl
                         Determine_if_response_received
                       }
             } until NOT input_inhibited in OIA
                     OR timeout
                     OR operator_disconnect
        } until response_received AND NOT input_inhibited
                OR timeout
                OR operator_disconnect
   } until no_more_calls
Disconnect_from_Keyboard (releases protection)
```

# Chapter 29. LANDP 3270 emulator high-level language API

This emulator operates in the **LANDP for DOS** environment and can also be run in the DOS MVDM under LANDP for OS/2 or LANDP for Windows.

Two ways are available to interact with host computer application programs written for the IBM 3270 Information Display System environment:

- The workstation is used as a 3270 terminal, through the 3270 emulator
- A workstation application program uses one of the 3270 emulator application programming interfaces

This chapter describes the LANDP 3270 emulator high-level application programming interface. The low level application programming interface is described in Chapter 28, “LANDP 3270 emulator API” on page 593.

Two functions of the local resource manager can be used with the LANDP for DOS 3270 emulator.

*Table 47. Function Codes used in the LANDP 3270 emulator HLL API*

| Function code | Description | Env. | Page |
|---|---|---|---|
| EC | End connection | 0-- | 519 |
| GS | Get status | 0-- | 520 |

The first and second columns give the function code and the name of the function. The third column shows the operating environment of the function. “0--” means that it is available from LANDP for DOS servers. The last column refers to the page where you can find the function described.

## Characteristics of the high-level language API

The LANDP 3270 Emulator High-Level Language Application Programming Interface (HLLAPI) is based on the IBM Personal Computer 3270 Entry Emulator High-Level Language Application Programming Interface (EEHLLAPI).

LANDP HLLAPI is largely compatible with other 3270 emulator high level language application programming interfaces. Compatibility and portability issues are discussed in “Compatibility with other IBM 3270 emulator application programming interfaces” on page 653.

An application program intended to use the LANDP 3270 emulator HLLAPI can perform tasks similar to those supported for the 3270 terminal operator. The application program can read and interpret the screen contents by using copy and compare commands. It can also write to the presentation space and send keystrokes in a similar way to the keyboard. The presentation space is an area in storage that corresponds to the image shown on the display screen when the 3270 emulator is used.

© Copyright IBM Corp. 1992, 2002

The following programming languages are supported:
- COBOL/2
- Pascal/2
- C/2
- BASIC Compiler/2
- Macro Assembler/2

## LANDP HLLAPI control flow

LANDP HLLAPI interacts with the software components listed below. These components must be installed before using LANDP HLLAPI.
- Language interface module (LIM)
- FBHLLAPI resident module
- 3270 emulator

The following figure shows the component interaction of LANDP HLLAPI and also the control flow of a LANDP HLLAPI function call.

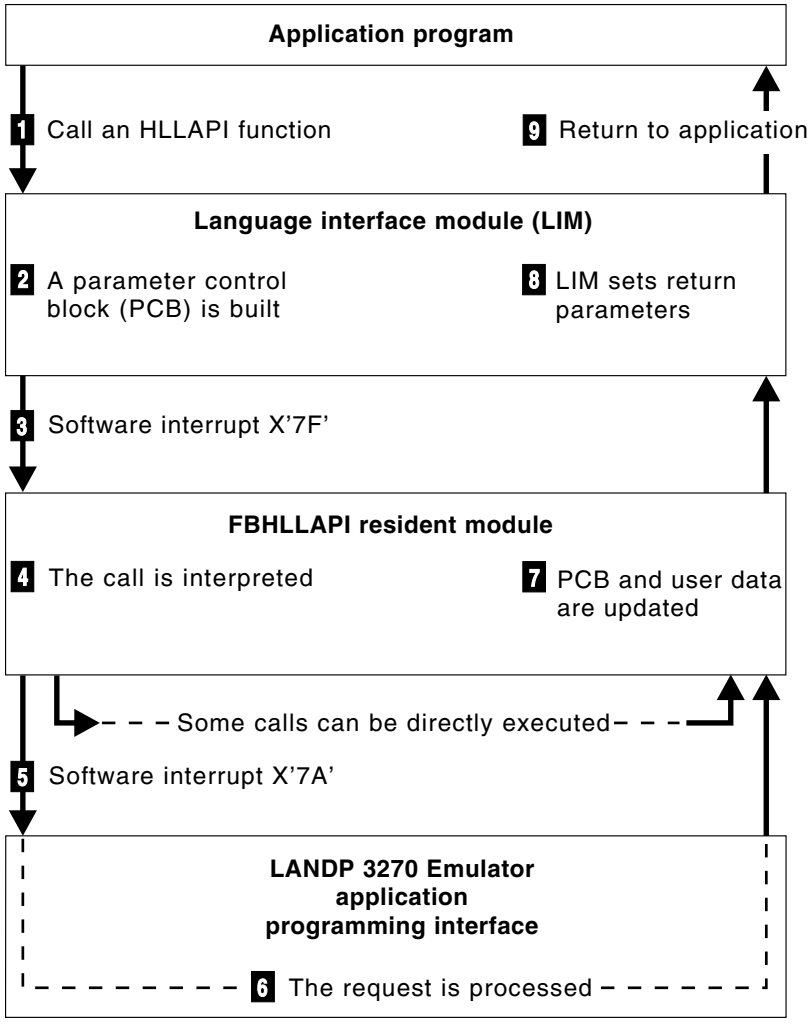



*Figure 6. Component Interaction of LANDP HLLAPI*

The sequence is:

1. The workstation application program sets the parameters and calls the appropriate LIM, which is link-edited with the application program.
2. The LIM takes the parameters and builds a parameter control block (PCB).
3. The LIM passes the parameter control block and the application program data to the FBHLLAPI resident module, using the software interrupt X'7F'.
4. The FBHLLAPI resident module interprets the received information.
5. The 3270 emulator application programming interface is called using the

software interrupt X'7A'. The 3270 emulator application programming interface is described in Chapter 28, “LANDP 3270 emulator API” on page 593.

6 The 3270 emulator processes the request.

7 The FBHLLAPI resident module analyzes the result and sets the responses in the PCB and in the application program variables.

8 Control is returned to the calling LIM that, in turn, sends the return information to the application program.

9 The application program receives control.

For calls to some complex functions, steps 5 and 6 are repeated without transferring control back to the application program. Some function calls can be processed directly by the FBHLLAPI resident module, going directly from step 4 to step 7.

## The language interface module

The primary function of the language interface module (LIM) is to provide language binding. To support different languages, LANDP HLLAPI is designed to use the LIM as a link between an application program and the FBHLLAPI resident module. The application program calls the appropriate LIM that passes the application program parameters to the FBHLLAPI resident module in a standard format.

Language interface modules are provided for the supported languages. The appropriate LIM is called as an external subroutine. Application programs written in Macro Assembler can invoke the functions directly. When an application program is linked, the appropriate LIM is included in the object module.

Detailed information about the LIM structure and functions can be found in “Writing your own language interface module” on page 654.

## The FBHLLAPI resident module

This module is a resident extension to IBM DOS. It must be loaded before using the last LANDP for DOS component. The syntax for loading FBHLLAPI, at the IBM DOS prompt, is:

```
FBHLLAPI [/u] [/t] [Sn]
```

The three parameters are optional. Their meaning is:

**/u** Unload. FBHLLAPI unloads itself from the interrupt chain and releases occupied storage.

**/t** Start trace at beginning of process. It is equivalent to specifying the TRON parameter by using the Set Session Parameters function. For information on this function, see “Set session parameters (function 9)” on page 622.

**Sn** Specifies the amount of stack FBHLLAPI allocates. `n` stands for the amount of stack in KB. The default is 1 KB. If you specify 0, FBHLLAPI uses the application stack. You may need to use this parameter when loading the LANDP 3270 emulators in expanded memory.

You can include FBHLLAPI invocation in the AUTOFBSS.BAT file.

## LANDP HLLAPI function calls

All function calls issued to LANDP HLLAPI use a fixed format with four parameters. Not all the function calls use every parameter to pass or receive data. However, all parameters must be set in the function call to meet application program linkage requirements.

### Calling parameters

The four parameters that must be set in any function call issued to LANDP HLLAPI are described in this section.

**Function Number**

Is always required and contains a 2-byte integer representing the function number.

**Data String**

Is used in different ways depending on the function. In most high-level languages, it is a structure. In others, it is a string of concatenated data items.

**String Length**

Is the length of the data string or concatenated list of data items.

When in *end of text* (EOT) mode, the length of the strings is not explicitly stated. Function calls that pass data to LANDP HLLAPI can use an EOT character, as a run time option, to terminate the data string. Here the length of an input string is not used.

**PS Position**

The presentation space position is the value associated with the screen size of the emulated session. When used, this parameter contains a 2-byte integer between 1 and the screen size value.

### Return values

The information is returned in the variables passed as parameters.

**Function Number**

Is returned unchanged.

**Data String**

Returns different information depending on the function. In most functions, it is a string of characters. In others, it is a string of concatenated data items.

The application program must allocate space enough for returned data strings.

**Result Code**

Usually contains a presentation space position value. In some functions, it contains other return information for the application program, or no information at all.

**Return Code**

Is a numeric return code.

**Note:** The return code must always be checked by the application program. If the return code is 9 (system error):
1. Call the Query System function
2. Extract the 8-byte extended system error data (at position 20) and print it. (This information may be useful for service personnel.)

### Function call format

The format of the function call in the application program depends on the programming language used. Some examples corresponding to different programming languages follow.

| Language | Statement |
|---|---|
| BASIC | CALL BLIM (FUNC%, SDATA$, DLEN%, RETC%) |
| COBOL | CALL 'HLLCOB' USING FUNC DATASTR DLEN RETC |
| Pascal | HLLPAS (Function_Num, Data_Str, Len, Ret_Cod) |
| C | HLLC (&API_FUNC, API_STR, &API_LEN, &API_RETC); |

## High-level language application programming interface services

To present the functions of these services in a logical way, they are grouped into six service categories:
- Operator Services
- Presentation Services
- Copy Services
- Device Services
- Communications Services
- Utility Services

Some services provided in the different groups are related to the presentation space (PS). The presentation space is an area in storage that corresponds to the image shown on the display screen when the 3270 emulator is used.

The presentation space contains a number of rows and columns. A PS position is a value between 1 (the home position) and n (the lower-right corner).

When the application program is connected to a presentation space, it can interact with both a *field formatted* presentation space and a *field unformatted* presentation space. A field formatted presentation space is that presentation space made up by one or more fields. A field is a group of consecutive positions in the presentation space, with similar characteristics, defined by a field attribute byte at the beginning of the field.

## Operator services

The function set provided in this group supports the interface between the application program and the 3270 emulator session. The functions are listed below and described in the following §s:
- Query System (Function 20)

- Query Sessions (Function 10)
- Query Session Status (Function 22)
- Set Session Parameters (Function 9)
- Send Key (Function 3)
- Wait (Function 4)
- Pause (Function 18)
- Start Host Notification (Function 23)
- Query Host Update (Function 24)
- Stop Host Notification (Function 25)
- Reset System (Function 21)

A typical sequence of function usage is:

1. *Query system*: used to obtain necessary information about the 3270 emulator session.
2. *Connect to presentation space* and *set session* parameters: used to define parameters.
3. *Start host notification*: used to allow the application program to use the Query Host Update function for the occurrence of the updates of the presentation space or the operator information area.
4. *Send key*: used to send, for example, LOGON SMITH to the session presentation space.
5. *Wait or pause*: used to monitor the host computer session. Then, the application program issues a *query host update* function call to determine whether the presentation space or the operator information area has been updated.
6. Search Presentation Space: used to find the location to enter or copy information (for example, ENTER PASSWORD) or for determining the status of a host computer response (for example, VM READ or RUNNING).
7. The application program can call functions from the other service groups.
8. After finishing, the application program must issue a disconnect from Presentation Space function call.
9. Reinitializing the application program may require the use of the Reset System function. This action resets all session parameters to default values.

## Query system (function 20)

This function obtains system-related information. It returns a string with the appropriate data. Most of this information is for use by service personnel, when system errors are met.

| **Calling Parameters:** | |
|---|---|
| **Function Number** | 20 |
| **Data String** | String with an allocated length of at least 35 bytes |
| **String Length** | 35 |
| **PS Position** | Not used |

**Return Values:**

**Data String:** Data string of 35 bytes with the following content:

| Position | Type | Length | Value |
|---|---|---|---|
| 1 | Character | 1 | FBHLLAPI major version |
| 2 | Character | 1 | FBHLLAPI minor version |
| 3 | Character | 2 | FBHLLAPI maintenance level |
| 5 | | 5 | Reserved |
| 10 | Character | 1 | LIM version |
| 11 | Character | 2 | LIM level |
| 13 | | 1 | Reserved |
| 14 | Character | 1 | Control program type, F=LANDP |
| 15 | Character | 1 | Emulator major version |
| 16 | Character | 1 | Emulator minor version |
| 17 | Character | 2 | Emulator maintenance level |
| 19 | Character | 1 | Reserved |
| 20 | Character | 4 | System error component code |
| 24 | Character | 4 | System error fault symptom code |
| 28 | | 8 | Reserved |

**Result code** Not used

**Return codes**

| | |
|---|---|
| **0** | Function successfully completed |
| **2** | String length specified too small |
| **9** | System error |

## Query sessions (function 10)

This function returns the number of defined 3270 emulator sessions and data describing each of the associated presentation spaces. Up to five 3270 emulator sessions can be defined in the same workstation.

| **Calling Parameters:** | |
|---|---|
| **Function Number** | 10 |
| **Data String** | String of 12 to 60 bytes |
| **String Length** | Length from 12 to 60 bytes. It must correspond to the size of the allocated string. |
| **PS Position** | Not used |

**Return Values:**

**Data String:** Array of concatenated session descriptors, each being 12 bytes long, containing information about one 3270 emulator session. The format is:

| Position | Type | Length | Value |
|---|---|---|---|
| 1 | Character | 1 | Session short name |
| 2 | Character | 8 | Session long name |
| 10 | | 1 | Reserved |
| 11 | Integer | 2 | Presentation space size |

**Result code** Number of defined 3270 emulator sessions. It is returned even though the *return code* might be 2.

**Return codes**

**0** Function successfully completed

**2** String length too small to store the data for the defined sessions

**9** System error

## Query session status (function 22)

This function returns session specific data.

**Calling Parameters:**

| | |
|---|---|
| **Function Number** | 22 |
| **Data String** | Short name of the 3270 emulator session. The allocated length for the string must be at least 18 bytes. |
| **String Length** | 18 bytes |
| **PS Position** | Not used |

**Return Values:**

**Data String:** A string of 18 bytes is returned:

| Position | Type | Length | Value |
|---|---|---|---|
| 1 | Character | 1 | Session short name |
| 2 | Character | 8 | Session long name |
| 10 | | 2 | Reserved |
| 12 | Integer | 2 | Number of rows |
| 14 | Integer | 2 | Number of columns |
| 16 | | 3 | Reserved |

**Result code** Not used

| Return codes | |
|---|---|
| **0** | Function successfully completed |
| **1** | Short name specified not valid |
| **2** | String length specified too small |
| **9** | System error |

## Set session parameters (function 9)

This function allows the application program to change certain default session options set in the FBHLLAPI resident module.

| Calling Parameters: | |
|---|---|
| **Function Number** | 9 |
| **Data String** | String containing the session parameters. It can contain any of the parameters listed below. Parameters must be separated by commas or blanks |
| **String Length** | Length of the data string. EOT mode cannot be used. |
| **PS Position** | Not used |

| Return Values: | | |
|---|---|---|
| **Data String** | Not used | |
| **Result Code** | Not used | |
| **Return Codes:** | | |
| | **0** | Function successfully completed |
| | **2** | Length of data string not valid |
| | **9** | System error |

The following table shows the *session parameters* that can be specified, using the Set Session Parameters function. The symbol (*) indicates default setting.

| Parameters affecting the Connect to Presentation Space function (see “Connect to presentation space (function 1)” on page 630). | |
|---|---|
| **CONPHYS** | Does a **physical** connect |
| **CONLOG** (*) | Does a **logical** connect |

| | |
|---|---|
| Parameters related to the copy service functions (see “Copy services” on page 639). | |
| **ATTRB** | Passes all attribute codes, even if they do not have an ASCII equivalent |
| **NOATTRB** (*) | Converts all attribute values to blanks |
| **STRLEN** (*) | An explicit length is used for all strings |
| **STREOT** | String lengths are not explicitly stated, strings are terminated with an EOT character |
| **EOT=n** | Specifies the EOT character for string end, when in STREOT mode. Binary zero is the default. If you insert a blank after the equal sign, blank is the EOT character |
| Parameters affecting sending keystrokes. | |
| **ESC=n** | Specifies the escape character for keystroke mnemonics. The default is @. If you insert a blank after the equal sign, blank is the escape character. |
| **AUTORESET** (*) | LANDP HLLAPI attempts to reset all inhibited conditions by prefixing all Send Key functions with a RESET key code, see “Send key (function 3)” on page 624. |
| **NORESET** | No AUTORESET |
| Parameters related to the Search Presentation Space function (see “Search presentation space (function 6)” on page 632) and Search Field function (see “Search field (function 30)” on page 635). | |
| **SRCHALL** (*) | The entire presentation space is searched |
| **SRCHFROM** | Start searches at a specified beginning position |
| **SRCHFRWD** (*) | Searches in the ascending (forward) direction |
| **SRCHBKWD** | Searches in an descending (backward) direction |
| Parameters related to the trace tools. The trace tools display the function and result codes for each LANDP HLLAPI function. For information on the trace tools provided with the LANDP programs, see the *LANDP Problem Determination* book. | |
| **TRON** | Turns trace on |
| **TROFF** (*) | Turns trace off |

| Parameters affecting the Wait function (see “Wait (function 4)” on page 625) and Pause function (see “Pause (function 18)” on page 626). | |
|---|---|
| **TWAIT** (*) | Timeout wait |
| **LWAIT** | Long wait |
| **NWAIT** | The Wait function checks status and returns immediately (no wait) |
| **FPAUSE** (*) | Full duration pause. It pauses as specified |
| **IPAUSE** | Interruptible pause. A host computer event satisfies the Pause function call |
| Parameters related to the Send File function (see “Send file (function 90)” on page 649) and Receive File function (see “Receive file (function 91)” on page 651). | |
| **QUIET** | SEND and RECEIVE messages are not displayed |
| **NOQUIET** (*) | Messages are displayed |

## Send key (function 3)

This function sends a series of keystrokes to the presentation space. Before sending keystrokes, the application program must issue a Connect to Presentation Space function (see “Connect to presentation space (function 1)” on page 630). The string of keystrokes is handled as if entered by the workstation operator. All fields that are protected for input or numeric only must be treated accordingly.

| **Calling Parameters:** | |
|---|---|
| **Function Number** | 3 |
| **Data String** | String of keystrokes. The maximum number is 255 |
| **String Length** | Length of the data string. If EOT mode is used, the parameter value is ignored |
| **PS Position** | Not used |

| **Return Values:** | | |
|---|---|---|
| **Data String** | Not used | |
| **Result Code** | Not used | |
| **Return Codes:** | | |
| | **0** | Function successfully completed |
| | **1** | Not connected to any session |
| | **2** | Parameter not valid |
| | **4** | 3270 session busy, all keystrokes could not be sent |
| | **5** | 3270 session locked, all keystrokes could not be sent |
| | **9** | System error |

**Notes:**

1. Keystroke input is not accepted after the first AID character.
2. Keystrokes cannot be sent to the host computer session when the keyboard is locked or busy. This can be checked with the Wait function (see "Wait (function 4)"). You can use the Set Session Parameters function (see "Set session parameters (function 9)" on page 622) to specify AUTORESET, which causes the keystrokes to be preceded by a RESET key. This action clears the locked condition but not the busy condition.
3. Multiple Send Key functions can be issued before sending an ENTER.
4. The Copy String to Field function (see "Copy string to field (function 33)" on page 639) and the Copy String to Presentation Space function (see "Copy string to presentation space (function 15)" on page 645) can be used instead of the Send Key function, but special control characters can only be sent with the Send Key function.

To send 3270 emulator control keys, a compound character coding scheme is used, which uses two ASCII characters to show one keystroke. It consists of @ followed by the key code. The coding scheme purpose is to allow for ASCII string representation of all necessary keystroke codes. When calculating data string lengths, the compound keystrokes count as being two bytes long. When issuing a Set Session Parameters function (see "Set session parameters (function 9)" on page 622), you can set an escape character other than @. In the following list, the upper and lower case alphabetic characters are mnemonic abbreviations for different keys. The symbol (*) indicates AID keys.

| | | | |
|---|---|---|---|
| **@@** | @ | **@V** | Down Cursor |
| **@B** | Back-Tab | **@Z** | Right Cursor |
| **@C** | Clear (*) | **@0** | Home |
| **@D** | Delete Character | **@1 to @9** | PF1 to PF9 (*) |
| **@E** | Enter (*) | **@a to @o** | PF10 to PF24 (*) |
| **@F** | Erase EOF | **@x to @z** | PA1 to PA3 (*) |
| **@I** | Insert | **@A@F** | Erase Input |
| **@L** | Left Cursor | **@A@H** | System Request (*) |
| **@N** | New Line | **@A@J** | Cursor Select (*) |
| **@R** | Reset | **@A@Q** | Attention (*) |
| **@T** | Tab | **@S@x** | Dup |
| **@U** | Up Cursor | **@S@y** | Field Mark |

## Wait (function 4)

This function monitors the status of the 3270 emulator session. If the 3270 emulator session is set on *terminal wait* or on *system wait*, waiting for a host computer response, the Wait function waits roughly one minute checking if the condition clears.

The *terminal wait* or *system wait* status is shown as "X CLOCK" and "X SYSTEM" in the operator information area (OIA). The OIA is the last line in the 3270 emulator screen, where the communication status with the host computer is displayed.

| Calling Parameters: | |
|---|---|
| **Function Number** | 4 |
| **Data String** | Not used |
| **String Length** | Not used |
| **PS Position** | Not used |

| Return Values: | | |
|---|---|---|
| **Data String** | Not used | |
| **Result Code** | Not used | |
| **Return Codes:** | | |
| | **0** | Keyboard ready for input |
| | **1** | Application program not connected to a valid session |
| | **4** | Timeout while still in *terminal wait* or *system wait* |
| | **5** | Keyboard locked |
| | **9** | System error |

The Wait function is used when issuing requests to the host computer that require time to complete—for example, when issuing a Send Key function call (see “Send key (function 3)” on page 624). The parameters influencing the performance of the Wait function are defined using the Set Session Parameters function. Those parameters are:

**TWAIT** Timeout after roughly one minute and return to the application program. The return code is 4 if either *terminal wait* condition or *system wait* condition remains.

**NWAIT** Checks for the status and returns to the application program immediately. The return code reflects the condition.

**LWAIT** No return until the condition is cleared. This value must be used with caution because control does not return to the application program until the host computer is available.

The Wait function is satisfied by the host computer by unlocking the keyboard. It does not mean that the transaction has been completed. The application program should use one of the search functions to look for expected keywords. If necessary, repeat the Wait function and search again.

## Pause (function 18)

This function pauses for a specified duration and should be used to wait for an event to occur. The FPAUSE (full duration pause) mode is the default. If the IPAUSE (interruptible pause) mode is selected with the Set Session Parameters function (see

"Set session parameters (function 9)" on page 622), and Start Host Notification function (see "Start host notification (function 23)") is called, the Pause function call is satisfied by a host computer event.

The Pause function should not be used for very long delays or for application programs requiring a precise timer. The interval provided by the Pause function is approximate.

| **Calling Parameters:** | |
|---|---|
| **Function Number** | 18 |
| **Data String** | Not used |
| **String Length** | Pause duration in half-second increments |
| **PS Position** | Not used |

| **Return Values:** | | |
|---|---|---|
| **Data String** | Not used | |
| **Result Code** | Not used | |
| **Return Codes:** | | |
| | **0** | Wait duration expired. |
| | **9** | System error. |
| | **26** | Host computer session presentation space or OIA updated. |

## Start host notification (function 23)

This function initiates the process to determine whether the presentation space or the operator information area has been updated. The Query Host Update function (see "Query host update (function 24)" on page 628) is used to determine which host computer event has occurred.

| Calling Parameters: | | | | |
|---|---|---|---|---|
| **Function Number** | 23 | | | |
| **Data String** | Information characters: | | | |
| | **Position** | **Type** | **Length** | **Value** |
| | 1 | Character | 1 | One of the following:<br>A specific presentation space short name<br>A blank, indicating a request to the currently connected presentation space |
| | 2 | Character | 1 | The character “B” asking for notification of both Presentation Space and OIA updates |
| | 3 | | 4 | Not used by LANDP HLLAPI (maintained for compatibility with other HLLAPIs) |
| **String Length** | Length of the host computer event buffer. This buffer is not used by LANDP HLLAPI, but the parameter is accepted for compatibility with other 3270 high-level programming interfaces. The length recommended is 256 | | | |
| **PS Position** | Not used | | | |

## Query host update (function 24)

This function allows the application program to determine whether the presentation space or the operator information area have been updated since the last function call. The Start Host Notification function (see “Start host notification (function 23)” on page 627) must be called before using the Query Host Update function.

| Calling Parameters: | |
|---|---|
| **Function Number** | 24 |
| **Data String** | One character short name of the 3270 emulator session, or a blank meaning a request to the currently connected presentation space |
| **String Length** | 1 |
| **PS Position** | Not used |

| Return Values: | | |
|---|---|---|
| **Data String** | Not used | |
| **Result Code** | Not used | |
| **Return Codes:** | | |
| | **0** | No updates made since last call |
| | **1** | Session short name specified not valid |
| | **8** | No prior Start Host Notification function called for specified presentation space |
| | **9** | System error |
| | **21** | Operator information area updated |
| | **22** | Presentation space updated |
| | **23** | Both presentation space and operator information area updated |

## Stop host notification (function 25)

This function disables the Query Host Update function (see “Query host update (function 24)” on page 628). It also prevents host computer events from affecting the Pause function (see “Pause (function 18)” on page 626).

| Calling Parameters: | |
|---|---|
| **Function Number** | 25 |
| **Data String** | One character short name of the 3270 emulator session, or a blank meaning a request to the currently connected presentation space |
| **String Length** | 1 |
| **PS Position** | Not used |

| Return Values: | | |
|---|---|---|
| **Data String** | | |
| Not used | | |
| **Result Code** | | |
| Not used | | |
| **Return Codes:** | | |
| | **0** | Function successfully completed |
| | **1** | Session short name specified not valid |
| | **8** | No prior Start Host Notification function called for this presentation space |
| | **9** | System error |

## Reset system (function 21)

This function reinitializes the FBHLLAPI resident module. It also allows the application program to set the system to a known initial condition. The reset actions are:

- Session parameters are reset to default values
- The presentation space is disconnected (if connected)
- The presentation space is released (if reserved)
- Host computer notification is stopped (if started)

| **Calling Parameters:** | |
|---|---|
| **Function Number** | 21 |
| **Data String** | Not used |
| **String Length** | Not used |
| **PS Position** | Not used |

| **Return Values:** | | |
|---|---|---|
| **Data String** | Not used | |
| **Result Code** | Not used | |
| **Return Codes:** | | |
| | **0** | Function successfully completed |
| | **9** | System error |

## Presentation services

The function set provided in this group supports access to the presentation space. The functions are listed below and described in the following §s:

- Connect to Presentation Space (Function 1)
- Disconnect from Presentation Space (Function 2)
- Search Presentation Space (Function 6)
- Query Cursor Location (Function 7)
- Query Field Attribute (Function 14)
- Search Field (Function 30)
- Find Field Position (Function 31)
- Find Field Length (Function 32)
- Set Cursor (Function 40)

## Connect to presentation space (function 1)

This function establishes connection between the application program and the presentation space. The application program must call the Connect to Presentation Space function before interacting with the presentation space.

| Calling Parameters: | |
|---|---|
| **Function Number** | 1 |
| **Data String** | One character short name of the presentation space |
| **String Length** | 1 |
| **PS Position** | Not used |

| Return Values: | | |
|---|---|---|
| **Data String** | Not used | |
| **Result Code** | Not used | |
| **Return Codes:** | | |
| | **0** | Function successfully completed. Presentation space unlocked and ready for input. |
| | **1** | Presentation space short name not valid. |
| | **4** | Connect successful, but presentation space busy. *Terminal wait* or *system wait* condition. |
| | **5** | Connect successful, but presentation space locked. *Input inhibited* condition. |
| | **8** | Application already connected to a presentation space. |
| | **9** | System error. |
| | **11** | Resource not available. Presentation space used by another system function. |

The two following parameters, which can be defined through the Set Session Parameters function, affect the Connect to Presentation Space function.

**Logical connection (CONLOG)**
: During connection, the 3270 emulator presentation space is not displayed. The application program controls what appears on the screen.

**Physical connection (CONPHYS)**
: During connection, the 3270 emulator presentation space is displayed. It is equivalent to pressing the 3270 emulator activation hot-key.

  If the application program writes to the screen, overwriting occurs. If the application program accesses the keyboard, it conflicts with the 3270 emulator keyboard access.

Before calling the following functions, it is not necessary to issue a call to the Connect to Presentation Space function.

- Set Session Parameters (Function 9)
- Query Sessions (Function 10)
- Pause (Function 18)

- Query System (Function 20)
- Reset System (Function 21)
- Query System Status (Function 22)
- Start Host Notification (Function 23)
- Query Host Update (Function 24)
- Stop Host Notification (Function 25)
- Send File (Function 90)
- Receive File (Function 91)
- Convert Position or RowCol (Function 99)

## Disconnect from presentation space (function 2)

This function disconnects the personal computer system session from the presentation space. The Disconnect from Presentation Space function must be called at the end of an application program. Otherwise, errors can occur to other application programs.

| **Calling Parameters:** | |
|---|---|
| **Function Number** | 2 |
| **Data String** | Not used |
| **String Length** | Not used |
| **PS Position** | Not used |

| **Return Values:** | | |
|---|---|---|
| **Data String** | Not used | |
| **Result Code** | Not used | |
| **Return Codes:** | | |
| | **0** | Function successfully completed |
| | **1** | Application program not connected to any presentation space |
| | **9** | System error |

When the Disconnect from Presentation Space function is called, the functions that interact with an active presentation space cannot be used. Examples of those functions are: the Send Key function (see “Send key (function 3)” on page 624), the Wait function (see “Wait (function 4)” on page 625), the Reserve function (see “Reserve (function 11)” on page 646) and the Release function (see “Release (function 12)” on page 647).

## Search presentation space (function 6)

This function allows the application program to search the connected presentation space for the occurrence of a specified string. A search is satisfied if the first character of the requested string starts within the bounds specified for a search function.

| Calling Parameters: | |
|---|---|
| **Function Number** | 6 |
| **Data String** | Target string of search |
| **String Length** | Length of string, if not in EOT mode |
| **PS Position** | Position within presentation space, not if in SRCHALL mode |

| Return Values: | | |
|---|---|---|
| **Data String** | Not used | |
| **Result Code** | Position where the string is found. A zero means it has not been found | |
| **Return Codes:** | | |
| | **0** | Function successfully completed |
| | **1** | Application program not connected to presentation space |
| | **2** | Parameter not valid. For example, no EOT found in EOT mode |
| | **7** | Presentation space not valid |
| | **9** | System error |
| | **24** | Search string not found |

The Search Presentation Space function scans the presentation space for the occurrence of a specified string. If the string is not found, the result code is set to zero. If the string is found, the result code is set to the location in the presentation space where the string begins.

The following parameters, which can be defined through the Set Session Parameters function (see “Set session parameters (function 9)” on page 622), affect the Search Presentation Space function.

- In SRCHFRWD mode the search begins:
  - At the home position (position 1), in SRCHALL mode
  - At the specified position, in SRCHFROM mode.

  In this mode, the first occurrence of the string is located. The search always stops at the end of the presentation space.
- In SRCHBKWD mode, the search begins at the end of the presentation space and proceeds backwards to:
  - Position 1, in SRCHALL mode
  - Position specified, in SRCHFROM mode

  In this mode, the last occurrence of the string is located.

The Search Presentation Space function is useful when a specific message must arrive before continuing. The application program can first use the Pause function (see “Pause

(function 18)" on page 626) or the Query Host Update function (see "Query host update (function 24)" on page 628) to verify that a host computer response has arrived.

## Query cursor location (function 7)

This function returns the cursor position in the presentation space. Before using the Query Cursor Location function, the application program must call the Connect to Presentation Space function (see "Connect to presentation space (function 1)" on page 630).

| Calling Parameters: | |
|---|---|
| **Function Number** | 7 |
| **Data String** | Not used |
| **String Length** | Not used |
| **PS Position** | Not used |

| Return Values: | | |
|---|---|---|
| **Data String** | Not used | |
| **Result Code** | Cursor position in presentation space | |
| **Return Codes:** | | |
| | **0** | Function successfully completed |
| | **1** | Application program not connected to presentation space |
| | **9** | System error |

## Query field attribute (function 14)

This function returns the attribute byte of the field containing the presentation space position.

| Calling Parameters: | |
|---|---|
| **Function Number** | 14 |
| **Data String** | Not used |
| **String Length** | Not used |
| **PS Position** | Presentation space position from where to get field attribute |

| Return Values: | |
|---|---|
| **Data String** | Not used |
| **Result Code** | Attribute value, or 0 if unformatted screen. Attribute bytes are values greater that X'C0' |
| **Return Codes:** | |
| | **0** Function successfully completed |
| | **1** Application program not connected to presentation space |
| | **7** Presentation space position not correct |
| | **9** System error |
| | **24** Attribute not found or presentation space unformatted |

## Search field (function 30)

This function examines one field within the presentation space for the occurrence of a string. If found, the function returns the position of the string. If not found, the function returns zero.

The Search Field function is used to search both protected and unprotected fields, but only in a field formatted presentation space.

| Calling Parameters: | |
|---|---|
| **Function Number** | 30 |
| **Data String** | Target string for search |
| **String Length** | Length of target string, if not in EOT mode |
| **PS Position** | Identifies the target field |

| Return Values: | |
|---|---|
| **Data String** | Not used |
| **Result Code** | Decimal presentation space position in presentation space. Zero means that the string is not found |
| **Return Codes:** | |
| | **0** Function successfully completed |
| | **1** Application program not connected to presentation space |
| | **7** Presentation space position not valid |
| | **9** System error |
| | **24** Search string not found, or presentation space unformatted |

The following parameters, which can be defined through the Set Session Parameters function (see “Set session parameters (function 9)” on page 622), affect the Search Field function.

- In SRCHALL mode the specified presentation space position specifies the target field, and the entire field is searched.
- In SRCHFROM mode the specified presentation space position specifies not only the target field but also a beginning or ending point to begin or end the search:
  - The search begins at the specified presentation space position and proceeds to the end, in SRCHALL mode
  - The search begins at the end of the field and proceeds to the beginning, in SRCHFROM mode

## Find field position (function 31)

This function returns the beginning position of a target field in the presentation space. The field position returned is the offset of the first data byte following the field attribute byte.

The Find Field Position function can be used either for the specified target field, or for the next field, or for a previous field. The fields can be both protected and unprotected fields in a formatted presentation space.

| **Calling Parameters:** | |
|---|---|
| **Function Number** | 31 |
| **Data String** | 2-byte data string containing one of the following items:<br>**T** Target field. Two blanks can also be used<br>**P** Previous field, either protected or unprotected<br>**N** Next field, either protected or unprotected<br>**NP** Next protected field<br>**PP** Previous protected field<br>**NU** Next unprotected field<br>**PU** Previous unprotected field |
| **String Length** | 2 |
| **PS Position** | Identification for the field within the presentation space where the search begins. It can be any position within the field. |

| Return Values: | |
|---|---|
| **Data String** | Not used |
| **Result Code** | Relative position of requested field from the origin of the presentation space. This position is defined to be the first position after the attribute byte. Zero means:<br>• Presentation space unformatted, if return code not equal to 28<br>• Field length zero, if return code = 28 |
| **Return Codes:** | |
| **0** | Function successfully completed |
| **1** | Application program not connected to presentation space |
| **2** | Parameter error |
| **7** | Presentation space position not valid |
| **9** | System error |
| **24** | No such field found |
| **28** | Field length zero |

## Find field length (function 32)

This function returns the length of the target field in the presentation space. The number of characters contained in the defined field is returned. This includes all characters from the beginning of the target field up to either the next attribute byte or the end of the presentation space.

The Find Field Length function can be used either for the specified target field, or for the next field, or for a previous field. The fields can be both protected and unprotected fields in a formatted presentation space, but only in a formatted presentation space.

| Calling Parameters: | |
|---|---|
| **Function Number** | 32 |
| **Data String** | 2-byte data string. Its content is the same as for the Find Field Position function (see “Find field position (function 31)” on page 636). |
| **String Length** | 2 |
| **PS Position** | Identification for the field within the presentation space where the search begins. It can be any position within the field |

| Return Values: | |
|---|---|
| **Data String** | Not used |
| **Result Code** | Length of requested field in presentation space. Zero means:<br>• Presentation space unformatted, if return code not equal to 28<br>• Field length zero, if return code = 28 |
| **Return Codes:** | |
| **0** | Function successfully completed |
| **1** | Application program not connected to the presentation space |
| **2** | Parameter not valid |
| **7** | Presentation space position not valid |
| **9** | System error |
| **24** | No such field found |
| **28** | Field length zero |

## Set cursor (function 40)

This function sets the position of the cursor in the presentation space. To use the Set Cursor function, the application program must be connected to the presentation space.

| Calling Parameters: | |
|---|---|
| **Function Number** | 40 |
| **Data String** | Not used |
| **String Length** | Not used |
| **PS Position** | Target cursor position |

| Return Values: | |
|---|---|
| **Data String** | Not used |
| **Result Code** | Not used |
| **Return Codes:** | |
| **0** | Function successfully completed |
| **1** | Application program not connected to the presentation space |
| **4** | Session busy |
| **5** | Session locked |
| **7** | Specified cursor location not valid (out of screen) |
| **9** | System error |

## Copy services

The function set provided in this group supports data exchange between the application program and the 3270 emulator. For data exchange purposes the application program treats the data as ASCII strings.

The functions are listed below and described in the following sections:

- Copy Field to String (Function 34)
- Copy String to Field (Function 33)
- Copy OIA (Function 13)
- Copy Presentation Space (Function 5)
- Copy Presentation Space to String (Function 8)
- Copy String to Presentation Space (Function 15)

## Copy string to field (function 33)

This function transfers a string of characters from the data string to a specified field in the presentation space. It can only be used in a field formatted presentation space. Copy is always to the first position in the field.

| **Calling Parameters:** | |
|---|---|
| **Function Number** | 33 |
| **Data String** | String containing data to be transferred to a target field in the presentation space |
| **String Length** | Length of source data string, if not in EOT mode |
| **PS Position** | Position of any byte in the target field |

| **Return Values:** | | |
|---|---|---|
| **Data String** | Not used | |
| **Result Code** | Not used | |
| **Return Codes:** | | |
| | **0** | Function successfully completed |
| | **1** | Application program not connected to presentation space |
| | **2** | Parameter not valid |
| | **5** | Target field protected or inhibited, or data not valid sent to the target field |
| | **6** | Copy completed, but data truncated. Probably the target field is shorter than the source string |
| | **7** | Presentation space position not valid |
| | **9** | System error |
| | **24** | Unformatted presentation space |

The copy of the string ends when any of the following conditions occurs:
- An EOT character is found in the string, when in EOT mode
- The number specified in the length is reached, when not in EOT mode
- The end of target field is found

## Copy field to string (function 34)

This function transfers characters from the presentation space to a string. It can be used with either protected or unprotected fields, but the presentation space must be field formatted.

The copy of the string ends when any of the following conditions occurs:
- The end of the field is reached
- The length of the target string is reached.

The position and length of the source field can be determined with the Find Field Position function (see “Find field position (function 31)” on page 636) and the Find Field Length function (see “Find field length (function 32)” on page 637).

| **Calling Parameters:** | |
|---|---|
| **Function Number** | 34 |
| **Data String** | Allocated target data string |
| **String Length** | Length of allocated string |
| **PS Position** | Position of the source field in the presentation space. It can be any position in the field, but copying always begins at the beginning of the field |

| **Return Values:** | | |
|---|---|---|
| **Data String** | Requested data string | |
| **Result Code** | Not used | |
| **Return Codes:** | | |
| | **0** | Function successfully completed |
| | **1** | Application program not connected to presentation space |
| | **2** | Parameter not valid |
| | **6** | Data to copy and target string are not of the same length. The data is truncated if the string length is smaller |
| | **7** | Incorrect presentation space position |
| | **9** | System error |
| | **24** | Unformatted presentation space |

## Copy OIA (function 13)

This function returns the current operator information area (OIA) data for the connected session. The OIA is located at the bottom of the screen and provides session status information.

| **Calling Parameters:** | |
|---|---|
| **Function Number** | 13 |
| **Data String** | Pre-allocated string of 103 bytes |
| **String Length** | 103 |
| **PS Position** | Not used |

| **Return Values:** | | |
|---|---|---|
| **Data String** | OIA data | |
| **Result Code** | Not used | |
| **Return Codes:** | | |
| | **0** | Function successfully completed. The presentation space is unlocked |
| | **1** | Application program not connected to presentation space |
| | **2** | String length not valid. OIA data is not returned |
| | **4** | Copy successful, but presentation space busy |
| | **5** | Copy successful, but presentation space locked |
| | **9** | System error |

The OIA data is returned in the *data string*. The corresponding format is presented in the next table.

| Position | Content |
|---|---|
| 1 | OIA format byte |

| Position | Content |
|---|---|
| 2 to 81 | OIA image. The image returned is in the format used by IBM 3270 PC in host computer code points. The most important codes are:<br><br>**2** System status:<br>**X'10'** No system available. Probably a configuration error or a LAN failure<br>**X'F4'** System available<br><br>**3** Communication status:<br>**X'10'** No communication available. The 3270 emulator is not connected to the host computer.<br>**X'CD'** Communication available. The 3270 emulator is connected to the host computer.<br><br>**4** Session status:<br>**X'10'** No session established<br>**X'F0'** System operator session. SSCP-LU dialog in progress<br>**X'CF'** Application program session. LU-LU dialog.<br><br>**10** Input inhibited status:<br>**X'10'** Keyboard ready, terminal ready, and system ready<br>**X'C6'** Keyboard locked, or terminal busy, or system busy |
| 82 to 103 | OIA group. This is a bit image of the operator information area. Only group 1 (1 byte) and group 8 (5 bytes) are valid. The remaining bytes and groups are reserved.<br>• Group 1 returns the bit image of the Online and Screen ownership information.<br>• Group 8 returns the bit image of the input inhibition reasons.<br><br>The states are ordered so that the higher the order bit, the higher the indicator priority.<br><br>The tables below show the meanings of the bits. |

The bit values corresponding to the group 1 byte of the operator information area are presented in the next table.

| OIA Group 1 Bit Image | | |
|---|---|---|
| **Byte** | **Bit** | **Value** |
| 82 | 7-6<br>5<br>4<br>3<br>2<br>1-0 | Reserved<br>SSCP-LU session owns screen<br>LU-LU session owns screen<br>Online and not owned<br>Subsystem ready<br>Reserved |

The bit values corresponding to the group 8 bytes of the operator information area are presented in the next table.

| OIA Group 8 Bit Image | | |
|---|---|---|
| **Byte** | **Bit** | **Value** |
| 89 | 7-6<br>5<br>4<br>3<br>2<br>1<br>0 | Reserved<br>Machine check<br>Communications check<br>Program check<br>Retry<br>Reserved<br>Reserved |
| 90 | 7<br>6<br>5<br>4<br>3<br>2-1<br>0 | Reserved<br>Terminal wait<br>Reserved<br>Minus function<br>Too much entered<br>Reserved<br>Numeric field |
| 91 | 7-4<br>3<br>2-0 | Reserved<br>Wrong place<br>Reserved |
| 92 | 7-6<br>5<br>4-0 | Reserved<br>System wait<br>Reserved |
| 93 | 7<br>6<br>5-0 | Reserved<br>Input disabled by application program<br>Reserved |

## Copy presentation space (function 5)

This function copies the entire contents of the presentation space into a data area allocated in the application program. The data area size must be large enough.

The Copy Presentation Space function translates the characters in the source presentation space to ASCII. Attribute bytes are translated to blanks. Using the Set Session Parameters function (see “Set session parameters (function 9)” on page 622), you can set the ATTRB parameter and transfer attribute bytes without translation.

| Calling Parameters: | |
|---|---|
| **Function Number** | 5 |
| **Data String** | Allocated target area of the size of the presentation space |
| **String Length** | Length of the presentation space |
| **PS Position** | Not used |

| Return Values: | | |
|---|---|---|
| **Data String** | Returned presentation space content | |
| **Result Code** | Not used | |
| **Return Codes:** | | |
| | **0** | Function successfully completed, presentation space ready and unlocked |
| | **1** | Application program not connected to presentation space |
| | **2** | Receiving string too small |
| | **4** | Copy successful, but presentation space busy |
| | **5** | Copy successful, but presentation space locked |
| | **9** | System error |

Using the Query Sessions function (see “Query sessions (function 10)” on page 620) or the Query Session Status function (see “Query session status (function 22)” on page 621), the application program can obtain the presentation space size.

## Copy presentation space to string (function 8)

This function transfers a portion of the presentation space, beginning at a specified position with a specified length, to an application program data area. The data area size must be large enough.

The value of the presentation space position plus the string length must not exceed the maximum size of the presentation space.

The Copy Presentation Space function translates the characters in the source presentation space to ASCII. Attribute bytes are translated to blanks. Using the Set Session Parameters function (see “Set session parameters (function 9)” on page 622), you can set the ATTRB parameter and transfer attribute bytes without translation.

| Calling Parameters: | |
|---|---|
| **Function Number** | 8 |
| **Data String** | Pre-allocated target string |
| **String Length** | Length of the target data string |
| **PS Position** | Byte position in the presentation space where the copy should begin |

| Return Values: | | |
|---|---|---|
| **Data String** | Content of the defined part of the presentation space | |
| **Result Code** | Not used | |
| **Return Codes:** | | |
| | **0** | Function successfully completed, the presentation space ready and unlocked |
| | **1** | Application program not connected to presentation space |
| | **2** | String length error |
| | **4** | Copy successful, but presentation space busy |
| | **5** | Copy successful, but presentation space locked |
| | **7** | Presentation space position not valid |
| | **9** | System error |

## Copy string to presentation space (function 15)

This function copies data from the application program to the presentation space. The ASCII data in the data string is copied directly into the connected presentation space at the location specified.

The value of the presentation space position plus the string length cannot exceed the maximum size of the presentation space.

| Calling Parameters: | |
|---|---|
| **Function Number** | 15 |
| **Data String** | String of ASCII data to be copied into the presentation space |
| **String Length** | Length of data string, if not in EOT mode |
| **PS Position** | Start position in the presentation space |

| Return Values: | | |
|---|---|---|
| **Data String** | Not used | |
| **Result Code** | Not used | |
| **Return Codes:** | | |
| | **0** | Function successfully completed, presentation space ready and unlocked |
| | **1** | Application program is not connected to the presentation space |
| | **2** | String length error |
| | **5** | Target presentation space protected or inhibited |
| | **6** | Copy completed, but data truncated |
| | **7** | Presentation space position not valid |
| | **9** | System error |

Using the Query Sessions function (see “Query sessions (function 10)” on page 620) or the Query Session Status function (see “Query session status (function 22)” on page 621), the application program can obtain the presentation space size.

Using the Query Cursor Location function (see “Query cursor location (function 7)” on page 634), you can set the presentation space location and place the data at the cursor location. On copy completion the presentation space cursor location is unchanged.

When any of the following conditions occur, the copy completes:
- An EOT character is found in the string, when in EOT mode
- The number specified in the length is reached, when not in EOT mode
- An attempt is made to copy data to a protected location

Using the Send Key function (see “Send key (function 3)” on page 624), you can also transfer data to the presentation space. However, the Copy String to Presentation Space function is recommended for performance reasons. You cannot send mnemonics using the Copy String to Presentation Space function. The Send Key function (see “Send key (function 3)” on page 624) should be used instead.

## Device services

The function set provided in this group prevents keyboard input, and allows for keyboard input. The functions are listed below and described in the following sections:
- Reserve (Function 11)
- Release (Function 12)

## Reserve (function 11)

This function locks the connected presentation space to prevent keyboard input.

The lock condition remains until either the Release function (see “Release (function 12)”) or the Reset System function (see “Reset system (function 21)” on page 630) are called.

| Calling Parameters: | |
|---|---|
| **Function Number** | 11 |
| **Data String** | Not used |
| **String Length** | Not used |
| **PS Position** | Not used |

| Return Values: | | |
|---|---|---|
| **Data String** | Not used | |
| **Result Code** | Not used | |
| **Return Codes:** | | |
| | **0** | Function successfully completed |
| | **1** | Application program not connected to presentation space |
| | **9** | System error |

## Release (function 12)

This function unlocks the presentation space to allow for keyboard input.

The application program must call the Release function before terminating. If the function call is not issued, the keyboard remains locked. To unlock the keyboard, the Reset System function (see “Reset system (function 21)” on page 630) is also available.

| Calling Parameters: | |
|---|---|
| **Function Number** | 12 |
| **Data String** | Not used |
| **String Length** | Not used |
| **PS Position** | Not used |

| Return Values: | | |
|---|---|---|
| **Data String** | Not used | |
| **Result Code** | Not used | |
| **Return Codes:** | | |
| | **0** | Function successfully completed |
| | **1** | Application program is connected to the presentation space |
| | **9** | System error |

## Communication services

The function set provided in this group transfers files between a workstation and a host computer. The functions are listed below and described in the following sections:

- Send File (Function 90)
- Receive File (Function 91)

The Send File function and the Receive File function are the same as those functions invoked through the SEND command and the RECEIVE command, at the IBM DOS prompt. LANDP HLLAPI calls either the Send/Receive programs provided by the File Transfer facility or, if these have been deleted or renamed, the send/Receive program provided by the 3270 Send/Receive facility. The File Transfer facility and 3270 Send/Receive facility are supplied with LANDP for DOS. For information on the File Transfer facility or the 3270 Send/Receive facility, see the *LANDP Servers and System Management* book.

The Send File function and the Receive File function provide additional abilities to application programs written in a high-level language. However, some effort may be required to use those functions with the application programs written in a high-level language for the LANDP HLLAPI. For more information, see the IBM *DOS Technical Reference and Application Programming* book.

- When IBM DOS loads a program, it normally assigns to it all memory from the program segment prefix to the high end of memory, including the memory occupied by COMMAND.COM that contains the program loader.

  Normally, the DOS SETBLOCK function is used to shrink its allocated memory block to the size actually needed. This action frees memory, which can then be used for loading following programs and the loader, if necessary. The Send File function and the Receive File function can only operate if there is enough free memory for the appropriate program to be loaded. If there is not enough space available, LANDP HLLAPI returns a 308 return code, meaning insufficient memory.

  Since there is no uniform way for LANDP HLLAPI to perform a SETBLOCK function for all languages and all situations, it does not do it. The application programmer should use the DOS SETBLOCK function (DOS interrupt X'21' AH=X'4A') in the programming language used so that the SEND and RECEIVE programs can run.

- LANDP HLLAPI uses the INT X'21' AH=X'4B' (EXEC) request that allows a parent application program to invoke a sub-process program. Since the DOS function requires the sub-process to inherit the environment segment of the parent, it is required that the application program has not freed or overwritten its environment.
- On return from the sub-process, IBM DOS re-invokes the COMMAND.COM program. Therefore COMMAND.COM must be available in the path specified in the environment variable COMSPEC.
- Since the application program cannot be connected to any presentation space, the error message *Unable to transmit command line* or *Unable to establish connection* is returned. Therefore:
  1. If the application program has used the Reserve function (see “Reserve (function 11)” on page 646), it must use the Release function (see “Release (function 12)” on page 647).
  2. If the application program has used the Connect to Presentation Space function (see “Connect to presentation space (function 1)” on page 630), it must then issue the Disconnect from Presentation Space function (see “Disconnect from presentation space (function 2)” on page 632).

  The two following parameters affect both the Send File function and the Receive File function. They are set using the Set Session Parameters function (see “Set session parameters (function 9)” on page 622).
  – QUIET
  – NOQUIET

## Send file (function 90)

This function sends a workstation file to a host computer.

| **Calling Parameters:** | |
|---|---|
| **Function Number** | 90 |
| **Data String** | Same parameters as for the File Transfer facility or the 3270 Send/Receive facility. For information on the file transfer facility or the 3270 Send/Receive facility, see the *LANDP Servers and System Management* book. |
| **String Length** | Length of target data string, if not in EOT mode |
| **PS Position** | Number assigned to the drive where the SEND program is located:<br>**0** Current Drive<br>**1** Drive A<br>**2** Drive B<br>**3** Drive C<br>**.** .<br>**26** Drive Z<br>The SEND program must be in the current directory of the specified drive |

| Return Values: | | |
|---|---|---|
| **Data String** | Not used | |
| **Result Code** | Not used | |
| **Return Codes:** | | |
| | **0** | Function successfully completed |
| | **2** | Data string length too long for the LANDP HLLAPI buffer |
| | **9** | System error |

An example on how to call the Send File function follows.

```
Func = 90;          /* send file to host computer function */
DataStr = "C:MYFILE.ASM MYFILE ASMBIN A /H:V1";
DLen = 35;          /* Could be ended with an EOT */
Retc = 3;           /* SEND.COM is on C: drive */
Call HLLAPI ( Func, DataStr, DLen, RetC );
```

Apart from the LANDP HLLAPI return codes, file transfer codes and IBM DOS extended codes can also be returned. The file transfer return codes are incremented by 200 by LANDP HLLAPI. For information on file transfer facility messages, see the *LANDP Problem Determination* book.

The IBM DOS extended return codes are incremented by 300 by LANDP HLLAPI. The most common are the following:

**301** Not valid function
**302** Program file not found
**305** Access denied
**308** Insufficient memory
**310** Not valid environment
**311** Not valid format

## Receive file (function 91)

This function receives a host computer file at the workstation.

| Calling Parameters: | | |
|---|---|---|
| **Function Number** | 91 | |
| **Data String** | Same parameters as for the File Transfer facility or the 3270 Send/Receive facility. For information on the file transfer facility or the 3270 Send/Receive facility, see the *LANDP Servers and System Management* book. | |
| **String Length** | Length of target data string, if not in EOT mode | |
| **PS Position** | Number assigned to the drive where the RECEIVE program is located: | |
| | **0** | Current Drive |
| | **1** | Drive A |
| | **2** | Drive B |
| | **3** | Drive C |
| | **.** | . |
| | **26** | Drive Z |
| | The RECEIVE program must be in the current directory of the specified drive | |

| Return Values: | | |
|---|---|---|
| **Data String** | Not used | |
| **Result Code** | Not used | |
| **Return Codes:** | | |
| | **0** | Function successfully completed |
| | **2** | Data string length too long for the LANDP HLLAPI buffer |
| | **9** | System error |

An example on how to call the Receive File function follows.

```
Func = 91;          /* receive file from host computer function */
DataStr = "C:MYFILE.ASM MYFILE ASMBIN A /H:V1";
DLen = 35;          /* Could be ended with an EOT */
Retc = 3;           /* RECEIVE.COM is on C: drive */
Call HLLAPI ( Func, DataStr, DLen, RetC );
```

Apart from the LANDP HLLAPI return codes, file transfer codes and IBM DOS extended codes can also be returned. The file transfer return codes are incremented by 200 by LANDP HLLAPI. For information on file transfer facility messages, see *LANDP Problem Determination* book.

The IBM DOS extended return codes are incremented by 300 by LANDP HLLAPI. The most common are the following:

**301** Not valid function
**302** Program file not found
**305** Access denied
**308** Insufficient memory
**310** Not valid environment
**311** Not valid format

## Utility services

The function provided supports conversion of presentation space position to screen coordinates. The function Convert Position or RowCol (Function 99) is described in the following §.

## Convert position or rowcol (function 99)

This function converts a presentation space positional value into the screen row and column coordinates, or the screen row and column coordinates into an presentation space positional value. The Convert Position or RowCol function does not change the cursor position.

The conversion takes the presentation space size into account.

| **Calling Parameters:** | | |
|---|---|---|
| **Function Number** | 99 | |
| **Data String** | Presentation space short name followed by one of the letters: | |
| | **P** | Convert Position |
| | **R** | Convert Row/column |
| **String Length** | Row when R specified. Not used when P specified | |
| **PS Position** | Column when R specified. Presentation space position when P specified | |

| **Return Values:** | | |
|---|---|---|
| **Data String** | Not used | |
| **Result Code** | Row. Zero means incorrect row input when P specified | |
| **Return Code** | Contains a status code as follows: | |
| | **0** | Incorrect column when R specified. Incorrect presentation space position when P specified |
| | **9998** | Presentation space short name not valid |
| | **9999** | Second character in data string not P or R |
| | **Other values** | Presentation space position when R specified. Column when P specified. |

An example on how to call the Convert Position or RowCol function follows.

```
Func = 99;          /* PS Position to Row-Col */
DataStr = '1P';
DLen = 2;
RetC = 801;
Call HLLAPI (Func, DataStr, DLen, RetC);
                   /* on return it contains :        */
                   /* DLen = 11                      */
                   /* RetC = 1                       */
```

## Compatibility with other IBM 3270 emulator application programming interfaces

LANDP HLLAPI is based on IBM Personal Computer 3270 Entry Emulator High-Level Language Application Program Interface (EEHLLAPI) and on the IBM 3270 PC High-Level Language Application Program Interface.

LANDP HLLAPI is, except for the Send function and Receive function, compatible with the IBM Communications Server for OS/2 Warp, Version 4 Emulator High-Level Language Interface. These are documented in:

- *Personal Communications/3270 Programmer's Guide for DOS (Entry Level)*
- *Personal Communications/3270 Programmer's Guide for OS/2*
- *Personal Communications/3270 Reference Guide for OS/2*

## Portability considerations

The following differences between LANDP HLLAPI and the above mentioned application programming interfaces must be considered:

- LANDP HLLAPI does not support interpretive BASIC. Application programs written in this language must be modified.

  The following statements are required as part of an interpretive BASIC application program. Remove them before compilation.

  ```
  100 DEF SEG = 0: APILOC=PEEK(&H1FE) + PEEK(&H1FF)
  110 IF APILOC = 0 THEN LPRINT "HLLAPI IS NOT AVAILABLE": END
  120 DATA &H04B8, &HB301, &HCDBB, &HCA7F, &H0002
  130 DIM APICALL% (4)
  140 FOR I=0 TO 4: READ APICALL%(I): NEXT I
  150 BLIMSET = VARPTR (APICALL%(0))
  160 DEF SEG: CALL BLIMSET (APICALL%(1))
  170 BLIM=APICALL%(1): DEF SEG = APICALL%(0)
  180 IF BLIM = &HB301 THEN PRINT "HLLAPI NOT AVAILABLE": END
  ```

- LANDP HLLAPI does not support the Storage Manager function, which is included in other high-level language application programming interfaces, because the Start Host Computer Notification function (see "Start host notification (function 23)" on page 627) requires an allocated queue from the application program. LANDP HLLAPI does not require this queue.

- LANDP HLLAPI does not support the following mnemonics in the Send Key function (see "Send key (function 3)" on page 624).

  | | |
  |---|---|
  | **@P** | Print |
  | **@A@C** | Test |
  | **@A@D** | Word delete |

| | |
|---|---|
| **@A@P** | Ident |
| **@A@R** | Device cancel |
| **@A@d** | Doc Mode |
| **@A@e** | Wrap |
| **@A@f** | Change Format |
| **@A@m** | Cursor Position |

Application programs that use these mnemonics must be modified.

- File transfer programs SEND.COM and RECEIVE.COM use a syntax different from other products. An application program using the Send File function and the Receive File function must take this into consideration, when setting the command line parameters through the *data string* (see “Send file (function 90)” on page 649 and “Receive file (function 91)” on page 651).

  The application program can set the environment variable IPTOPT to ″parms″ and then, call the SEND and RECEIVE programs, without parameters. For information on compatibility with non-LANDP emulators, see the *LANDP Servers and System Management* book.
- LANDP HLLAPI does not support the option TIMEOUT=n because file transfer programs do not display the `In progress ...` message.

  To timeout on long file transfer processes, the application programs can use the relevant command option and specify the timeout value.
- With the Copy OIA function (see “Copy OIA (function 13)” on page 641), LANDP HLLAPI returns more information than other products. The application program can check for system and session status.
- LANDP HLLAPI has an additional function, Set Cursor, that places the cursor at a specified location in the presentation space, see “Set cursor (function 40)” on page 638.

## Writing your own language interface module

This § contains implementation and design details that you need to access LANDP HLLAPI from Macro Assembler/2 or to write your own language interface module (LIM). There are three possibilities:

- Writing a LIM to support any programming language
- Extending an existing LIM, for example, to use other external functions
- Changing the format of the function calls and parameter lists.

The primary functions of a LIM are to:

- Handle linkage to and from the application program
- Obtain data parameter pointers and values
- Convert any language-specific data formats
- Build a parameter control block (PCB) for the resident module
- Call the FBHLLAPI resident module using interrupt X'7F'
- Receive control back from the FBHLLAPI resident module
- Pass parameters and control to the application program

After saving all required registers, the LIM invokes FBHLLAPI resident module, with the following interface:

- A PCB is built by the LIM. Registers DS:SI are set to point to the PCB

- AH is set to X'01' and AL is set to X'04'
- BX is set to X'0000'
- The FBHLLAPI resident module is called using the shared interrupt X'7F'

Before issuing interrupt X'7F', the LIM must verify that the interrupt vector is set for the FBHLLAPI resident interface module. The PCB has the following format:

```
PCB     STRUC
PCB_HDR         DB      'PCB'   ;PCB header (in uppercase)
PCB_FUNC        DB      0       ;Function code value
PCB_DSEG        DW      0       ;String segment address
PCB_DOFS        DW      0       ;String offset address
PCB_LENGTH      DW      0       ;Data length value
                DB      0       ;Not used
PCB_RETCODE     DW      0       ;Return code value
PCB_STRLEN      DW      25000   ;Maximum string size
PCB     ENDS
```

To build the PCB:

- The header PCB (upper case required) starts the control block.
- The function code must be a 1-byte numeric (binary) function code.
- The pointer to the data string contains the segment and offset address. The segment address precedes the offset.

  The string pointer must point to the actual data string, and not point to a length or pointer prefix.
- The length must be the actual binary length value passed by the application program.
- The string length field contains the actual length of the allocated string. If the language used to write the application program provides a length value, the FBHLLAPI resident module can check the length before placing data in the string.

After performing the requested function, the return code and the result code are passed back in the PCB. The data string is passed back directly to the allocated data area. The LIM must retain the parameter addresses to move the return code and result code from the PCB to the application program data area.

## 3270 extended data stream

The LANDP 3270 emulator supports the following write structured fields:

**Outbound (host to 3270 emulator)**

- 3270DS structured field
- Set reply mode structured field
- Erase/reset structured field
- Read partition query structured field

**Inbound (3270 emulator to host)**

- Query reply structured field

`SEND`, `RECEIVE`, and `IND$FILE` use the following subset of the Distributed Data Management (DDM) architecture:

- DDM open structured field

- Acknowledgement for DDM open request
- Acknowledgement for DDM data downloaded (insert normal reply)
- DDM upload data buffer
- Acknowledgement for DDM close
- Acknowledgement for DDM open messages
- Acknowledgement for DDM message
- DDM open for download
- DDM open for upload
- DDM insert with data (download data)
- DDM set cursor and get
- DDM data for get
- DDM close
- DDM error

# Chapter 30. LANDP 3287 printer emulator API

The 3287 printer emulator is an optional **LANDP for DOS** internal application program. It is loaded into storage at LANDP for DOS initialization, and is started by the supervisor. The emulator starts processing when it gets control from the supervisor. The initial conditions depend on customization definitions and whether the operator interface is loaded. The 3287 printer emulator checks for the availability of workstation-attached printers.

You can use the LANDP for OS/2 or LANDP for Windows 3287 emulator in a virtual DOS machine only. The emulator does not support double-byte character string (DBCS) mode operation.

For DBCS mode, information here relating to the IBM 4201 Proprinter applies also to the IBM 5575 Printer or the IBM 5577 Printer for Korea and Taiwan and to the ESC/P printers for the People's Republic of China. Information relating to the IBM 4712 Transaction Printer or the IBM 4722 Document Printer applies also to the IBM 4748 Document Printer.

## IBM 3287 printer emulator characteristics

The main characteristics of the 3287 printer emulator are:

- The 3287 emulator (EMU3287) supports up to three printers attached to parallel ports or redirected by the operating system to a send port or a LAN-attached printer. The printers can be IBM Proprinters or HP laser-jet printers (IBM 40x9 printers).

  The EMU3287R program supports one printer redirected to the LANDP financial printer server.

  **Note:** The 3287 printer emulator can be assigned only one type of printer—either a 47x2 printer or up to three 4201 Proprinters. Thus, all the host computer sessions that use the 3287 printer emulator are served by the same type of printer.
- The 3287 printer emulator is compatible with both local mode printer use and other LANDP component printer use.
- The 3287 emulator can print, through the parallel port, on IBM 4019 and 4029 Laser Printers in PPDS mode.
- The 3287 emulator can print, through the parallel port, on IBM 4019, 4029, and 4039 Laser Printers in HP/PCL emulation mode.
- The 3287 printer emulator uses the SNA server (see Chapter 2, "SNA communication server" on page 35 and interacts with logical units, type 1 (LU 1). The LU 1 sessions are used for communication with one or more host computers. Up to five LU 1 sessions, numbered from 1 to 5, can be defined. The number assigned corresponds to the priority for dealing with the print requests.
- SNA character strings (SCS) are received from the host computer over the LU 1 sessions. The string is interpreted and printed.

© Copyright IBM Corp. 1992, 2002

- When the 3287 printer emulator uses a 47x2 printer, the operator interface and the local resource manager have no control over the 3287 printer emulator.

The LANDP program provides an EBCDIC-to-ASCII translation table that contains translations from the International Host Code Page 500 (EBCDIC) to PC Code Page 850 (ASCII). This table can be modified at customization. It is not used with DBCS mode.

All printers using the same 3287 printer emulator work with the same EBCDIC-to-ASCII translation table. A different EBCDIC-to-ASCII translation table can be defined for each 3287 printer emulator. Only one 3287 printer emulator can be installed in each workstation.

## Accessing the IBM 3287 printer emulator

With a 4201 Proprinter, you can access the 3287 printer emulator by using the operator interface or the local resource manager. If you run the 3287 printer emulator in a virtual DOS machine or redirect the 3287 printer emulator output to a 47x2 printer, the operator interface and the local resource manager have no control over the 3287 printer emulator.

The operator interface and the local resource manager allow the workstation operator to communicate with the 3287 printer emulator. For more information, refer to the *LANDP Servers and System Management* book and Chapter 22, “Local resource manager” on page 513.

You can also use the printer manager server to control access to these printers (see Chapter 11, “Printer manager server” on page 199).

Using the operator interface or the local resource manager you can:

- Get information about, for example, the 3287 printer emulator, the printer status, the communication status, and the working mode.
- Issue commands to the 3287 printer emulator to change, for example, print density modes and printer assignments.

*Table 48. Function Codes used with the 3287 printer emulator*

| Function code | Description | Env. | Page |
|---|---|---|---|
| AI | Activate application intervention | 0--- | 522 |
| AS | Activate service | 0--- | 523 |
| CA | Change printer assignment | 0--- | 524 |
| CC | Cancel pending functions | 0--- | 526 |
| CM | Change print density mode | 0--- | 528 |
| CP | Change LU priority | 0--- | 529 |
| EI | End operator intervention | 0--- | 531 |
| EL | End listing | 0--- | 533 |
| GS | Get status | 0--- | 534 |
| IS | Initiate session | 0--- | 536 |

*Table 48. Function Codes used with the 3287 printer emulator (continued)*

| Function code | Description | Env. | Page |
|---|---|---|---|
| SS | Suspend service | 0--- | 537 |
| TS | Terminate session | 0--- | 539 |

The first and second columns give the function code and the name of the function. The third column shows the operating environment of the function. "0---" means that it is available from LANDP for DOS servers only. The last column refers to the page where you can find the function described.

## Facilities not supported

The 3287 printer emulator does not support all the facilities of an IBM 3287 Printer, because of hardware restrictions in the IBM 4201 Proprinter, the IBM 4712 Transaction Printer, and the IBM 4722 Document Printer. The facilities not supported are:

- Audible alarm
- Extended highlighting
- Index switch
- Mono/dual case switch/light
- Programmed symbols
- Reset switch
- SCS data structured fields
- Setup switch
- Test switch/light
- Underscore
- X print error indication

## Host computer communication

The SNA server supports different types of logical units (LUs), which are used for different purposes. Logical unit type 1 (LU 1) is used by the 3287 printer emulator to print SNA Character Strings (SCS).

The main characteristics of host-computer-to-printer communication are:

- Connection is always initiated by the host computer.
- If the printer is available, a host computer request receives immediate attention.
- The priorities of the LUs that use the same printer, and have pending print requests, are assigned from 1 to 5 consecutively. This priority, which also defines the order of service, corresponds to the LU session number. When printing has ended for the LU with the highest priority, the priorities are shifted up in the queue and the LU just being serviced has the lowest priority.
- After a host computer or communication problem, the host computer automatically reconnects and initiates the session. Virtual Telecommunications Access Method/Network Control Program (VTAM/NCP) parameters must be properly defined—see the *LANDP Installation and Customization* book.

  If X.25 data link control is used, host computer operator intervention is often needed for a restart.

If more than one LU is assigned to the same printer, the 3287 printer emulator waits for a *timeout between listings* interval for the session to be reestablished. If recovery does not occur within that interval, the next LU is receives attention.

- For X.25 data link control, an automatic session end is caused by the 3287 printer emulator when the end of the listing is detected. If the DISCONNECT=YES parameter is defined in the NCP physical unit (PU) macros and all the LUs of the PU are logged off, an automatic release occurs.
- Any BIND sent by a host computer application must have *brackets*. If it does not, the BIND is rejected by the SNA server (see “BIND parameters” on page 38).
- An SCS parameter error results in a negative response to the host computer (sense code X'1005').
- A printer *not on-line* does not generate a message to the host computer. The 3287 printer emulator waits for the *on-line* status.
- When a 4201 Proprinter runs out of paper or is in *power off* condition for more than five minutes, the 3287 printer emulator produces *notify power off* and a negative response (sense code X'0831') to the host computer.

  When a 47x2 printer runs out of paper no notification is sent to the host computer.
- A successful recovery from *out of paper* or from *power off* condition produces *notify power on* and an LUSTAT (sense code X'082B') to the host computer. Then normal operation resumes.
- With a 4201 Proprinter, the 3287 printer emulator session can be ended through the operator interface.
- The 3287 printer emulator does not provide an automatic way of detecting paper jams at the printer it serves.

## SNA character string control codes

The SNA character string (SCS) control codes used by the IBM 3287 Printer and supported by the 3287 printer emulator are:

| Code | EBCDIC value | Function |
|---|---|---|
| BS | X'16' | Back space |
| BEL | X'2F' | Bell facility — the IBM 3287 Printer buzzer sounds until the *hold print* key is pressed, and the printer controlled by the 3287 printer emulator produces a short acoustic alarm |
| CR | X'0D' | Carriage return |
| ENP | X'14' | Enable presentation — results in *no operation* |
| FF | X'0C' | Forms feed |
| GE | X'08' | Graphic escape — is substituted by a hyphen |
| HT | X'05' | Horizontal tab |
| INP | X'24' | Inhibit presentation — results in *no operation* |

| Code | EBCDIC value | Function |
|---|---|---|
| IRS | X'1E' | Interchange record separator — is treated as a new line (NL) code |
| LF | X'25' | Line feed |
| NL | X'15' | New line |
| SHF | X'2BC1' | Set horizontal format — up to 28 horizontal tabs are supported |
| SLD | X'2BC6' | Set line density |
| SVF | X'2BC2' | Set vertical format — up to 64 vertical tabs are supported |
| TRN | X'35' | Transparent — a byte is sent that indicates the string length, following the control code. The string is sent to the printer without translation |
| VCS | X'04xx' | Vertical channel select — is regarded as a line feed (LF) code |
| VT | X'0B' | Vertical tab |

**Notes:**

1. Some of these control codes, although supported, are handled in a different way. This is due to the different hardware characteristics between an IBM 3287 Printer and an IBM 4201 Proprinter, IBM 4712 Printer, and IBM 4722 Printer.
2. Other control codes are substituted by a specific character, selected at customization. This does not involve sending a response to the host computer.
3. If the SET code (X'2B') is not SHF, SLD, or SVF, a negative response is sent to the host computer (sense code X'1003'). Even though the *power off* key and then the *power on* key are pressed, the 3287 printer emulator keeps the SHF, SLD, or SVF format. The format is no longer kept after an IPL.

## User exits

The 3287 printer emulator provides three exits, allowing a user-written server to change buffer data to be printed, to translate control characters contained in an SNA Character String (SCS), and to know when a listing is ended. The user-written server must be named EXFS and must provide three functions:

- Print Data (PD)
- Data Received (DR)
- End Listing (EL)

(See also “Server exit for user data processing” on page 107 for another use of the EXFS server.)

The 3287 printer emulator issues a call to the EXFS server when there is data to print, or after reading a message from SNA buffers for the 3287 emulator, or when a listing is ended.

## Writing the EXFS server

*Table 49. Function Codes used in the EXFS User-written Exit Server*

| Function code | Description | Env. | Page |
|---|---|---|---|
| **Functions available from the 3287 emulator:** | | | |
| DR | Data received | 0--- | 663 |
| EL | End listing | 0--- | 664 |
| PD | Print data | 0--- | 664 |
| **Functions available from the program-to-program communication emulator:** | | | |
| PR | Process read data | -26- | 107 |
| PS | Process send data | -26- | 107 |

This section provides guidelines to help you supply the necessary information in the Request CPRB fields and understand the information you receive in the Reply CPRB fields. If you need more information about the CPRB fields, see Appendix A, “Connectivity programming request block” on page 669.

| CPRB Fields on Request | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 10 | 2 | | Function code |
| 14 | 2 | | Request PARMLIST length |
| 16 | 4 | Address | Request PARMLIST address |
| 20 | 2 | | Request DATA length |
| 22 | 4 | Address | Request DATA address |
| 26 | 2 | | Reply PARMLIST length |
| 28 | 4 | Address | Reply PARMLIST address |
| 32 | 2 | | Reply DATA length |
| 34 | 4 | Address | Reply DATA address |
| 94 | 2 | 8 | Server name length |
| 96 | 8 | EXFS | Server name |

The following fields are variable and are discussed in each function request description:

- Function code
- Request PARMLIST length
- Request DATA length
- Reply PARMLIST length
- Reply DATA length

| CPRB Fields on Reply | | | |
|---|---|---|---|
| **Offset** | **Length** | **Value** | **Content** |
| 4 | 4 | | Router return code |
| 40 | 4 | | Server return code |
| 44 | 2 | | Replied PARMLIST length |
| 46 | 2 | | Replied DATA length |

If the request was successful, the *router return code* is X'00000000'. The *server return code* should also be X'00000000'. See the *LANDP Problem Determination* book for the treatment given to errors in LANDP-supplied servers. The return values in Reply PARMLIST and Reply DATA should be ignored when an error occurs.

The following fields are variable and are discussed in each function request description:
- Replied PARMLIST length
- Replied DATA length

## Data received (DR function)

Sometimes it is necessary to manipulate a SNA Character String (SCS) received in EBCDIC, from the Host. The need arises when a 4700 listing contains SCS control characters that are not supported by the 3287 emulator, and when a “bypass” of these characters by the 3287 emulator is not required. Instead of being “bypassed”, the characters should be translated to supported control characters (an example is the X'34' Print Position Control Character).

The DR function is requested immediately after a message has been read from the SNA buffers for the 3287 emulator. The Request DATA supplied with the DR function is the same as the Reply DATA for a Read Host (RH function) using the SNA Server.

| **CPRB Field** | **Content/Description** |
|---|---|
| Function code | DR |
| Request DATA length | 1 to 4096 |
| Request PARMLIST length | 26 |
| Reply DATA length | 4096 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 4096 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | SNA session ID |
| 2 | 1 | Printer identifier (from 1 to 3 in ASCII) |

The data received in Reply DATA area contains the data that the 3287 emulator translates to ASCII and sends to be printed.

## End listing (EL function)

When a listing is ended, the 3287 printer emulator issues a call to the End Listing function, and the EXFS server is notified that the listing is ended.

The 3287 printer emulator assumes end of listing when:

- It receives the end indicator ENDL=*nnnnnnnn*, provided the host computer application controls end of listing. The first line of the listing contains: LIST=*nnnnnnnn*. (*nnnnnnnn* is an eight-character string, defined at customization, to identify the beginning and the end of the listings.)
- It detects the *end bracket*, and the timeout between listings has elapsed. The timeout is defined at customization.

Depending on your environment, you should decide which end listing control is best. Using the *end indicator* is most common. However, using the *end bracket* plus timeout enables error recovery without restarting. It may be useful when printing large listings containing several *end brackets*.

| CPRB Field | Content/Description |
|---|---|
| Function code | EL |
| Request DATA length | 0 |
| Request PARMLIST length | 26 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | SNA session ID |
| 2 | 1 | Printer identifier (from 1 to 3 in ASCII) |

## Print data (PD function)

When there is data to print, the 3287 printer emulator issues a call to the Print Data function, and the EXFS server receives the complete buffer of ASCII characters that is to be printed, in the Request DATA area. The PD function can change, add, or suppress, any character in the buffer with the following constraints:

- The Replied DATA length must not be greater than 4096 bytes.
- Any character changed, added, or suppressed that modifies the current print position may cause unpredictable results in the listing.

| CPRB Field | Content/Description |
|---|---|
| Function code | PD |
| Request DATA length | 1 to 4096 |
| Request PARMLIST length | 26 |
| Reply DATA length | 4096 |
| Reply PARMLIST length | 26 |
| Replied DATA length | 4096 |
| Replied PARMLIST length | 26 |

| Request PARMLIST Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 2 | SNA session ID |
| 2 | 1 | Printer identifier (from 1 to 3 in ASCII) |

The data in the Reply DATA area is sent to the printer. However, the EXFS server can tell the 3287 printer emulator not to print the data, by issuing an NP return code.

# Part 9. Appendixes

© Copyright IBM Corp. 1992, 2002

# Appendix A. Connectivity programming request block

This appendix list the fields held within the connectivity programming request block.

| Offset | Length | Field Name | Content/Description |
|---|---|---|---|
| X'00' (00) | 4 | ehc_reserved1 | Reserved |
| X'04' (04) | 4 | ehcretcode | Router return code |
| X'08' (08) | 1 | ehcverb_type | Verb type (provided by LANDP) |
| X'09' (09) | 1 | ehc_flags | Flags (some reserved) |
| X'0A' (10) | 2 | ehcfunct | Function code |
| X'0C' (12) | 2 | ehctimeout | Timeout per request |
| X'0E' (14) | 2 | ehcqparml | Request PARMLIST length |
| X'10' (16) | 4 | ehcqparmad | Request PARMLIST address |
| X'14' (20) | 2 | ehcqdatal | Request DATA length |
| X'16' (22) | 4 | ehcqdataad | Request DATA address |
| X'1A' (26) | 2 | ehcrparml | Reply PARMLIST length |
| X'1C' (28) | 4 | ehcrparmad | Reply PARMLIST address |
| X'20' (32) | 2 | ehcrdatal | Reply DATA length |
| X'22' (34) | 4 | ehcrdataad | Reply DATA address |
| X'26' (38) | 2 | ehc_event_id | Event ID for LANDP for OS/2 or Windows ZN function, or reserved |
| X'28' (40) | 4 | ehcservrc | Server return code provided by the server |
| X'2C' (44) | 2 | ehcrepldplen | Replied PARMLIST length provided by the server |
| X'2E' (46) | 2 | ehcreplddlen | Replied DATA length provided by the server |
| X'30' (48) | 2 | ehcpc_id | Originator workstation ID |
| X'32' (50) | 8 | ehcresource_origin | Originator resource name |
| X'3A' (58) | 2 | ehcdest_pc_id | Destination workstation ID |
| X'3C' (60) | 6 | ehc_reserved5 | Reserved |
| X'42' (66) | 2 | ehcfirst_pc_origin | First originator workstation ID of a server-to-server request |

© Copyright IBM Corp. 1992, 2002

| Offset | Length | Field Name | Content/Description |
|---|---|---|---|
| X'44' (68) | 8 | ehcfirst_res_origin | First originator resource name of a server-to-server request |
| X'4C' (76) | 1 | ehcfields_meaningful | Server-to-server request indicator provided by LANDP |
| X'4D' (77) | 1 | ehc_reserved6 | Reserved |
| X'4E' (78) | 2 | ehcfirst_pid_origin | First originator process ID of a server-to-server request |
| X'50' (80) | 4 | ehc_reserved7 | Reserved |
| X'54' (84) | 2 | ehcpid_origin | Originator process ID |
| X'56' (86) | 2 | ehcpid_dest | Destination process ID |
| X'58' (88) | 6 | ehc_reserved8 | Reserved |
| X'5E' (94) | 2 | ehcservnamlen | Destination server name length: always equal to X'0008' |
| X'60' (96) | 8 | ehcserver | Destination server name |

# Appendix B. Switch printer mode within an application (Z6 function)

When BPP is used to drive a serial attached printer using LANDP for OS/2, the Z6 application programming function allows a printer to be switched between local and remote modes from within an application program. In local mode, the printer is available for use by workstation applications; in remote mode, the printer is assigned to the IBM 4700 Finance Communication Processor host system.

**Note:** For local printing to a serial attached printer, the PRTMON program must have been started before the AUTOFBSS.CMD was issued.

| CPRB Field | Content/Description |
|---|---|
| Function code | Z6 |
| Request DATA length | 3 or 7 |
| Request PARMLIST length | 0 |
| Reply DATA length | 0 |
| Reply PARMLIST length | 0 |
| Replied DATA length | 0 |
| Replied PARMLIST length | 0 |

| Request DATA Values | | |
|---|---|---|
| **Offset** | **Length** | **Content** |
| 0 | 1 | 'S' for status - Request DATA length is 3<br>'C' for command - Request DATA length is 7 |
| 1 | 2 | ## SNA session ID |
| 3 | 2 | 'xxx' Requested mode<br>- 'PH' Switch to remote (host) mode<br>- 'PL' Switch to local mode |
| 5 | 1 | Forced switches:<br>- 'F' Force the switch<br>- ' ' Do not force the switch |
| 6 | 1 | Reserved |

**Example:** 'CXXPL ' in the Request DATA area of a Z6 call sets the printer to local mode, without forcing it.

For further details, see the information on printer sharing in *Financial Branch System Integrator/2 Programmer's Reference* and the description of the BPP parameters in the *LANDP Installation and Customization* manual.

© Copyright IBM Corp. 1992, 2002

# Appendix C. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

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

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:**
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS " WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM documentation or non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those documents or Web sites. The materials for those documents or Web sites are not part of the materials for this IBM product and use of those documents or Web sites is at your own risk.

© Copyright IBM Corp. 1992, 2002

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

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

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

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

## Trademarks and service marks

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

*Table 50. Trademarks and service marks*

| | |
|---|---|
| ACF/VTAM | LAN Distance |
| AnyNet | LANDP |
| Application System/400 | Macro Assembler/2 |
| AS/400 | Micro Channel |
| AT | MQSeries |
| BookManager | Operating System/2 |
| C/2 | OS/2 |
| CICS | OS/390 |
| COBOL/2 | RETAIN |
| Common User Access | S/390 |
| CUA | ThinkPad |
| DB2 Universal Database | VisualAge |
| Distributed Database Connection Services/2 | VisualGen |
| Distributed Relational Database Architecture | VM/ESA |
| DRDA | VTAM |
| IBM | WIN-OS/2 |
| IBMLink | Workplace Shell |
| IMS/ESA | WebSphere |

Lotus is a registered trademark of Lotus Development Corporation in the United States and other countries.

Tivoli and NetView are registered trademarks of Tivoli Systems Inc. in the United States and other countries. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc in the United States and other countries.

Microsoft, Windows, Windows NT, Windows 2000, and the Windows logo, are registered trademarks of Microsoft Corporation.

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

Intel is a registered trademark of Intel.

PC/TCP and NetManage are registered trademarks of NetManage Inc.

Red Hat is a registered trademark of Red Hat, Inc.

SuSE and its logo are registered trademarks of SuSE AG.

Caldera Systems, the C-logo, and OpenLinux are either registered trademarks or trademarks of Caldera Systems, Inc.

TurboLinux is a registered trademark of TurboLinux, Inc.

Other company, product, and service names may be trademarks or service marks of others.

# Glossary

This glossary includes abbreviations, terms, and definitions used in the IBM LANDP Licensed Programs Family publications. It does not include all terms previously established for IBM networks, programs, operating systems, or other IBM products.

If you do not find the term you are looking for, refer to the *IBM Dictionary of Computing*.

This glossary includes terms and definitions from the following sources:

- The *IBM Dictionary of Computing*, New York: McGraw-Hill, copyright 1994 by International Business Machines Corporation. Copies may be purchased from McGraw-Hill or in bookstores.
- The *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.
- The ANSI/EIA Standard—440-A, *Fiber Optic Terminology*. Copies may be purchased from the Electronic Industries Association, 2001 Pennsylvania Avenue, N.W., Washington, DC 20006. Definitions are identified by the symbol (E) after the definition.
- The *Information Technology Vocabulary*, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

Definitions that are specific to IBM products are so labeled, for example, “In LANDP,” or “In SNA.”

## A

**abend.** Abnormal end of task.

**Abnormal end of task.** Termination of a task before its completion because of an error condition that cannot be resolved by recovery facilities while the task is executing.

**abstract class.** A class that provides common information for subclasses, and that therefore cannot be instantiated. Abstract classes provide at least one abstract method.

**abstract method.** A method with a signature, but no implementation. You provide the implementation of the method in the subclass of the abstract class that contains the abstract method.

**account.** In the AIX operating system, the log-in directory and other information that gives a user access to the system.

**ACF.** Advanced Communications Function

**ACF/NCP.** Advanced Communications Function for the Network Control Program.

**activate logical unit request (ACTLU).** A request, sent by the host to the LANDP SNA server, to establish a logical session.

**activate physical unit request (ACTPU).** A request, sent by the host to the LANDP SNA server, to establish a physical session.

**active.** In an XLR environment, the server (and by implication, the workstation) that handles client requests and sends logging data to the backup.

**ACTLU.** Activate logical unit request.

**ACTPU.** Activate physical unit request.

© Copyright IBM Corp. 1992, 2002

**adapter.** A part that electrically or physically connects a device to a computer or to another device. A printed circuit board that modifies the system unit to allow it to operate in a particular way.

**address.** The unique code assigned to each device or workstation connected to a network. A standard Internet address is a 32-bit address field. This field can be broken into two parts. The first part contains the network address; the second part contains the host number.

**Advanced Communications Function (ACF).** A group of IBM licensed programs, principally VTAM programs, TCAM, NCP, and SSP, that use the concepts of Systems Network Architecture (SNA), including distribution of function and resource sharing. See also Network Control Program (NCP).

**Advanced Communications Function for the Network Control Program (ACF/NCP).** An IBM program product that provides communication controller support for single-domain, multiple-domain, and interconnected network capability. See also Advanced Communications Function (ACF) and Network Control Program (NCP).

**advanced program-to-program communication (APPC).** The general facility characterizing the LU 6.2 architecture and its various implementations in products.

**AID.** Attention identifier.

**AIX (Advanced Interactive Executive).** IBM's licensed version of the UNIX operating system.

**alert.** A message sent to a management services focal point in a network to identify a problem or an impending problem. In the NetView program, a high-priority event that warrants immediate attention. A database record is generated for certain event types that are defined by user-constructed filters.

**alert condition.** A problem or impending problem for which information is collected and possibly forwarded for problem determination, diagnosis, or resolution.

**alert description.** Information in an alert table that defines the contents of a Systems Network Architecture (SNA) alert for a particular message ID.

**alert focal point.** The system in a network that receives and processes (logs, displays, and optionally forwards) alerts. An alert focal point is a subset of a problem management focal point.

**alert ID number.** A value created from specific fields in the alert using a cyclic redundancy check. A focal point uses this value to refer to a particular alert, for example, to filter out duplicate alerts.

**alert type.** A value in an alert that indicates the problem being reported.

**American National Standards Institute (ANSI).** An organization consisting of producers, consumers, and general interest groups, that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States. (A)

**ANSI.** American National Standards Institute.

**APAR.** Authorized program analysis report.

**API.** Application program interface.

**APPC.** Advanced program-to-program communication.

**applet.** A Java program designed to run within a Web browser. Contrast with application.

**application.** In LANDP, a program using IBM LANDP for DOS, IBM LANDP for OS/2, IBM LANDP for Windows, IBM LANDP for Linux, IBM FBSS/2, IBM PC/Integrator, or IBM PC Integrator/2, tailored to the needs of the workstation user. The use to which an information processing system is put; for example, a payroll application, an airline reservation application, a network application. A collection of software components used to perform specific types of user-oriented work on a computer. In Java programming, a self-contained, stand-alone Java program that includes a static main method. It does not require an applet viewer. Contrast with applet.

**application program.** A program that is specific to the solution of an application problem. Synonymous with application software. (T) A program written for or by a user that applies to the user's work, such as a program that does inventory control or payroll. A program used to connect and communicate with stations in a network, enabling users to perform application-oriented activities.

**application program interface (API).** In LANDP, the common interface by which server functions are requested. Requests are expressed by issuing a call to the supervisor. A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program. The interface through which an application program interacts with an access method.

**application software.** Software that is specific to the solution of an application problem. (T) Synonymous with application program. Software coded by or for an end user that performs a service or relates to the user's work. Software products such as games, spreadsheets, and word processing programs designed for use on a personal computer.

**argument.** An independent variable. (I) (A) Any value of an independent variable; for example, a search key; a number identifying the location of an item in a table. (I) (A) A parameter passed between a calling program and a called program.

**arrival sequence.** An order in which records are retrieved that is based on the order in which records are stored in a physical file.

**AS/400®.** IBM Application System/400®.

**ASCII (American National Standard Code for Information Interchange).** The standard code, using a coded character set consisting of 7-bit coded characters (8-bits including parity check), used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters. (A)

**Note:** IBM has defined an extension to ASCII code (characters 128-255).

**ASCIIZ format.** A string of ASCII characters ending with a null character (X'00').

**ASYNC.** Asynchronous.

**asynchronous (ASYNC).** Pertaining to two or more processes that do not depend upon the occurrence of specific events such as common timing signals. (T) Without regular time relationship; unexpected or unpredictable with respect to the execution of program instructions.

**attention identifier (AID).** A code in the inbound 3270 data stream that identifies the source or type of data that follow. A character in a data stream indicating that the user has pressed a key, such as the Enter key, that requests an action by the system.

**authorization.** In computer security, the right granted to a user to communicate with or make use of a computer system. (T) An access right. The process of granting a user either complete or restricted access to an object, resource, or function.

**authorized program analysis report (APAR).** A report of a problem caused by a suspected defect in a current unaltered release of a program.

# B

**back-out.** To restore a file to a previous condition by removing changes in the inverse chronological order from which the changes were originally made.

**backup.** In an XLR environment, the server (and, by implication, the workstation) that accepts logging data from the active and maintains a mirror set of databases (at a transaction level).

**BASIC.** Beginner's all-purpose symbolic instruction code. A procedural algebraic language originally designed for ease of learning with a small instruction repertoire. (A) A high-level programming language with a small number of statements and a simple syntax that is designed

to be easily learned and used and that is widely used for interactive applications on microcomputers.

**Basic Input/Output System (BIOS).** Code that controls basic hardware operations, such as interactions with diskette drives, hard disk drives, and the keyboard. See also NetBIOS.

**BAT, bat.** A DOS batch file extension (.BAT). A batch file that contains a series of commands to be processed sequentially.

**BB.** Begin bracket.

**begin bracket (BB).** An SNA bracket protocol term issued by the LANDP SNA server when bracket protocol is requested in the bind session. Contrast with end bracket.

**BID.** In SNA, a request to start a bracket.

**bind.** To associate a variable with an absolute address, identifier, or virtual address, or with a symbolic address or label in a program.

**BIND.** In SNA, a request to start a session between two logical units. Contrast with UNBIND.

**binding.** In programming, an association between a variable and a value for that variable that holds within a defined scope. The scope may be that of a rule, a function call, or a procedure invocation. (T) In the AIX operating system, a temporary association between a client and both an object and a server that exports an interface to the object. A binding is meaningful only to the program that sets it and is represented by a bound handle.

**BIOS.** Basic Input/Output System.

**block.** The smallest complete unit of data that can be transmitted between units in a communication network. The maximum size of a block depends on the characteristics of the sending or receiving unit. A group of contiguous characters recorded as a unit. See also connectivity programming request block, program control block.

**browser.** An Internet-based tool that lets users browse web sites.

**buffer.** A routine or storage used to compensate for a difference in rate of flow of data, or time of occurrence of events, when transferring data from one device to another. (A) A portion of storage used to hold input or output data temporarily.

# C

**C language.** A language used to develop software applications in compact, efficient code that can be run on different types of computers with minimal change.

**call.** In LANDP, the invocation of one of the LANDP API routines, RMTREQ, GETRPLY and RMTAREQ (client calls) and GETREQ, RMTRPLY, and SRVINIT (server calls). A LANDP client uses the RMTREQ call to request a LANDP function. Calls use the connectivity programming request block (CPRB) to pass and receive information. The syntax of a call varies with the programming language. The following examples are for COBOL and C respectively

```
CALL "RMTREQ" USING BY REFERENCE EHC-CPRB
                    BY VALUE     EHC-RESERVED

retcode = GETREQ(&mycprb, EHC_RESERVED);
```

**CCITT.** Comité Consultatif International Télégraphique et Téléphonique. The International Telegraph and Telephone Consultative Committee.

**CD.** Compact disc.

**CD-ROM.** Compact disc-read-only memory.

**CICS®.** Customer Information Control System.

**CID.** Configuration, Installation, and Distribution. An IBM standard methodology for installing and distributing products under DOS, OS/2, and Windows 3.1.

**ciphertext.** In computer security, text produced by encryption. Synonym for enciphered data.

**cleartext.** Nonencrypted data. Synonym for plaintext.

**class.** An encapsulated collection of data and methods to operate on data. A class can be instantiated to produce an object that is an instance of the class.

**CLASSPATH.** In your deployment environment, the environment variable keyword that specifies the directories in which to look for class and record files.

**client.** A functional unit that receives shared services from a server. (T) A user. See also client/server, client workstation, server, and user.

**client workstation.** In IBM LANDP for DOS, IBM LANDP for OS/2, IBM LANDP for Linux, IBM LANDP for Windows, IBM FBSS/2, IBM PC/Integrator, and IBM PC Integrator/2, a workstation in a LAN that uses a service. See also client, client/server, server, and user.

**client/server.** In communications, the model of interaction in distributed data processing in which a program at one site sends a request to a program at another site and awaits a response. The requesting program is called a client; the answering program is called a server. See also client, client workstation, server, and user.

**CLIST, clist.** Command list.

**close.** A LANDP family function used to release a server. To end the processing of a file. A data manipulation function that ends the connection between a file and a program. Contrast with open.

**COBOL.** Common business-oriented language. A high-level programming language, based on English, that is used primarily for business applications.

**code page.** An assignment of graphic characters and control function meanings to all code points; for example, assignment of characters and meanings to 256 code points for an 8-bit code, assignment of characters and meanings to 128 code points for a 7-bit code.

**collating sequence.** A specified arrangement used in sequencing. (I) (A)

**COM, com.** A DOS file with the file extension .COM.

**command.** Loosely, a mathematical or logic operator. (A) A request from a terminal for performance of an operation or processing of a program. A character string from a source external to a system that represents a request for system action.

**command list (CLIST, clist).** A list of commands and statements designed to perform a specific function for the user.

**Common User Access™ architecture.** Guidelines for the dialog between a human and a workstation or terminal. One of the three SAA architectural areas.

**communication configuration.** In LANDP, the process of selecting and describing to the LANDP programs the particular arrangement of communication functions about a particular user.

**communication controller.** A device that directs the transmission of data over the data links of a network; its operation may be controlled by a program executed in a processor to which the controller is connected or it may be controlled by a program executed within the device. (T) A type of communication control unit whose operations are controlled by one or more programs stored and executed in the unit. It manages the details of line control and the routing of data through a network.

**communication server.** A server that communicates with a remote computer for various workstations in a local area network.

**Communications Server.** An IBM licensed program that supports the development and use of OS/2 applications involving two or more connected systems or workstations. IBM SecureWay Communications Server for OS/2 Warp provides multiple concurrent connectivities using different protocols for IBM 3270 and 5250 emulation sessions, printer sessions, and file transfers. It supports a range of application programming interfaces (API), which may be called concurrently and are designed for a variety of applications. IBM

SecureWay Communications Server for OS/2 Warp includes the necessary interfaces for network management.

**compact disc (CD).** A disc, usually 4.75 inches in diameter, from which data is read optically by means of a laser. A disc with information stored in the form of pits along a spiral track. The information is decoded by a compact-disc player and interpreted as digital audio data, which most computers can process.

**compact disc-read-only memory (CD-ROM).** A 4.75-inch optical memory storage medium, capable of storing about 550 megabytes of data. The standards for CD-ROM storage are known as the “Yellow Book.”

**compaction.** Any method for encoding data to reduce the storage it requires. In SNA, the transformation of data by packing two characters in a byte so as to take advantage of the fact that only a subset of the allowable 256 characters is used; the most frequently sent characters are compacted. See also compression and encode.

**compression.** The process of eliminating gaps, empty fields, redundancies, and unnecessary data to shorten the length of records or blocks. In SNA, the replacement of a string of up to 64 repeated characters by an encoded control byte to reduce the length of the data stream sent to the LU-LU session partner. The encoded control byte is followed by the character that was repeated (unless that character is the prime compression character). Contrast with decompression.

**config.sys.** A file created during the customization process that holds the details about the system configuration. The CONFIG.SYS file is used during system operation.

**configuration.** The manner in which the hardware and software of an information processing system are organized and interconnected. (T) The physical and logical arrangement of devices and programs that make up a data processing system. The devices and programs that make up a system, subsystem, or network.

**connection.** An association established between functional units for conveying information. The path between two protocol modules that provide reliable stream delivery service. On the Internet, a connection extends from a TCP module on one machine to a TCP module on the other.

**connectivity.** The capability to attach a variety of functional units without modifying them.

**connectivity programming request block (CPRB).** The control block used for communication between a server and a client. This control block contains the information that is exchanged between clients and servers, and the information required for routing the requests and replies.

**constructor.** A method called to set up a new instance of a class.

**control program.** A computer program designed to schedule and supervise the execution of programs of a computer system. (I) (A)

**coprocessor.** A supplementary processor that performs operations in conjunction with another processor. In personal computers, a microprocessor on an expansion board that extends the address range of the processor in the system unit or adds specialized instructions to handle a particular category of operations; for example, an I/O coprocessor, math coprocessor, or networking coprocessor.

**corrective service diskette.** A diskette provided by IBM to registered service coordinators for resolving user-identified problems with previously installed software. This diskette includes program updates designed to resolve problems.

**CPRB.** Connectivity programming request block.

**CRC.** The cyclic redundancy check character. (A)

**critical error handler.** A routine that the operating system calls automatically if an error occurs in an operating system function call. There is a standard error handler or the user can provide one for special functions.

**CRV.** Cryptography verification request.

**cryptography.** The transformation of data to conceal its meaning. In computer security, the principles, means, and methods for encrypting plaintext and decrypting ciphertext.

**cryptography key.** A parameter that determines cryptographic transformations between plaintext and ciphertext.

**cryptography verification (CRV) request.** A request unit sent by the primary logical unit (PLU) to the secondary logical unit (SLU) as part of cryptographic session establishment, to allow the SLU to verify that the PLU is using the correct session cryptography key and initialization vector (IV).

**CTS.** Clear to Send.

**CUA™ architecture.** Common User Access™ architecture.

**cursor.** A movable, visible mark used to show a position of interest on a display surface. (A) In SAA Common User Access architecture, a visual cue that shows a user where keyboard input will appear on the screen.

**Customer Information Control System (CICS®).** An IBM licensed program that allows transactions entered at remote terminals to be processed concurrently by user-written application programs. It includes facilities for building, using, and maintaining databases.

**Customer Information Control System for Virtual Storage (CICS/VS).** An IBM licensed program used in a communications network.

**customization.** The process of designing a data processing installation or network to meet the requirements of particular users.

**customization workstation.** A workstation on which LANDP is installed, and which is used to customize a LANDP workgroup.

**cyclic redundancy check character (CRC).** A character used in a modified cyclic code for error detection and correction. (A)

# D

**DASD.** Direct access storage device.

**data circuit-terminating equipment (DCE).** In a data station, the equipment that provides the signal conversion and coding between the data terminal equipment (DTE) and the line. (I)

**Notes:**

1. The DCE may be separate equipment or a part of the DTE or an integral part of the DTE or of the intermediate equipment.
2. A DCE may perform other functions that are usually performed at the network end of the line.

**Data Encryption Standard (DES).** In computer security, the National Institute of Standards and Technology (NIST) Data Encryption Standard, adopted by the U.S. government as Federal Information Processing Standard (FIPS) Publication 46, which allows only hardware implementations of the data encryption algorithm.

**data flow control (DFC).** In SNA, a request/response unit (RU) category used for requests and responses exchanged between the data flow control layer in one half-session and the data flow control layer in the session partner. Half duplex, flip-flop is the only LANDP-supported data flow control for both send and receive.

**data link control (DLC).** In SNA, the layer that consists of the link stations that schedule data transfer over a link between two nodes and perform error control for the link. Examples of data link control are SDLC for serial-by-bit link connection and data link control for the System/370™ channel. See also Systems Network Architecture (SNA). In SNA, a set of rules used by two nodes on a data link to accomplish an orderly exchange of information.

**data set.** The major unit of data storage and retrieval, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access. Sometimes called a file.

**data terminal equipment (DTE).** The part of a data station that serves as a data source, data sink, or both. (I) (A)

**database description (DBD).** In LANDP, the shared-file server descriptor. In IMS/VS, the collection of macro-parameter statements that describes an IMS/VS database. These statements describe the hierarchical structure, IMS/VS organization, device type, segment length, sequence fields, and alternate search fields. The statements are assembled to produce database description blocks.

**datagram.** The basic unit of information that is passed across the Internet. It consists of one or more data packets.

**DBCS.** Double-byte character set.

**DBD.** Database description.

**DBM.** Database manager.

**DCA.** Direct communication adapter.

**DCE.** Data circuit-terminating equipment. Distributed Computing Environment.

**DDE.** Dynamic data exchange.

**DDT.** Diagnostic and debugging tool.

**decipher.** To convert enciphered data in order to restore the original data. (T) In computer security, to convert ciphertext into plaintext by means of a cipher system. To convert enciphered data into clear data. Synonymous with decrypt. Contrast with encipher.

**decompression.** A function that expands data to the length that preceded data compression. Contrast with compression.

**decrypt.** In computer security, to decipher or decode. Synonymous with decipher. (T)

**default.** A value, attribute or option that is assumed when none is explicitly specified.

**delimiter.** A character used to show the beginning and end of a character string. (T) A character that groups or separates words or values in a line of

**deprecation.** An obsolete component that may be deleted from a future release of a product.

**DES.** Data Encryption Standard.

**development workstation.** A workstation which is part of a LANDP workgroup, and which is customized via a customization workstation.

**device driver.** In Advanced DOS, a file that contains the code needed to attach and use a device.

**DFC.** Data flow control.

**DIN.** Deutsches Institut für Normung.

**direct access.** The capability to obtain data from a storage device, or to enter data into a storage device, in a sequence independent from their relative position, by means of addresses indicating the physical position of the data. (T) Contract with sequential access.

**direct access storage device (DASD).** A device where access time is effectively independent of the location of the data.

**directory.** A table of identifiers and references to the corresponding items of data. (I) (A) A type of file containing the names and controlling information for other files or other directories. An index that is used by a control program to locate one or more blocks of data that are stored in separate areas of a data set in direct access storage. A listing of the files stored on a diskette.

**directory service (DS).** An application service element that translates the symbolic names used by application processes into the complete network addresses used in an OSI environment. (T)

**disk.** A round, flat data medium that is rotated to read or write data. (T) Loosely, a magnetic disk unit.

**disk operating system.** An operating system for computer systems that use disks and diskettes for auxiliary storage of programs and data.

**diskette.** A thin, flexible magnetic disk and a semi-rigid protective jacket, where the disk is permanently enclosed. Contrast with hard disk.

**Distributed Computing Environment (DCE).** The Open Software Foundation (OSF) specification (or a product derived from this specification) that assists in networking. DCE provides such functions as authentication, directory service (DS), and remote procedure call (RPC).

**distributed system.** A data processing system where processing, storage, and control functions, and also input and output operations, are distributed among remote locations.

**distribution diskette.** A diskette on which IBM sends programs and documentation to a customer.

**DLC.** Data link control.

**DLL.** Dynamic link library.

**DMA.** Direct memory access.

**domain.** The part of a computer network where the data processing resources are under common control. (T) In a database, all the possible values of an attribute or a data element. In SNA, a system services control point (SSCP) and the physical units (PUs), logical units (LUs), links, link stations, and all associated resources that the SSCP could control with activation requests and deactivation requests.

**DOS.** Disk Operating System.

**double-byte character set (DBCS).** A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets. Because each character requires 2 bytes, the typing, display, and printing of DBCS characters requires hardware and programs that support DBCS. Contrast with single-byte character set (SBCS).

**DS.** Directory service.

**DSR.** Data Set Ready.

**DTE.** Data terminal equipment. (A)

**DTE/DCE interface.** The physical interface and link access procedures between a data terminal equipment (DTE) and a data circuit-terminating equipment (DCE).

**dynamic data exchange (DDE).** The exchange of data between programs or between a program and a data-file object. Any change made to information in one program or session is applied to the identical data created by the other program.

**dynamic link library (DLL).** A file containing executable code and data bound to a program at load time or run time, rather than during linking. The code and data in a dynamic link library can be shared by several applications simultaneously.

# E

**EB.** End bracket.

**EBCDIC.** Extended binary-coded decimal interchange code.

**EGA.** Enhanced graphics adapter.

**EID.** End-of-message (EOM) identification.

**EMM.** Expanded memory manager.

**emulation.** The use of a data processing system to imitate another data processing system, so that the imitating system accepts the same data, executes the same programs, and achieves the same results as the imitated system. Emulation is usually achieved with hardware or firmware. (T)

**encipher.** To scramble data or to convert data to a secret code that masks the meaning of the data to any unauthorized recipient. Synonymous with encrypt. (T) In computer security, to convert plaintext into an unintelligible form by means of a

cipher system. Synonymous with cipher. Contrast with decipher. See also encode.

**enciphered data.** Data whose meaning is concealed from unauthorized users or observers. Synonymous with encode.

**encode.** To convert data by the use of a code in such a manner that reconversion to the original form is possible. (T) In computer security, to convert plaintext into an unintelligible form by means of a code system. See also plaintext.

**encrypt.** In computer security, to encode or encipher. Synonym for encipher. (T)

**end bracket (EB).** An SNA bracket protocol term used when the bind session specifies the end bracket call. If specified in the bind session, the personal computer may send both begin bracket and end bracket calls (not-response mode protocol). Contrast with begin bracket.

**end-of-message (EOM).** The character or sequence of characters that shows the end of a message or record.

**enhanced graphics adapter (EGA).** An adapter, such as the IBM Enhanced Graphics Adapter, that provides high-resolution graphics, allowing the use of a color display for text processing and also graphics applications.

**environment.** A named collection of logical and physical resources used to support the performance of a function.

**EOM.** End-of-message.

**erase.** To remove data from a data medium. Erasing is usually accomplished by overwriting the data or deleting the references. (T)

**error log.** A data set or file in a product or system where error information is stored for later access. A record of machine checks, device errors, and volume statistical data.

**error message.** An indication that an error has been detected. (A)

**ERRORLEVEL.** A parameter of the IF command used by batch files. It is used in testing for failure of recently loaded programs.

**event.** An occurrence or happening. An occurrence of significance to a task; for example, the completion of an asynchronous operation, such as an input/output operation. A data link control command and response passed between adjacent nodes that allows the two nodes to exchange identification and other information necessary for operation over the data link. In the NetView program, a record indicating irregularities of operation in physical elements of a network.

**exception.** An object that has caused some new condition, such as an error. In Java, throwing an error means passing that object to an interested party. A signal indicates what condition has occurred. Catching the condition means receiving the sent object. Handling this exception means dealing with the problem after receiving the object (though it might mean doing nothing, which is bad programming practice).

**exchange identification (XID).** The ID that is exchanged with the remote physical unit when an attachment is first established.

**EXE, exe.** An executable file with the file extension .EXE.

**extended ASCII.** A set of ASCII codes that uses the eighth (most significant) bit to define 127 additional codes. Standard ASCII uses 7 bits and defines 128 codes.

**extended binary-coded decimal interchange code (EBCDIC).** A coded character set of 256 8-bit characters.

**external logging replicator (XLR).** Shared-file mode of operation in which fault-tolerant data replication is achieved by logging database updates to an external server.

# F

**facility.** An operational capability, or the means for providing such a capability. (T) A service

provided by an operating system for a particular purpose; for example, the checkpoint/restart facility.

**FBSI.** Financial Branch Systems Integrator.

**FBSS (DOS).** IBM Financial Branch Systems Service (DOS). The predecessor to LANDP.

**FBSS/2.** Financial Branch Systems Service/2.

**FCB.** File control block.

**FIC.** First-in-chain.

**file.** A named set of records stored or processed as a unit. (T) A collection of information treated as a unit. A collection of data that is stored and retrieved by an assigned name.

**file control block (FCB).** A record that contains all of the information about a file, such as its structure, length, and name.

**file index table (FIT).** A table used by WorkSpace On-Demand to redirect file access requests from a client workstation's boot drive to the appropriate location on the boot server.

**file server.** A high-capacity disk storage device or a computer that each computer on a network can use to access and retrieve files that can be shared among the attached computers.

**file transfer.** In remote communications, the transfer of one or more files from one system to another over a communications link.

**first-in-chain (FIC).** A request unit (RU) whose request header (RH) begin chain indicator is on and whose RH end chain indicator is off.

**FIT.** file index table

**fixed disk.** Synonym for hard disk.

**flag.** A variable indicating that a certain condition holds. (T) Any of various types of indicators used for identification; for example, a word mark. (A) A character that signals the occurrence of some condition, such as the end of a word. (A)

**FMH.** Function management header.

**format identification (FID) field.** In SNA, a field in each transmission header (TH) that shows the format of the transmission header; that is, the presence or absence of certain fields.

**forward recovery.** The process of reconstructing a file from a particular point by restoring a saved version of the file and then applying changes to that file in the same order in which they were originally made.

**function.** In IBM LANDP for DOS, IBM LANDP for OS/2, IBM LANDP for Windows NT, IBM FBSS (DOS), IBM FBSS/2, IBM PC/Integrator, and IBM PC Integrator/2 a function is the specification of an activity to be performed by a server. In computer programming, synonym for procedure.

**function management header (FMH).** A special record or part of a record that contains control information for the data that follow. In SNA, one or more headers optionally present in the leading request units (RUs) of an RU chain that allow a half-session in an LU-LU session to: (a) select a destination as session partner and control way where end-user data it sends are handled at the destination, (b) change destination or characteristics of data during session, and (c) send between session partners status or user information about destination; for example, whether it is a program or device.

## G

**gateway.** In LANDP, the workstation that connects the LANDP workgroup to a host computer with the necessary LANDP software and the respective physical attachment. A functional unit that interconnects two computer networks with different network architectures. A gateway connects networks or systems of different architectures. A bridge interconnects networks or systems with the same or similar architectures. (T) A network that connects hosts. Contrast with router.

**generic alert.** A product-independent method of encoding alert data by means of both (a) code points indexing short units of stored text and (b) textual data.

# H

**hard disk.** A rigid magnetic disk such as the internal disks used in the system units of IBM personal computers and in external hard disk drives. Synonym for fixed disk. Contrast with diskette.

**HDLC.** High-level data link control.

**hexadecimal.** Describing a numbering system with base of sixteen; valid numbers use the digits 0 through 9 and characters A through F, where A represents 10 and F represents 15.

**high-level data link control (HDLC).** In data communication, the use of a specified series of bits to control data links under the International Standards for HDLC: ISO 3309 Frame Structure and ISO 4335 Elements of Procedures.

**host, host computer, host processor, or host system.** The primary or controlling computer in a multiple computer installation. A computer used to prepare programs for use on another computer or on another data processing system; for example, a computer used to compile, link edit, and test programs to be used on another system.

**hot-key.** The key combination used to change from one session to another on the workstation.

**Hypertext Transfer Protocol (HTTP).** The Internet protocol, based on TCP/IP, that is used to fetch hypertext objects from remote hosts.

# I

**I/O.** Input/output.

**IBM Operating System/2® (OS/2).** Pertaining to the IBM licensed program that can be used as the operating system for personal computers. The OS/2 licensed program can perform multiple tasks at the same time.

**ICV.** Initial chaining value.

**ID.** Identifier. Identification.

**identification.** In computer security, the process that allows a system to recognize an entity with personal, equipment, or organizational characteristics or codes.

**identifier.** One or more characters used to identify or name a data element or possibly to show certain properties of that data element. (A)

**IEEE.** Institute of Electrical and Electronics Engineers.

**IMS/VS.** Information Management System/Virtual Storage.

**indexed access.** Pertaining to the organization and accessing of the records of a storage structure through a separate index to the locations of the stored records. (A)

**indexed sequential access.** Pertaining to the organization and accessing of records through an index of the keys that are stored in arbitrarily partitioned sequential files. (A)

**initial chaining value (ICV).** An 8-byte pseudo-random number used to verify that both ends of a session with cryptography have the same session cryptography key. The initial chaining value is also used as input to Data Encryption Standard (DES) algorithm to encipher or decipher data in a session with cryptography.

**initial program load (IPL).** The initialization procedure that causes an operating system to begin operation. The process by which a configuration image is loaded into storage at the beginning of a work day or after a system malfunction. The process of loading system programs and preparing a system to run jobs.

**initialization.** The operations required for setting a device to a starting state, before the use of a data medium, or before implementation of a process. (T) Preparation of a system, device, or program for operation.

**initiate self.** An SNA command issued by the LANDP SNA server to initiate a host application. The SNA command is issued in response to the receipt of an Open command from the personal computer.

**INITSELF.** Initiate self.

**input/output (I/O).** Describing a device whose parts can perform an input process and an output process at the same time. (I) Describing a functional unit or channel involved in an input process, output process, or both, concurrently or not, and to the data involved in such a process.

**Instruction Pointer (IP).** In System 38, a pointer that provides addressability for a machine interface instruction in a program.

**interface.** A shared boundary between two functional units, defined by functional characteristics, signal characteristics, or other characteristics, as appropriate. The concept includes the specification of the connection of two devices having different functions. (T)

**International Organization for Standardization (ISO).** An organization of national standards bodies from various countries established to promote development of standards to simplify international exchange of goods and services, and develop cooperation in intellectual, scientific, technological, and economic activity.

**Internet Protocol (IP).** A protocol used to route data from its source to its destination in an Internet environment.

**interoperability.** The capability to communicate, execute programs, or transfer data among various functional units in a way that requires the user to have little or no knowledge of the unique characteristics of those units. (T) In SAA usage, the ability to link SAA and non-SAA environments and use the combination for distributed processing.

**IP.** Instruction Pointer. Internet Protocol.

**IPL.** Initial program load.

**ISAM.** Indexed sequential access method.

**ISO.** International Organization for Standardization.

# J

**Jar file format.** Java Archive, a platform-independent file format that aggregates many files into one. Multiple Java applets and their requisite components (.class files, images, sounds, and other resource files) can be bundled in a JAR file and subsequently downloaded to a browser in a single HTTP transaction.

**Java.** An object-oriented programming language for portable, interpretive code that supports interaction among remote objects. Java was specified and developed by Sun Microsystems, Incorporated. The Java environment consists of the JavaOS, the Virtual Machines for various platforms, the object-oriented Java programming language, and several class libraries.

**Java Development Kit (JDK).** A set of Java technologies made available to licensed developers by Sun Microsystems. Each release of JDK consists of the Java compiler, Java virtual machine, Java class libraries, Java applet viewer, Java debugger, and other tools.

**JavaDoc.** Sun Microsystems tool for generating HTML documentation of classes by extracting comments from the Java source code files.

**Java eXtensions for Financial Services (J/XFS).** J/XFS is a proposed standard, developed jointly by IBM, Siemens Nixdorf, NCR, DeLaRue and Sun Microsystems, for a standardised interface to all common financial devices that can be used by applications and applets within the Java programming language. The J/XFS standard defines APIs for accessing financial devices, and a control mechanism that enables any number of applications to access these devices. The following LANDP servers are supported through the J/XFS API:

- Financial Printer Server (4722 document printer, 9068 multipurpose passbook printer)
- Magnetic Stripe Reader/Encoder (MSR/E) Server (4717, 4777)
- Personal Identification Number (PIN) Pad Server (4778 including MSD support)

**Java Remote Method Invocation (RMI).** Method invocation between peers, or between client and server, when applications at both ends of the invocation are written in Java. Java RMI is included in JDK 1.1, or later.

**Java Runtime Environment (JRE).** A subset of the Java Development Kit (JDK) that contains the core executables and files that constitute the standard Java platform. The JRE includes the Java Virtual Machine, core classes and supporting files.

**Java Virtual Machine.** A software implementation of a central processing unit (CPU) that runs compiled Java code (applets and applications).

**journal.** A chronological record of changes made in a set of data; the record may be used to reconstruct a previous version of the set. (T) A special-purpose data set that provides an audit trail of operator and system actions, or as a means of recovering superseded data.

**JRE.** Java Runtime Environment.

**JVM.** Java Virtual Machine.

**J/XFS.** Java/eXtensions for Financial Services.

# K

**KB.** Kilobyte; 1024 bytes.

**key.** An identifier within a set of data elements. (T) One or more characters used to identify the record and establish the order of the record within an indexed file.

**keystroke.** Actuation of a key on a keyboard to perform or release a machine function. (T)

**keyword.** A name or symbol that identifies a parameter or an ordered set of parameters.

# L

**LAN.** Local area network.

**LAN configuration.** The process by which the details about the structure of the LAN for a particular user are provided to the LANDP family programs. This includes details about the workstations forming the LAN, the services provided by each workstation, and the workstations that receive the services.

**LAN trace.** A LANDP family trace facility that informs about the LANDP-related LAN and displays the status of the local area network.

**LAN Distributed Platform.** The former name for the LANDP family of products.

**last-in-chain (LIC).** A request unit (RU) whose request header (RH) end chain indicator is on and whose RH begin chain indicator is off.

**LDA.** Logical device address.

**LED.** Light-emitting diode.

**LIC.** Last-in-chain.

**light-emitting diode (LED).** A semiconductor chip that gives off visible or infrared light when operated.

**link connection.** In SNA, the physical equipment providing two-way communication between one link station and one or more other link stations; for example, a telecommunication line and data circuit-terminating equipment (DCE).

**LIP.** LAN Internet Protocol.

**LLAP.** Logical link access path.

**loader.** A routine, commonly a computer program, that reads data into main storage. (A)

**local area network (LAN).** A computer network located on a user's premises within a limited geographical area. Communication within a local area network is not subject to external regulations; however, communication across the LAN boundary may be subject to some form of regulation. (T)

**local host.** In the Internet, the computer to which a user's terminal is directly connected without using the Internet.

**logging.** The recording of data about specific events.

**logical device address (LDA).** A number used to represent a terminal or terminal component within a workstation. See also physical device address.

**logical link access path (LLAP).** In a multi-system environment, the path between any two systems. One or more logical link paths must be defined for each logical link.

**logical unit (LU).** In SNA, a port through which an end user accesses the SNA network to communicate with another end user and through which the end user accesses the functions provided by the system services control points (SSCPs). An LU can support at least two sessions, one with an SSCP and one with another LU, and may be capable of supporting many sessions with other logical units. A type of network addressable unit that allows end users to communicate with each other and gain access to network resources.

**longitudinal parity check.** A parity check of a row of binary digits that are members of a set forming a matrix; for example, a parity check of the bits of a track in a block on a magnetic stripe. (T)

**longitudinal redundancy check (LRC).** Synonym for longitudinal parity check.

**LRC.** Longitudinal redundancy check.

**LU.** Logical unit.

**LU—LU session type 0.** In SNA, a type of session between two LU—LU half-sessions using SNA-defined protocols for transmission control and data flow control, but using end-user or product-defined protocols to augment or replace FMD services protocols.

**LU—LU session type 1.** In SNA, a type of session between an application program and single- or multiple-device data processing terminals in an interactive, batch data transfer, or distributed processing environment.

**LU—LU session type 2.** In SNA, a type of session between an application program and a single display terminal in an interactive environment, using the SNA 3270 data stream.

**LUSTAT.** An SNA command used to send logical unit status information.

# M

**MAC.** Message authentication code.

**mapper.** A device, such as a piece of code, which performs a mapping function.

**mapping.** A list, usually in a profile, that establishes a correspondence between items in two groups; for example, a keyboard mapping can establish what character is displayed when a certain key is pressed. In a database, the establishing of correspondences between a given logical structure and a given physical structure. (T)

**MB.** Megabyte; 1 048 576 bytes.

**memory.** All of the addressable storage space in a processing unit and other internal storages that is used to execute instructions. (T)

**message.** An assembly of characters and sometimes control codes that is transferred as an entity from an originator to one or more recipients. A message consists of two parts: envelope and content. (T) A communication sent from a person or program to another person or program. A unit of data sent over a telecommunication line. One or more message segments transmitted among terminals, application programs, and systems. In SAA Common User Access architecture, information not requested by a user but displayed by an application in response to an unexpected event, or when something undesirable could occur.

**message authentication code (MAC).** In computer security, a value, part of, or accompanying a message, used to determine that the contents, origin, author, or other attributes of all or part of the message are as they appear to be. In cryptography: (a) a number or value derived by processing data with an authentication

algorithm, (b) the cryptographic result of block cipher operations on text or data using a cipher block chain (CBC) mode of operation, (c) a digital signature code.

**method.** A fragment of Java code within a class that can be invoked and passed a set of parameters to perform a specific task.

**MIC.** Middle-in-chain.

**MICR.** Magnetic ink character recognition.

**microcode.** One or more microinstructions. A code, representing the instructions of an instruction set, that is done in a part of storage that is not program-addressable. To design, write, and also to test one or more microinstructions.

**middle-in-chain (MIC).** A request unit (RU) whose request header (RH) begin chain indicator and RH end chain indicator are both off.

**mnemonic.** A symbol chosen to help the user remember the significance of the symbol.

**mode.** A method of operation.

**mode switching.** Operator switching between a concurrently running personal computer application and 3270 emulation or other internal application.

**MSR, MSR/E.** Magnetic stripe reader; Magnetic stripe reader/encoder.

**multi-tasking.** A mode of operation that provides for concurrent performance, or interleaved execution of two or more tasks. (I) (A)

# N

**name server.** The server that stores resource records about hosts. In the AIX operating system, a host that provides name resolution for a network. Name servers translate symbolic names assigned to networks and hosts into the Internet addresses used by machines. In TCP/IP, synonym for domain name server.

**NAU.** Network addressable unit.

**NCP.** Network Control Program.

**NDIS.** Network Driver Interface Specification

**NetBIOS.** Network Basic Input/Output System. A standard interface to networks, IBM personal computers (PCs), and compatible PCs, that is used on LANs to provide message, print-server, and file-server functions. Application programs that use NetBIOS do not need to handle the details of LAN data link control (DLC) protocols. See also BIOS.

**NetView program.** An IBM licensed program used to monitor and manage a network and to diagnose network problems.

**network.** An arrangement of nodes and connecting branches. (T) A configuration of data processing devices and software connected for information interchange.

**network addressable unit (NAU).** In SNA, a logical unit, a physical unit, or a system services control point. The NAU is the origin or the destination of information transmitted by the path control network. See also logical unit, physical unit, system services control point (SSCP).

**Network Control Program (NCP).** An IBM licensed program that provides communication controller support for single-domain, multiple-domain, and interconnected network capability. See also Advanced Communications Function (ACF).

**network management vector transport (NMVT).** A management services request/response unit (RU) that flows over an active session between physical unit management services and control point management services (SSCP-PU session).

**network resource.** In ACF/VTAM®, a network component such as a local network control program, an SDLC data link, or a peripheral node.

**network services procedure error (NSPE).** A request unit that is sent by a system services control point (SSCP) to a logical unit (LU) when a procedure requested by that LU has failed.

**NLS.** National language support.

**NMVT.** Network management vector transport.

**node.** In a network, a point at which one or more functional units connect channels or data circuits. (I) In network topology, the point at an end of a branch. (T)

**NPSI.** X.25 NCP Packet Switching Interface.

**NSPE.** Network services procedure error.

## O

**object.** The principal building block of object-oriented programs. Objects are software programming modules. Each object is a programming unit consisting of related data and methods.

**object-oriented programming (OOP).** A programming approach based on the concepts of data abstraction and inheritance. Unlike procedural programming techniques, object-oriented programming concentrates on the data objects that constitute the problem and how they are manipulated, not on how something is accomplished.

**ODBC.** Open Database Connectivity is a standardized set of API function calls that can be used to access data stored in both relational and non-relational DBMSs.

**OIA.** Operator information area.

**OIC.** Only-in-chain.

**only-in-chain (OIC).** A request unit (RU) for which the request header (RH) begin chain indicator and RH end chain indicator are both on.

**OOP.** object-oriented programming

**open.** The function that connects a file to a program for processing. Contrast with close.

**open system.** A system with specified standards, and that therefore can be readily connected to other systems that comply with the same standards.

**operating system.** Software that controls the execution of programs and that may provide services such as resource allocation, scheduling, input/output control, and data management. Although operating systems are predominantly software, partial hardware implementations are possible. (T)

**operator information area (OIA).** In the 3270 Information Display System, the area near the bottom of the display area where terminal or system status information is displayed.

**option.** A specification in a statement that may be used to influence the processing of the statement.

**OS/2 operating system.** IBM Operating System/2.

## P

**pacing.** A technique by which a receiving station controls the rate of transmission of a sending station to prevent overrun.

**package.** A program element that contains classes and interfaces.

**packet.** A sequence of binary digits, including data and control signals, that is transmitted and switched as a composite entity.

**panel.** A formatted display of information that appears on a display screen.

**parallel port.** On a personal computer system, a port used to attach devices such as dot matrix printers and input/output units; it transmits data one byte at a time. See also serial port.

**parameter.** A variable that is given a constant value for a specified application and that may denote the application. (I) (A) An item in a menu for which the user specifies a value or for which the system provides a value when the menu is interpreted. Data passed between programs or procedures.

**Pascal.** A high-level, general purpose programming language, related to ALGOL.

Programs written in Pascal are block structured, consisting of independent routines. They can run on different computers with little or no modification.

**path.** In a personal computer system, the logical relationship between directories.

**PBM.** Personal banking machine.

**PC.** Personal computer.

**PC-ID.** Workstation identifier.

**PCB.** Program control block.

**PC/TCP.** FTP Software's implementation of TCP/IP for systems running DOS and Windows. Now called PC/TCP Network Software version 5.0 and available from NetManage Inc..

**PDA.** Physical device address.

**PDP.** Problem determination procedure.

**personal computer system.** IBM Personal System/2 and also the various IBM Personal Computer system units, unless otherwise described.

**Personal Identification Number (PIN) pad.** A pad with twelve keys in a specific arrangement that display alphabetic and numeric characters that may be entered onto a financial transaction terminal. (T) (A)

**physical device address (PDA).** An address or set of addresses that identifies a particular device.

**physical unit (PU).** In SNA, the component that manages and monitors the resources, such as attached links and adjacent link stations, associated with a node, as requested by an SSCP via an SSCP-PU session. An SSCP starts a session with the physical unit to indirectly manage, through the PU, resources of the node such as attached links. This term applies to type 2.0, type 4, and type 5 nodes only.

**PIN.** Personal identification number.

**plaintext.** Nonencrypted data. Synonymous with cleartext. Synonym for clear data.

**PLU.** Primary logical unit.

**PM.** Presentation Manager® (in OS/2).

**pointing device port.** The IBM PS/2 port that allows attachment of various devices including pointing devices.

**port.** An access point for data entry or exit. A connector on a device to which cables for other devices such as display stations and printers are attached. A specific communications end point within a host. A port is identified by a port number.

**Post Telephone and Telegraph Administration (PTT).** An organization, usually a government department, that provides communication common carrier services in countries other than the USA and Canada. Examples of PTTs are the Bundespost in Germany, and the Nippon Telephone and Telegraph Public Corporation in Japan.

**PPC.** Program to program communications.

**Presentation Manager.** A component of OS/2 that provides a complete graphics-based user interface, with pull-down windows, action bars, and layered menus.

**primary logical unit (PLU).** In SNA, the logical unit (LU) that contains the primary half-session for a particular LU—LU session. Contrast with secondary logical unit (SLU). See also logical unit (LU).

**problem determination procedure (PDP).** A prescribed sequence of steps taken to identify the source of a problem.

**process.** A unique, finite course of events defined by its purpose or by its effect, achieved under defined conditions. Any operation or combination of operations on data. A function being performed or waiting to be performed. A program in operation.

**processor.** In a computer, a functional unit that interprets and executes instructions. A processor consists of at least an instruction control unit and an arithmetic and logic unit. (T) The functional unit that interprets and processes instructions.

**profile.** In computer security, a description of the characteristics of an entity to which access is controlled. Data that describes the significant characteristics of a user, a group of users, or one or more computer resources.

**program.** A sequence of instructions suitable for processing by a computer. Processing may include the use of an assembler, a compiler, an interpreter, or a translator to prepare the program for execution, and also to execute it. (I)

**program control block (PCB).** LANDP family shared-file server pointer related to a specific DBD.

**Program temporary fix (PTF).** A temporary solution or bypass of a problem diagnosed by IBM as resulting from a defect in a current unaltered release of the program.

**protocol.** In SNA, the meanings of and the sequencing rules for requests and responses used for managing the network, transferring data, and synchronizing the states of network components.

**PS/2.** Personal System/2.

**PTF.** Program temporary fix.

**PTT.** Post Telephone and Telegraph Administration.

**PU.** Physical unit.

# Q

**QLLC.** Qualified logical link control.

**qualified logical link control (QLLC).** An X.25 protocol that allows the transfer of data link control information between two adjoining systems network architecture (SNA) nodes that are connected through an X.25 packet-switching data network. The QLLC provides the qualifier “Q” bit in X.25 data packets to identify packets that carry logical link protocol information.

**query.** A request for information from a file relying on specific conditions. In the AS/400 system, the query management object that is used to define queries against relational data.

**quiescing.** The process of bringing a device or a system to a stop by rejection of new requests for work. (A)

# R

**RAM.** Random access memory. (A)

**random access memory (RAM).** A storage device where data can be written and read.

**RC.** Return code.

**RCMS.** Remote change management services.

**RDBMS.** Relational database management system. A generic name for any relational database system such as DB2.

**re-synchronization.** Restarting the transmission of a function at the point where it was interrupted.

**read-only memory (ROM).** A storage device where data, under normal conditions, can only be read. (T) See also read-only storage (ROS).

**read-only storage (ROS).** A storage device whose contents cannot be modified, except by a particular user, or when operating under particular conditions. See also read-only memory (ROM).

**record.** In programming languages, an aggregate that consists of data objects, possibly with different attributes, that usually have identifiers attached to them. In some programming languages, records are called structures. (I) A set of data treated as a unit. (T) A set of one or more related data items grouped for processing.

**remote attachment.** A method of connecting two devices over a telecommunication line.

**remote initial program load (remote IPL).** A feature that permits a computer to receive its initial program from another computer, rather than from its own internal disk or diskette storage.

**remote method invocation.** A specific instance of the more general term RPC (remote procedure call). Remote method invocation (RMI) allows objects to be distributed over a network, that is, a Java program running on one computer can call

the methods of an object running on another computer. RMI and java.net are the only 100% pure Java APIs for controlling Java objects in remote systems.

**remote procedure call (RPC).** A facility that a client uses to request the execution of a procedure call from a server. This facility includes a library of procedures and an external data representation.

**REMS.** Reader/encoder magnetic stripe.

**request/response header (RH).** In systems network architecture (SNA), control information preceding a request/response unit (RU) that specifies the type of RU and contains control information associated with the RU.

**request/response unit (RU).** In systems network architecture (SNA), a generic term for a request unit or a response unit.

**resource.** Any of the data processing system elements needed to perform required operations, including storage, input/output units, one or more processing units, data, files, and programs. (T) See also network resource.

**retry.** To resend data a prescribed number of times or until the data is received correctly.

**return code (RC).** A code used to influence the execution of succeeding instructions. (A) A value returned to a program to indicate the results of an operation requested by that program.

**RH.** Request/response header.

**roll back.** To remove changes that were made to database files under commitment control since the last commitment boundary.

**RMI.** Remote Method Invocation.

**rollback.** A programmed return to a prior checkpoint. (A) The process of restoring data changed by an application program or user to the state of its last commitment boundary. In SQL, the process of restoring data changed by an application program or user to the state of its last commit point.

**ROM.** Read-only memory. (A)

**ROS.** Read-only storage.

**router.** A computer that determines the path of network traffic flow. The path selection is made from several paths based on information obtained from specific protocols, algorithms that attempt to identify the shortest or best path, and other criteria such as metrics or protocol-specific destination addresses. An attaching device that connects two LAN segments, which use similar or different architectures, at the reference model network layer. Contrast with bridge, gateway. In OSI terminology, a function that determines a path by which an entity can be reached.

**RPC.** Remote procedure call.

**RTR.** Ready to Receive.

**RU.** Request/response unit.

# S

**SAM.** Service availability manager.

**SAP.** Service access point.

**SBC.** Server Based Computing.

**SBCS.** Single-byte character set.

**scan code.** A code generated by a keyboard.

**SCS.** Systems network architecture character string.

**SDLC.** Synchronous data link control.

**secondary logical unit (SLU).** In systems network architecture (SNA), the logical unit (LU) that contains the secondary half-session for a particular LU-LU session. Contrast with primary logical unit (PLU). See also logical unit (LU).

**SEQ.** Sequential file.

**sequential access.** The capability to enter data into a storage device or a data medium in the same sequence as the data is ordered, or to obtain data in the same order as it has been

entered. (T) An access method in which records are read from, written to, or removed from a file based on the logical order of the records in the file. Contrast with direct access.

**serial port.** On personal computer systems, a port used to attach devices such as display devices, letter-quality printers, modems, plotters, and pointing devices such as light pens and mice; it transmits data one bit at a time. See also parallel port.

**serialization.** Turning an object into a stream and back again.

**server.** A functional unit that provides shared services to workstations over a network; for example, a file server, a print server, a mail server. (T) In LANDP, a functional area that provides functions to LANDP workstations in a LANDP workgroup. The computer that hosts the Web page that contains an applet. The .class files that make up the applet, and the HTML files that reference the applet reside on the server. When someone on the Internet connects to a web page that contains an applet, the server delivers the .class files over the Internet to the client that made the request. The server is also known as the originating host. See also client, client workstation, and user. In LANDP, a function provided by a server.

**server based computing (SBC).** This is a generic term that is used to describe network computer systems such as WorkSpace on Demand and Windows Terminal Server.

**service access point (SAP).** A logical point made available by a token-ring adapter where information can be received and transmitted.

**service availability manager (SAM).** Facility used by the shared-file server to provide fault-tolerant data access in an XLR environment.

**servlet.** Server-side program that executes on and adds function to a Web server. Java servlets allow for the creation of complicated, high-performance, cross-platform Web applications. They are highly extensible and flexible, making it easy to expand from client or single-server applications to multi-tier applications.

**session.** In systems network architecture (SNA), a logical connection between two network addressable units (NAU) that can be started, tailored to provide various protocols, and deactivated, as requested. The time during which programs or devices can communicate with each other.

**single-byte character set (SBCS).** A character set in which each character is represented by a one-byte code. Contrast with double-byte character set (DBCS).

**SLU.** Secondary logical unit.

**SNA.** Systems network architecture.

**SNUF.** Systems network architecture up-line facility.

**socket.** An endpoint for communication between processes or applications. A pair consisting of TCP port and IP address.

**SOM.** Start-of-message code.

**SPC, spc.** Specification file.

**specification file (SPC, spc).** In LANDP, a file with the file extension .SPC. This file can be edited. It contains information for customization purposes.

**SQL.** Structured query language.

**SSCP.** System services control point.

**start-of-message code (SOM).** A character or group of characters transmitted by the polled terminal and indicating to other stations on the line that what follows are addresses of stations to receive the answering message.

**storage.** A functional unit into which data can be placed, where it can be retained, and from which it can be retrieved. (T)

**stream.** A continuous sequence of data elements being transmitted, or intended for transmission, in character or binary-digit form, using a defined format.

**structured query language (SQL).** An established set of statements used to manage information stored in a database. By using these statements, users can add, delete, or update information in a table, request information through a query, and display the results in a report.

**subdirectory.** A directory contained within another directory in a file system hierarchy.

**synchronous.** About two or more processes that depend on the occurrence of a specific event such as common signal timing. Occurring with a regular or predictable time relationship. See also asynchronous.

**synchronous data link control (SDLC).** A discipline conforming to subsets of the Advanced Data Communication Control Procedures (ADCCP) of the American National Standards Institute (ANSI) and High-level Data Link Control (HDLC) of the International Organization for Standardization, for managing synchronous, code-transparent, serial-by-bit information transfer over a link connection. Transmission exchanges may be duplex or half-duplex over switched or not-switched links. The configuration of the link connection may be point-to-point, multi-point, or loop. (I)

**system diskette.** The diskette, either real or virtual, that contains your control program. In personal computer systems, the diskette on which you have the operating system.

**system distribution manager.** A system that contains the files and programs required for product installation, and initiates or manages the installation process.

**system services control point (SSCP).** In systems network architecture (SNA), the focal point within an SNA network for managing the configuration, coordinating network operator and problem determination requests, and providing directory support and other session services for end users of the network.

**systems network architecture (SNA).** The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through and controlling the configuration and operation of networks.

**systems network architecture character string (SCS).** In systems network architecture (SNA), a character string composed of EBCDIC controls, optionally intermixed with end-user data, that is carried within a request/response unit (RU).

**systems network architecture network (SNA network).** In systems network architecture (SNA), the part of an application program network that conforms to the formats and protocols of SNA. It allows reliable transfer of data among end users and provides protocols for controlling the resources of various network configurations. The SNA network consists of network addressable units (NAU), boundary function components, and the path control network.

**systems network architecture up-line facility (SNUF).** The communications support that allows an AS/400 system to communicate with CICS/VS and IMS/VS application programs on a host computer.

# T

**takeover.** In an XLR environment, the process by which a backup server assumes the role of the (failed) active. This involves backing out incomplete transactions, rebuilding indexes, and informing SAM of the new active workstation.

**TCP/IP.** Transmission Control Protocol/Internet Protocol.

**terminal status line.** Synonym for operator information area (OIA).

**TH.** Transmission header.

**thin client.** A client workstation that loads its operating system environment and applications

across a network from a server. The degree of local processing power in a thin client can vary considerably depending on the implementation of the thin client concept.

The term thin client usually refers to a system that runs on a resource-constrained machine or that runs on a small operating system. This clients do not require local system administration, and they execute Java applications delivered over the network.

**Time Sharing Option (TSO).** An operating system option; for the System/370 system, the option provides interactive time sharing from remote terminals.

**token-ring network.** A ring network that allows unidirectional data transmission between data stations by a token passing procedure, so that the transmitted data returns to the transmitting station. (T) A network that uses a ring topology, where tokens are passed in a circuit from node to node. A node that is ready to send can capture the token and insert data for transmission.

**trace.** A record of the execution of a computer program. It exhibits the sequences in which the instructions were executed. (A) The process of recording the sequence in which the statements in a program are executed and, optionally, the values of the program variables used in the statements. To record a series of events as they occur. For data links, a record of the frames and bytes transmitted or received.

**trace file.** A file that contains a record of events that occur in a system.

**trace function.** A function used for problem determination.

**trace log.** A file in which trace events are recorded.

**trace program.** A computer program that performs a check on another computer program by exhibiting the sequence in which the instructions are executed and, usually, the results of executing the instructions. (I) (A)

**trace routine.** A routine that provides an historical record of specified events in the execution of a computer program. (A)

**transaction.** An exchange between a workstation and another device that accomplishes a particular action or result.

**translation.** Conversion of a code or codes to another code or codes according to a set of specifications.

**transmission.** The sending of data from one place for reception elsewhere. (A)

**Notes:**

1. Transmission implies only the sending of data; the data may or may not be received.
2. The term transmit is used to describe the sending of data in telecommunication operations. The terms move and transfer are used to describe movement of data in data processing operations.

**transmission control (TC) layer.** The layer within a half-session or session connector that synchronizes and paces session-level data traffic, checks session sequence numbers of requests, and enciphers and deciphers end-user data.

**Transmission Control Protocol (TCP).** A communications protocol used in the Internet and in any network that follows the US Department of Defense standards for inter-network protocol. TCP provides a reliable host-to-host protocol between hosts in packet-switched communications networks and in interconnected systems of such networks. It assumes that the Internet protocol is the underlying protocol.

**Transmission Control Protocol/Internet Protocol (TCP/IP).** A set of communication protocols that support peer-to-peer connectivity functions for both local and wide area networks.

**transmission header (TH).** In systems network architecture (SNA), control information, optionally followed by a basic information unit (BIU) or a BIU segment, that is created and used by path control to route message units and to control their flow within the network.

**transmission services (TS) profile.** In systems network architecture (SNA), a specification in a session activation request (and, optionally in the responses) of transmission control (TC) protocols, such as session-level pacing and the usage of session-level requests, to be supported by a particular session. Each defined TS profile is identified by a number.

**trap.** An unprogrammed conditional jump to a specified address that is automatically activated by hardware. A recording is made of the location from which the jump occurred.

**TRDLC.** Token-ring data link control.

**TS.** Transmission services.

**TSO.** Time Sharing Option.

# U

**UDP.** User Datagram Protocol.

**UNBIND.** In systems network architecture (SNA), a request to deactivate a session between two logical units (LU). Contrast with BIND.

**user.** A function that uses the services provided by a server. A host can be a user and a server at the same time. Any person or any thing that may issue or receive commands and messages to or from the information processing system. (T) Any person who requires the services of a computing system. See also client, client/server, client workstation, and server.

**User Datagram Protocol (UDP).** In TCP/IP, a packet-level protocol built directly on the Internet protocol layer. UDP is used for application-to-application programs between TCP/IP host systems.

**user profile.** In computer security, a description of a user that includes such information as user identification (ID), user name, password, access authority, and other attributes obtained at log-on.

**user-written server.** In LANDP, a server not supplied with a LANDP program, but developed by the customer.

**utility program.** A computer program which supports computer processes; for example, a sort program. (T) A program designed to perform an everyday task such as copying data from one storage device to another. (A)

# V

**validation.** The checking of data for correctness, or compliance with applicable standards, rules, and conventions. (A)

**VDM.** Virtual DOS machine.

**vector.** A set of `keyword=parameter` statements that define configuration items. These items can correspond to both model and real configurations.

**verify.** To determine whether a transcription of data or other operation has been accomplished accurately. (A)

**VFS.** Virtual file system.

**virtual DOS machine (VDM).** A functional simulation of a machine running under DOS.

**virtual file system (VFS).** A remote file system that has been mounted so that it is accessible to the local user.

**virtual machine (VM).** A virtual data processing system that seems to be at the exclusive disposal of a particular user, but whose functions are accomplished by sharing the resources of a real data processing system. (T)

**Virtual Telecommunications Access Method (VTAM).** A set of programs that maintain control of the communication between terminals and application programs running under Disk Operating System/Virtual Storage (DOS/VS), OS/VS1, and OS/VS2 operating systems.

**VisualGen®.** A high-level object-oriented programming language.

**VM/CMS.** Virtual machine/conversational monitor system.

**VTAM.** Virtual Telecommunications Access Method.

## W

**WAN.** Wide area network.

**wide area network (WAN).** A network that provides communication services to a geographical area larger than that served by a local area network.

**WebSphere™.** A comprehensive solution to build, deploy, and manage e-business Web sites. WebSphere is the cornerstone of IBM's overall Web strategy The Websphere product line provides companies with an open, standards-based, Web server deployment platform, together with Web site development and management and management tools to help accelerate the process of moving to e-business.

**window.** A division of a screen where one of several programs being run concurrently can display information.

**workgroup.** In LANDP, the logical connection of LANDP for DOS, LANDP for OS/2, LANDP for Windows NT, and LANDP for AIX workstations through the LANDP client/server mechanism, which is available with each LANDP program.

**Workspace On-Demand.** A set of management utilities that enables OS/2 Warp Server to remotely load a thin client operating system, known as Workspace On-Demand client, into a client workstation across a LAN. The client workstation component of Workspace On-Demand, which is loaded into a client workstation from a server machine running OS/2 Warp Server and Workspace On-Demand Server.

**Workspace On-Demand Server.** A server, running OS/2 Warp Server and Workspace On-Demand, that is used to boot client workstations.

**workstation.** A functional unit at which a user works. In LANDP, personal computer system in a local area network (LAN).

**wrapper.** A language binding.

## X

**X.25.** A CCITT recommendation that defines the physical level (physical layer), link level (data link layer), and packet level (network layer) of the open systems inter-connection (OSI) reference model. An X.25 network is an interface between data terminal equipment (DTE) and data circuit-terminating equipment (DCE) operating in the packet mode, and connected to public data networks by dedicated circuits. X.25 networks use the connection-mode network service.

**X.25 NCP Packet Switching Interface.** An IBM-licensed program that allows systems network architecture (SNA) users to communicate over packet switched data networks that have interfaces complying with Recommendation X.25 (Geneva 1980) of the International Telegraph and Telephone Consultative Committee (CCITT). It allows SNA programs to communicate with SNA equipment or with non-SNA equipment over such networks.

**XID.** Exchange identification.

**XLR.** External logging replicator.

**XOR.** Logical operation exclusive-or.

## Numerics

**4700 Processor.** IBM Finance Communication System 4701 Controller Model 3 and IBM 4702 Branch Automation Processor, unless otherwise described.

# Bibliography

This bibliography includes publications cited in this book and other publications on related topics. Where a shortened title is used in the text, the short title is listed after the full title.

This bibliography also includes books which might now be obsolete.

## IBM LANDP Family

*IBM LANDP Family: Introduction and Planning*, GC34-6043.
**Short title:** *LANDP Introduction and Planning*.

*IBM LANDP Family: Installation and Customization*, GC34-6044.
**Short title:** *LANDP Installation and Customization*.

*IBM LANDP Family:Programming Guide*, SC34-6048.
**Short title:** *LANDP Programming Guide*.

*IBM LANDP Family: Programming Reference*, SC34-6045.
**Short title:** *LANDP Programming Reference*.

*IBM LANDP Family: Servers and System Management*, SC34-6046.
**Short title:** *LANDP Servers and System Management*.

*IBM LANDP Family: Problem Determination*, GC34-6047.
**Short title:** *LANDP Problem Determination*.

## IBM Financial Branch System Services Licensed Programs

*IBM FBSS Licensed Programs Family General Information*. GC19-5172.

*IBM FBSS Licensed Programs Family Installation and Customization*. SC19-5173.

*IBM FBSS Licensed Programs Family Program Description*. SC19-5176.

*IBM FBSS Licensed Programs Family Version 2 Programmer's Reference*. GA19-5450.

*IBM FBSS Licensed Programs Family Version 2 Application Programming*. SC19-5174.

## IBM Financial Branch System Integrator Licensed Programs

*Financial Branch System Integrator and Financial Branch System Integrator/2 General Information*. GC19-5187.

*Financial Branch System Integrator Programmer's Reference Manual*. GA19-5452.

*Financial Branch System Integrator/2 Programmer's Reference Manual*. SC19-5188.

## IBM Transaction Security System

*IBM Transaction Security System: General Information Manual and Planning Guide*. GA34-2137.

*IBM Transaction Security System: Programming Guide and Reference*. SC31-2934.

## Banking Self-Service

*IBM 4721 Self-Service Document Printer Programmer's Reference*. GA19-5342.

*IBM 4731/38/39 Personal Banking Machines P-Models Software Customization and Programming Reference*. GA19-5462.

*IBM 4733 Teller Assist Unit Programmer's Reference*. GA19-5425.

*IBM 4737 Self-Service Transaction Station Programmer's Reference*. GA19-5408.

*IBM Financial Application Development Toolkit Version 2 Program Description and Operation*. SB11-8461.

## IBM workstations

*PC DOS 7 Technical Update*. GG24-4459.

*PC DOS 7 User Guide*. S83G-9260.

*PC DOS 7 Command Reference*. S83G-9309.

1. This information is available in multiple languages. Contact your IBM representative for ordering information.

© Copyright IBM Corp. 1992, 2002

*PC DOS 7 Keyboard and Code Pages*. S83G-9310.

*IBM TCP/IP Version 2.1.1 for DOS: Installation and Administration*. SC31-7047.

*IBM TCP/IP Version 2.1.1 for DOS: User's Guide*. SC31-7045.

*IBM TCP/IP Version 2.1.1 for DOS: Programmer's Reference*. SC31-7046.

*OS/2 Warp Version 4 Up and Running!*. S84H-3098.

*OS/2 Warp Server for e-Business*. SG24-5393.

*OS/2 Warp, PM Programming Reference Vol I*. G25H-7190.

*OS/2 Warp PM Programming Reference Vol II*. G25H-7191.

*OS/2 2.0 Application Design Guide*. S10G-6260.

*OS/2 2.0 Virtual Device Driver Reference*. S10G-6310.

*DB2/2 Guide*. S62G-3663.

*OS/2 LAN Server Network Administration Reference Volume 1: Planning, Installation and Configuration*. S10H-9680.

*OS/2 LAN Server Network Administrator Reference Volume 2: Performance Tuning*. S10H-9681.

*OS/2 LAN Server Network Administrator Reference Volume 3: Network Administrator Tasks*. S10H-9682.

*IBM Systems Application Architecture Common Programming Interface Dialog Reference*. SC26-4356.

*IBM Systems Application Architecture Common Programming Interface Presentation Reference*. SC26-4359.

*IBM OS/2 Programming Tools and Information V1.3 Programming Guide*. S91F-9259.

*TCP/IP for OS/2 Warp Programming Reference*, SC31-8407.

*IBM Network SignON Coordinator/2 Getting Started*. S96F-8629.

## IBM Local Area Network

*IBM Token-Ring Network: Introduction and Planning Guide*. GA27-3677.

*IBM Token-Ring Network: Problem Determination Guide*. SX27-3710.

*Local Area Network: Administrator's Guide*. GA27-3748.

*IBM PC Network: Technical Reference*.

*IBM Personal Computer LAN Support Program*.

*IBM Personal Computer Baseband and Broadband*.

*IBM Cabling System Planning and Installation Guide*. GA27-3361.

*Using the IBM Cabling System with Communication Products*. GA27-3620.

*IBM Token-Ring Network Architecture Reference*. SC30-3374.

*IBM Local Area Network Technical Reference*. SC30-3587.

*IBM Local Area Network Support Program User's Guide*. SC21-8288.

## IBM 3270

*IBM 3270 Personal Computer Control Program Programming Guide*. SC23-0165.

*IBM 3270 Information Display System Character Set Reference*. GA27-2837.

*IBM PC 3270 Emulation Program, Entry Level V2.0 Programmer's Guide*. S91F-8583.

*IBM 3270 PC High Level Language API Programming Reference*. SC23-2473.

*Personal Communications Version 4.3 Emulator Programming*, SC31-8478.

## Wide Area Communications

*SNA Primary Custom Feature Description*. GC31-2509.

*Advanced Function for Communications: System Summary*. GA27-3099.

*System Network Architecture (SNA) Technical Overview*. GC30-3073.

*System Network Architecture (SNA) Format and Protocol Reference Manual*. SC30-3112.

*System Network Architecture (SNA) Formats*. GA27-3136.

*System Network Architecture (SNA) Format and Protocol Reference Manual: Management Services*. SC30-3346.

*System Network Architecture (SNA) Sessions between Logical Units*. GC20-1868.

*CCITT X.25 Recommendations, Interface between Data Terminal Equipment (DTE) and Data Circuit Terminating Equipment (DCE) for Terminals Operating in the Packet Mode on Public Data Networks*. Vol. VIII. Fascicle VIII.5. This document is useful when writing X.25 applications.

*RT PC X.25 Communication Support User's Guide*. SC33-0630.

*X.25 Interface for Attaching SNA Nodes to Packet-Switched Data Network General Information Manual*. GA27-3345.

*RT PC X.25 Communications Support Programmer's Reference*. SC33-0631.

*IBM Cryptographic Subsystem Concepts and Facilities*. GC22-9063.

*IBM X.25 Co-Processor Support Program User's Guide*. X07F-8915.

*IBM X.25 Co-Processor Support Program Programmer's Reference*. X07F-8916.

*IBM X.25 Interface Co-Processor/2 Technical Reference*. S16F-1879.

*SNA Advanced Peer-to-Peer Networking Dependent LU Requester Architecture Reference*. SV40-1010.

*Multiprotocol Transport Networking (MPTN) Architecture: Formats*. GC31-7074.

*Multiprotocol Transport Networking (MPTN) Architecture: Technical Overview*. GC31-7073.

*Telnet Protocol Specification, STD 8, RFC 854, USC/Information Sciences Institute. J. Postel and J. Reynolds* .

To view this book, use the keyword **RFC 854** with an internet search engine.

Printed copies of RFCs are available for a fee from:

SRI International, Room EJ291
333 Ravenswood Avenue
Menlo Park, CA 94025
(415) 859-3695
(415) 859-6387
FAX (415) 859-6028

## IBM NetView

*NetView Distribution Manager: General Information V1.6*. GH19-6792.

*NetView Distribution Manager: Planning*. SH19-6589.

*NetView Distribution Manager Release 6: Installation and Customization*. SH19-6794.

*NetView Distribution Manager: Operation*. SH19-6592.

*NetView Distribution Manager: User's Guide V1.6*. SH19-6795.

*NetView Distribution Manager: Diagnosis R5*. LY19-6374.

*NetView Distribution Manager: Messages and Codes V1.6*. SH19-6798.

## IBM Financial I/O Devices

*IBM 4009 Operator's manual*. GA19-5650.

*IBM 4009 Service manual/Parts catalogue*. SY19-6392.

*IBM 4009 Quick Reference Card*. GX11-6316.

*IBM 4009 Customer Setup*. GA19-5651.

*IBM 4009 Safety Instructions*. GA19-5651.

*IBM 4009 Product and Programming Description (PPD) DOS*. SH19-4015.

*IBM 4009 Product and Programming Description (PPD) OS/2*. SH19-4038.

*IBM 4700 Finance Communication System Summary*. GC31-2016.

*IBM 4700 Financial I/O Planning Guide*. GC31-3762.

*IBM 4700 Financial I/O Devices Programming Guide*. GC31-3770.

*IBM 4700 Financial I/O Devices Programming Guide for OS/2*. GC31-2661.

*IBM 4700 Finance Communication System, Controller Programming Library, Volume 5, Cryptographic Programming*. GC31-2070.

*IBM 4700 Financial I/O Devices Operating Guide*. SC31-3763.

*IBM 4712 Transaction Printer Models 1, 2, and 3 Reference Card*. SC31-3765.

*IBM 4722 Document Printer Model 3 Programming Addendum*. GC31-2928.

*IBM 4722 Document Printer Models 1, 2, and 3 Reference Card*. SC31-3767.

*IBM 4748 Document Printer Programming Guide*. SA34-2090.

*IBM 4748 Document Printer Operating Guide*. SA34-2068.

*IBM 4748 Document Printer Service Guide*. SA34-2091.

*IBM 4770 Ink Jet Transaction Printer Product Profile*. G571-0276.

*IBM 4772 Universal Financial Printer Model 1 Programming Guide*. SA34-2199.

*IBM 4772 Universal Financial Printer Model 1 and 2 Installation and Operating Guide*. GA34-2192.

*IBM 4772 Universal Financial Printer Model 1 and 2 Reference Card*. GX31-2077.

*IBM 4772 Universal Financial Printer Model 1 and 2 Service Guide*. SA34-2193.

*IBM 4777 Magnetic Stripe Unit Installation and Operating Guide*. GA34-2189.

*IBM 4777 Magnetic Stripe Unit: Programming Guide for OS/2*. SA34-2194.

*IBM 4777 Magnetic Stripe Unit: Programming Guide for DOS*. SA34-2195.

*IBM 4778 PIN-Pad Magnetic Stripe Reader Installation and Operating Guide*. GA34-2190.

*IBM 4778 PIN-Pad Magnetic Stripe Reader: Programming Guide for OS/2*. SA34-2196.

*IBM 4778 PIN-Pad Magnetic Stripe Reader: Programming Guide for DOS*. SA34-2197.

*IBM 4777 Magnetic Stripe Unit and 4778 PIN-Pad Magnetic Stripe Reader AIX Programming Guide*. SA34-2358.

*IBM 9055-001 Document Printer: Planning and Programming Guide*. SA18-7496.

*IBM 9055-002 Document Printer: Planning and Programming Guide*. SA18-7489.

*IBM 9068-A01/A03 Multi-Purpose Passbook Printer Operating Guide*. GA18-7719.

*IBM 9068-A01/A03 Multi-Purpose Passbook Printer Planning and Programming Guide*. GA18-7721.

*IBM 9068 Multi-Purpose Passbook Printer Model D01 Planning and Programming Guide*. SA18-7505.

*IBM 9068 Multi-Purpose Passbook Printer Model S01 Planning and Programming Guide*. SA18-7506.

*IBM 9068-S01 Multi-Purpose Passbook Printer Operating Guide*. SA18-7507.

*IBM 9069 Printer Planning and Programming Guide*, SA18-7525.

*IBM 9069 Operating Guide*, SA18-7524.

## Encryption and Decryption

*IBM Cryptographic Subsystem Concepts and Facilities*. GC22-9063.

*IBM 4700 Finance Communication System, Controller Programming Library, Volume 5, Cryptographic Programming*. GC31-2070.

*IBM Transaction Security System Workstation Security Services: Installation and Operating Guide*. SA34-2141.

*IBM Transaction Security System Concepts and Programming Guide: Volume 1, Access Controls and DES Cryptography*. GC31-3937.

## IBM VisualAge C++

Product web site:

`http://www.ibm.com/software/ad/vacpp/`

## IBM VisualAge Generator

Product web site:

`http://www.ibm.com/software/ad/visgen/`

IBM Redbook:

*VisualAge Generator Version 3.1 System Development Guide*, SG24-4230-02

## IBM VisualAge Smalltalk

Product web site:

`http://www.ibm.com/software/ad/smalltalk/`

IBM Redbooks:

*VisualAge for Smalltalk Handbook - Volume 1: Fundamentals*, SG24-4828-00

*VisualAge for Smalltalk Handbook - Volume 2: Features*, SG24-2219-00

## Java

IBM Java web site:

`http://www.ibm.com/software/java/`

IBM VisualAge for Java product web site:

`http://www.ibm.com/software/ad/vajava/`

IBM developerworks web site:

`http://www.ibm.com/software/developer/java/`

IBM VisualAge Developers Domain web site:

`http://www.ibm.com/software/vadd/`

IBM Redbooks:

*Programming with VisualAge for Java Version 2*, SG24-5624

*Application Development with VisualAge for Java Enterprise*, SG24-5081

## IBM Personal Communications

*Personal Communications AS/400 and 3270 for OS/2 Up and Running*, SC31-8258.

*Personal Communications AS/400 and 3270 for Windows NT Up and Running*, GC31-8314.

*Personal Communications/3270 Programmer's Guide for DOS (Entry Level)*, S20H-1774.

*Personal Communications/3270 Programmer's Guide for OS/2*, S85G-8681.

*Personal Communications/3270 Reference Guide for OS/2*, S85G-8721.

*Personal Communications Version 4.3 for Windows 98 and Windows NT Reference, Volumes 1 and 2* , SC31-8682, SC31-8680.

*Personal Communications Windows NT Quick Beginnings*, GC31-8679.

## IBM Communications Server

*IBM Communications Server for OS/2 Warp Quick Beginnings*, GC31-8189.

*IBM Communications Server for Windows NT Quick Beginnings*, GC31-8424.

*eIBM Network Communications Server for OS/2 Guide to AnyNet*, GC31-8193, GC31-8320

## WorkSpace On-Demand

*WorkSpace On-Demand Road Map Release 2.0*, SG24-5117

*WorkSpace On-Demand Handbook Release 2.0*, SG24-5117

*WorkSpace On-Demand Handbook (Release 1)*, SG24-2028

*WorkSpace On-Demand Early Customer Experiences*, SG24-5107

*IBM Up and Running! OS/2 Warp Server*, S25H-8004

*WorkSpace On-Demand Administrator's Guide.,*, on the web at `www.ibm.com/software/network/workspace/library`

## MQSeries

*MQSeries for Windows NT and Windows 2000 Quick Beginnings*, GC34-5389

*MQSeries for OS/Warp Quick Beginnings*, GC33-1868

*MQSeries Planning Guide*, GC33-1349

*MQSeries Intercommunication*, SC33-1872

*MQSeries Clients*, GC33-1632

*MQSeries System Administration*, SC33-1873

*MQSeries Command Reference*, SC33-1369

*MQSeries Programmable System Management*, SC33-1482

*MQSeries Messages*, GC33-1876

*MQSeries Application Programming Guide*, SC33-0807

*MQSeries Application Programming Reference*, SC33-1673

*MQSeries Using C++*, SC33-1877

# Index

## Special Characters


_AE translation routine 573
_EA translation routine 573


## Numerics


3270
  starting emulator (DOS) 10
3270 books 704
3270 emulator API
  characteristics 593
  invocation 593
  services 594
4019 laser printer 657
4029 laser printer 657
4039 laser printer 657
4201 Proprinter 657
4704 read compatibility mode (MSR/E server) 217
4712 transaction printer 141, 657
4722 document printer 141, 657
5575 printer 657
5577 printer for Korea and Taiwan 657
9068-D01 Multi-Purpose Passbook Printer 167


## A


AA (ask for asynchronous events) supervisor local function 6
abbreviations 677
AC (acquire a parallel printer) function in printer manager server 201
account number, personal (PAN) 232
acquire a parallel printer (AC) function in printer manager server 201
activate and deactivate timers (T0–T8) supervisor local function 21
activate service (AS) function in local resource manager 523
ACTLU received (SNA server) 51
ACTPU received (SNA server) 50
AD (advise) function in DDE access server 443
Add record (AR) function
  electronic journal server 392
add user to user profile file (AU) function in system manager server 476
add user to user profile file (Year-2000) (AX) function in system manager server 476
advise (AD) function in DDE access server 443
AE (ASCII-EBCDIC translation)
  AE routine 573
AI (activate application intervention) function in local resource manager 522
AL (allocate physical electronic journal) function in electronic journal server 392
alerts (in system manager server)
  management 498
  resolution notification (UN) function 503
  update alerts status (UA) function 489
allocate physical electronic journal (AL) function 392
alternate mode, set (SA) function in local resource manager 542
AN (alerts notification) function in system manager server 499
application data maintenance (in system manager server) 474
application program disconnection (EJ) supervisor local function 11
application programming
  3270 emulator API 593
  3270 emulator invocation 593
  3270 emulator services 594
applications, query (QA) function in DDE access server 440
AR function
  electronic journal server 392
arm operator panel for user input (AR) function in financial printer server 146
arm the PIN pad or MSR for data input (AR) function 228
AS (activate service) function in local resource manager 523
AS (associate semaphore) supervisor local function 7
ask for asynchronous events (AA) supervisor local function 6
associate semaphore (AS) supervisor local function 7
asynchronous events 558
  ask for events (AA) supervisor local function 6
  asking for TT request (Z4) supervisor local function 29
  event notification or cancellation (Z5) supervisor local function 30
  event notification or cancellation with RMTRPLY 32
  extended event notification or cancellation (ZN) supervisor local function 31
  managing 3
  notation for codes xxvii
  start posting (SP) supervisor local function 20
  stop posting (TP) supervisor local function 23
  switch printer mode (Z6), FBSI/2 671
  wait (WM) supervisor local function 24
Asynchronous events 565
attributes, get (GA) function in PPC server 112
AU (add user to user profile file) function in system manager server 476
audience for this book xxv
auto connect 562, 567


© Copyright IBM Corp. 1992, 2002

AX (add user to user profile file (Year-2000)) function in system manager server 476

## B

backout operation (BO) function in object postbox server 580
banking self-service books 703
begin transmission (TB) function in store-for-forwarding server 418
bibliography 703
BID protocol (SNA server) 39
BIND parameters and protocols (SNA server) 38
bit positions, convention for xxvii
BO (backout operation) function in object postbox server 580
books for LANDP xxviii
bracket protocol (SNA server) 38

## C

CA (change printer assignment) function in local resource manager 524
call CICS (CC) function in CICS interface serve 429
called and caller session initiation (X.25 server) 92
cancel pending functions (CC) function in local resource manager 526, 546
CB (close batch) function in shared-file server 264
CC (call CICS) function in CICS interface server 429
CC (cancel pending functions) function in local resource manager 526, 546
CD (change database) function in ODBC query server 360
CD (clear operator panel display) function in financial printer server 148
CH (check the write status) function in MSR/E server 208
chaining protocol (SNA server) 40
chaining value, initial (IV) function in PIN pad server 241
change database (CD) function in ODBC query server. 360
change direction protocol (SNA server) 41
change LU priority (CP) function in local resource manager 529
change password (CP) function in system manager server 463
change print density mode (CM) function in local resource manager 528
change printer assignment (CA) function in local resource manager 524
change printer to local mode (LP) function in local resource manager 549
change printer to remote mode (HP) function in local resource manager 548
character, load user-defined (LD) function in 4748 printer server 169, 182
characters download (DU) function in financial printer server 155
check NMVT (CN) function in system manager server 503
check status (RD) function in object postbox server 583
check the write status (CH) function in MSR/E server 208
CICS, call (CC) function in CICS interface serve 429
CICS, read (RC) function in CICS interface server 433
cipher block chaining 76
ciphering operations, normal request unit 89
CL (close conversation) function in PPC server 111
CL (close PIN pad or MSR) function in PIN pad server 236
clear operator panel display (CD) function in financial printer server 148
client/server mechanism
  supervisor local functions 3
clipboard functions (in DDE access server) 449
close batch (CB) function in shared-file server 264
close query mode (CQ) function in ODBC query server. 368
close query mode (CQ) function in query server 319
CM (change print density mode) function in local resource manager 528
CM (compress data) function in SNA compression server exit 71
CN (check NMVT) function in system manager server 503
command execution (EC) function in DDE access server 444
command run (RC) function in DDE access server 440
commit work (CW) function in ODBC query server. 368
commit work (CW) function in query server 319
common data maintenance (in system manager server) 487
communication
  books 704
communication services in 3270 emulator 648
communication status set (CS) function in store-for-forwarding server 410
Communications Server
  books 707
compatibility of LANDP 3270 emulator HLLAPI with other 3270 emulator APIs 653
compress data (CM) function in SNA compression server exit 71
compression of data with DOS 68
compression server 69
contention (in program-to-program communication server) 105, 106
control functions (in system manager server) 462
conventions, in this book xxvi
copy service in 3270 emulator 606, 639
copy text (CT) function in DDE access server 449

CP (change LU priority) function in local resource manager 529
CP (change password) function in system manager server 463
CQ (close query mode) function in ODBC query server. 368
CQ (close query mode) function in query server 319
create index (DI) function in query server 333
cryptographic interface 75
  example of SNA session-level encryption 85
  SNA session-level encryption 85
cryptography
  books 706
CS (set communication status) function in store-for-forwarding server 410
CT (copy text) function in DDE access server 449
CW (commit work) function in ODBC query server. 368
CW (commit work) function in query server 319

# D

DA (deallocate physical electronic journal) function in electronic journal server 394
data compression with DOS 68
  compression (CM) function in SNA compression server exit 71
  decompression (UC) function in SNA compression server exit 71
data input, arm PIN pad or MSR (AR) function 228
data insertion (ID) function in DDE access server 438
data management servers
  ODBC query server 353
  query server 301
data received (DR) function in EXFS user-written exit server 663
data request (RD) function in DDE access server 447
data security protocol (SNA server) 41
data stream
  3270 extended 655
  PIN pad 227
data, get (GD) function in DDE access server 437
data, read (RD)
  function in 4748 printer server 185
data, read (RD) function in financial printer server 160
data, receive (RD) function in PPC server 115
data, retrieve application program (GU) function in system manager server 477
data, send (SD) function in PPC server 116
data, update application program (SU) function in system manager server 482
Database Server support (OS/2 Warp) 301
database, change (CD) function in ODBC query server. 360
DB/2 database support 301
DB2 for OS/2 xxvii
DB2 Universal Database xxvii
DD (define DBD) function in query server 332
DDT (trace tools), start 10
deactivate and activate timers (T0–T8) supervisor local function 21
deallocate physical electronic journal (DA) function 394
decompress data (UC) function in SNA compression server exit 71
decrypt (X'0031') in cryptographic interface 81
decryption and encryption books 706
define DBD (DD) function in query server 332
define index (DI) function in query server 333
define PCB (DP) function in query server 334
defined users list, retrieve (RE) function in system manager server 467
definitions of terms 677
Delete record (DL) function
  electronic journal server 394
delete row (DR) function in ODBC query server. 370
delete row (DR) function in query server 321
delete user in user profile file (DU) function in system manager server 477
density mode of printing, change (CM) function in local resource manager 528
describe query (DQ) function in ODBC query server. 369
describe query (DQ) function in query server 320
describe table (DT) function in ODBC query server. 370
describe table (DT) function in query server 321
device servers, events from 26
device services in 3270 emulator 646
DI (define index) function in query server 333
direct access method (in shared-file server) 255
direct indexed access method (in shared-file server) 255
disable storing (DS) function in store-for-forwarding server 412
disconnection
  application program (EJ) supervisor local function 11
display, write (WD) function in PIN pad server 249
dissociate semaphore (DS) supervisor local function 9
Distributed Relational Database Architecture (DRDA) 301
distributor server, shared-file 258
DL function
  electronic journal server 394
DOS box (VDM) 555
DOS compression server exit 68
DosBox (LANDP for Linux) 563
DOSEMU 563
download user characters (DU) function in financial printer server 155
download user-defined characters (LD) function in 4748 printer server 169
DP (define PCB) function in query server 334

DQ (describe query) function in ODBC query server 369
DQ (describe query) function in query server 320
DR (data received) function in EXFS user-written exit server 663
DR (delete row) function in ODBC query server. 370
DR (delete row) function in query server 321
DRDA (Distributed Relational Database Architecture) 301
DS (disable storing) function in store-for-forwarding server 412
DS (dissociate semaphore) supervisor local function 9
DT (describe table) function in ODBC query server. 370
DT (describe table) function in query server 321
DU (delete user in user profile file) function in system manager server 477
DU (download user characters) function in financial printer server 155

# E

EA (EBCDIC-ASCII translation) routine 573
EC (end connection) function in local resource manager 519
EC (execute command) function in DDE access server 444
ED (erase DBD) function in query server 335
EE (simulate hot-key) supervisor local function 10
EHC#ALRU.DAT file 501
EHC#RESU.DAT file 501
EHCALRH user exit (in system manager server) 503
EHCBOXM.EXE monitor program 555
EHCBOXS.CFG file 557, 564
EHCCOMP (SNA compression server exit) 70
EHCCONN SNA connection program 54
EHCDBTR (ASCII-EBCDIC translation server) 569
EHCEXITS.C user exit 557, 564
EHCINFO program 14
EHCLAD (DDE access server) 436
EHCLRMGR (local resource manager) 518
EHCODB## (ODBC query server) 359
ehcplug.c 563
EHCSFD## (shared-file distributor server) 262
EHCSFR## (shared-file replicator server) 262
EHCSQL## (query server) 310
EHCTRAN (CICS interface server) 428
ehcvdmgr 563
EHCVDMGR.EXE 556
EHCVDMGR.EXE server 555
ehcvdmir 563
EHCVDMIR.EXE 556
EHCVDMVD.SYS driver program 555
EHCVDSPV.COM 556, 563
EHCVDVXD.DLL 556
EI (end operator intervention) function in local resource manager 531
EI (erase index) function in query server 336
EJ (disconnect an application program) supervisor local function 11
ELECJO## (electronic journal server) 390
electronic journal server
  database error messages 392
  server functions 392
  using the server 390
emulate hot-key (EE) supervisor local function 10
EN (enable storing) function in store-for-forwarding server 412
enable storing (EN) function in store-for-forwarding server 412
encrypt (X'0030') in cryptographic interface 80
encryption and decryption
  books 706
end an application program (EJ) supervisor local function 11
end connection (EC) function in local resource manager 519
end operator intervention (EI) function in local resource manager 531
end query (EQ) function in ODBC query server. 371
end query (EQ) function in query server 322
end transmission (TE) function in store-for-forwarding server 419
EP (erase PCB) function in query server 336
EQ (end query) function in ODBC query server. 371
EQ (end query) function in query server 322
erase DBD (ED) function in query server 335
erase index (EI) function in query server 336
erase PCB (EP) function in query server 336
error number and message tables (in system manager server) 507
ES (unload LANDP)
  supervisor local function 12
ES (unload LANDP) supervisor local function 12
ESC/P printers for People's Republic of China 657
event codes, notation for xxvii
event notification
  event codes xxvii
  I/O device servers 26
event query (QE) supervisor local function 17
examples
  4712/4722 printer in passbook mode 154
  4748 printer sharing 169
  asynchronous event codes xxvii
  cryptographic interface 85
  electronic journal, physical 381
  electronic journal, searching 388
  function codes xxvii
  ODBC query server 355
  query server 304

examples *(continued)*
return codes xxvii
shared-file server 296
SNA server 71
SNA session-level encryption 85
exclusive mode, set (SE) function in local resource manager 544
exclusive use of record (in shared-file server) 276
execute command (EC) function in DDE access server 444
extended 3270 data stream 655
external call interface (CICS) 427

## F

FBHLLAPI resident module 616
FBSI/2 switch printer mode (Z6 function) 671
FBSS
books 703
fetch next record (FN) function in shared-file server 268
fetch previous record (FP) function in shred-file server 269
fetch row (FR) function in ODBC query server. 372
fetch row (FR) function in query server 323
fetch unique record (FU) function in shred-file server 271
file locking in on-line mode (in shared-file server) 261
file query (QD) function in store-for-forwarding server 414
financial printer server
format definition
4009 printer 150
4712 and 4722 printers, example in passbook mode 154
4712, 4722, and 9068-S01 printers 150
4722-003 printer with magnetic stripe 154
4772 printer 150
9068-S01 printer 150
MICR reader, 9069 printer 141
FN (fetch next record) function in shared-file server 268
format definition
4722-003 printer with magnetic stripe 154
FP (fetch previous record) function in shred-file server 269
FR (fetch row) function in ODBC query serverfunction in ODBC query server. 372
FR (fetch row) function in query server 323
FU (fetch unique record) function in shred-file server 271
function codes
notation for xxvii
notation for operating environment xxvi
function management headers protocol (SNA server) 41

## G

GA (generate message authentication code) function in PIN pad server 240
GA (get attributes) function in PPC server 112
Gaiji character, load user-defined (LD) function in 4748 printer server 182
gateway (SNA server) 35
GD (get data) function in DDE access server 437
GD (retrieve system date and time) function in system manager server 485
generate message authentication code (GA) function in PIN pad server 240
get application program data (GU) function in system manager server 477
get attributes (GA) function in PPC server 112
get data (GD) function in DDE access server 437
get defined record structures server functions 493
get defined users list (RE) function in system manager server 467
get LANDP workgroup common data (RD) function in system manager server 489
get list of users holding locks (RK) function in system manager server 468
get LOG record (R1) function in system manager server 509
get LOG record (RL) function in system manager server 507
get record format (RR) function in system manager server 493
get signed-on PC (GP) function in system manager server 465
get signed-on PC (Year-2000) (GW) function in system manager server 466
get signed-on user (GI) function in system manager server 464
get signed-on user (GL) function in system manager server 463
get signed-on users list (GO) function in system manager server 470
get signed-on users list (RO) function in system manager server 469
get system date and time (GD) function in system manager server 485
get system status (GG) function in system manager server 487
GG (get system status) function in system manager server 487
GI (get signed-on user) function in system manager server 464
GL (get signed-on user) function in system manager server 463
glossary 677
GO (get signed-on users list) function in system manager server 470

GP (get signed-on PC) function in system manager server 465
GU (retrieve application program data) function in system manager server 477
GW (get signed-on PC (Year-2000) function in system manager server 466

## H

header, read (RH) function in shared-file server 290
high-level functions (in DDE access server) 437
highlighting, in this book xxvi
host computer, events from 6
hot-key simulation (EE) supervisor local function 10
HP (change printer to remote mode) function in local resource manager 548

## I

I/O books 705
I/O devices, events from 6
I/O structure blocks (ODBC query server) 357
I/O structure blocks (query server) 306
IC (initiate conversation) function in DDE access server 445
ID (insert data) function in DDE access server 438
identification, user (in system manager server) 462
II (inquire information) supervisor local function 14
IL (inquire logical) function in electronic journal server 395
IN (initialize) supervisor local function 16
IND$FILE command 3270 HLLAPI 655
indexed access method (in shared-file server) 255
indexed sequential access method (in shared-file server) 255
information about IBM products 703
information about query (DQ) function in ODBC query server. 369
information about query (DQ) function in query server 320
information inquiry (II) supervisor local function 14
information of system status (SI) supervisor local function 18
information, server (ZI) function in SNA server 66
initial chaining value (IV) function in PIN pad server 241
initialize (IN) supervisor local function 16
initiate conversation (IC) function in DDE access server 445
initiate session (IS) function in local resource manager 536
initiate store (SI) function in store-for-forwarding server 417
input data formats MSR/E server 216
input queue, list (LI) function in object postbox server 580
inquire information (II) supervisor local function 14
inquire logical (IL) function in electronic journal server 395
inquire physical (IP) function in electronic journal serve 396
inquire status (IS) function in store-for-forwarding server 413
insert data (ID) function in DDE access server 438
insert row (IR) function in ODBC query server. 373
insert row (IR) function in query server 323
intervention by operator, end (EI) function in local resource manager 531
IP (inquire physical) function in electronic journal server 396
IR (insert row) function in ODBC query server. 373
IR (insert row) function in query server 323
IS (initiate session) function in local resource manager 536
IS (inquire status) function in store-for-forwarding server 413
IV (load initial chaining value) function in PIN pad server 241

## J

Java websites and redbooks 707

## K

KD (peek data) function in DDE access server 446
key add (X'0050') in cryptographic interface 83
key load (LK) function in PIN pad server 242
key record delete (X'0052') in cryptographic interface 84
key records in cryptographic interface 78
key update (X'0051') in cryptographic interface 84
keyboard
  events from 6
knowledge, prerequisite xxv

## L

LAN books 704
LANDP family books xxviii
LANDP for DOS applications in Linux workstations 563
LANDP for DOS applications in OS/2 or Windows workstations 555
language interface module (LIM) in 3270 emulator 616, 654
laser printer support 163
LD (load user-defined character) function in 4748 printer server 169, 182
LI (list input queue) function in object postbox server 580
Linux
  semaphore-handling 5
  supported distributions xxviii
Linux DosBox support 563
list input queue (LI) function in object postbox server 580

list of users holding locks, retrieve (RK) function in system manager server 468
list output queue (LO) function in object postbox server 581
LK (load key) function in PIN pad server 242
LM (load master key) function in PIN pad server 243
LO (list output queue) function in object postbox server 581
LO (lock topic) function in DDE access server 439
load initial chaining value (IV) function in PIN pad server 241
load key (LK) function in PIN pad server 242
load master key (LM) function in PIN pad server 243
load PIN verification parameters (LP) function 244
load program (LP) function in DDE access server 439
load user-defined character (LD) function in 4748 printer server 169, 182
local mode, change printer to (LP) function in local resource manager 549
lock topic (LO) function in DDE access server 439
low-level functions (in DDE access server) 442
LP (change printer to local mode) function in local resource manager 549
LP (load PIN verification parameters) function in PIN pad server 244
LP (load program) function in DDE access server 439
LU priority, change (CP) function in local resource manager 529
LU–LU session (SNA server) 42

## M

MAC (message authentication code) in PIN pad server 240
MAIL program 588
maintaining data, application 474
master key load (LM) function in PIN pad server 243
message tables (in system manager server) 507
MICR reader, 9069 printer 141
  financial printer server flag 145
  read data (RD) function 160
  reply DATA values 161
Microsoft Windows
  supported releases xxvii
MO (send message to NetView operator) function in system manager server 502
mouse events 6
MQSeries Link server 137
  begin transaction (BT) 123
  checkpoint (CP) 124
  close queue (CQ) 125
  close session (CS) 124
  end transaction (ET) 126
  integration With MQSeries 122
  MQGET (GQ function) 127
  MQPUT (PQ function) 131
MQSeries Link server *(continued)*
  MQPUT1 (PQ1 function) 134
  rollback (RB function) 135
  using 120
MSR device, arm (AT) function in PIN pad server 234
MSRE47## (MSR/E server) 205
MVDM relay 555

## N

NetView books 705
NetView operator, send message to (MO) function in system manager server 502
NMVT, check (CN) function in system manager server 503
non-4704 read compatibility mode MSR/E server 216
notification of alerts (AN) function in system manager server 499
notification of resolution (UN) function in system manager server 503
number of personal account (PAN) 232

## O

OB (open batch) function in shared-file server 286
obtain RDBMS information (RI) function in ODBC query server 374
ODBC query server 353, 379
  as an access path to the RDBMS 354
  CD (change database) function in ODBC query server. 360
  close query mode (CQ) 368
  close session (CS) 361
  commit work (CW) 368
  delete row (DR) 370
  describe query (DQ) 369
  describe table (DT) 370
  end query (EQ) 371
  fetch row (FR) 372
  grant (GF) 361
  habilitate logging (HL) 362
  I/O structure blocks 357
  inhibit logging (IL) 362
  insert row (IR) 373
  long descriptor I/O block 357
  modes of operation 354
  not-logged-on mode 360
  obtain RDBMS information (RI) 374
  open query mode (OQ) 362
  open session (OS) 363
  pre-fetch I/O block 358
  query mode 365
  replace row (RR) 374
  request reference
    any mode 360
    query mode 368
  return code and explanation handling 356

ODBC query server *(continued)*
revoke (RF) 364
rollback work (RW) 375
set parameters (SP) 375
short descriptor I/O block 357
structured query (SQ) 376
supported data types 366
test status (TS) 364
using 358
offline transaction (in shared-file server) 296
OIA (operator information area) service in 3270 emulator 609
OP (open conversation) function in PPC server 113
OP (open PIN pad) function in PIN pad server 245
OPBS (object postbox server) 579
open batch (OB) function in shared-file server 286
open conversation (OP) function in PPC server 113
open PIN pad (OP) function 245
open query mode (OQ) function in ODBC query server. 362
open query mode (OQ) function in query server 315
operating environment, notation for xxvi
operator information area (OIA) service in 3270 emulator 609
operator interface, start 10
operator intervention, end (EI) function in local resource manager 531
operator receiver, update (UO) function in system manager server 492
operator services in 3270 emulator 618
OQ (open query mode) function in ODBC query server. 362
OQ (open query mode) function in query server 315
organization of this book xxv
OS/2 virtual DOS machine 555, 563
output data formats (MSR/E server). 219
output data formats MSR/E server 219
output queue, list (LO) function in object postbox server 581

# P

PAN (personal account number) 232
parallel redirection settings (RM) function in financial printer server 161
parameters, set (SP) function in ODBC query server 375
parameters, set (SP) function in query server 325
password, change (CP) function in system manager server 463
paste text (PT) function in DDE access server 450
PD (poke data) function in DDE access server 446
PD (print data) function in EXFS user-written exit server 664
peek data (KD) function in DDE access server 446
personal account number (PAN) 232
Personal Communications books 707
personal computer books 703
personal identification number (PIN) pad server 223, 251
physical electronic journal
allocate (AL) 392
PIN verification parameters, load (LP) function 244
PINP47## (PIN pad server) 226
poke data (PD) function in DDE access server 446
portability considerations in 3270 emulator 653
posting events
start (SP) 20
stop (TP) 23
PPC#### (name of PPC server) 108
PR (process read data) function in EXFS user-written server 107
PR4748.UDC 170
PR4748## (4748 printer server) 171
PR4770## (4770 printer server) 190
PR47X2## (financial printer server) 144
prerequisite knowledge xxv
presentation services in 3270 emulator 630
preventing locking by using timeouts 28
print data (PD) function in EXFS user-written exit server 664
print density mode, change (CM) function in local resource manager 528
printer manager server 199
priority of LU, change (CP) function in local resource manager 529
process read data (PR) function in EXFS user-written server 107
process send data (PS) function in EXFS user-written server 108
program load (LP) function in DDE access server 439
protect record in shared-file server 282, 286
PRTMGR (printer manager server) 200
PRTMON.EXE 161
PS (process send data) function in EXFS user-written server 108
PT (paste text) function in DDE access server 450

# Q

QA (query applications) function in DDE access server 440
QD (query file) function in store-for-forwarding server 414
QE (query event) supervisor local function 17
QP (query PCB) function in shared-file server 287
query functions
information (II) supervisor local function 14
query event (QE) supervisor local function 17
query status (TS) function in ODBC query server. 364
query status (TS) function in query server 316

query, describe (DQ) function in ODBC query server. 369
query, describe (DQ) function in query server 320
query, end of (EQ) function in ODBC query server. 371
query, end of (EQ) function in query server 322
query, structured (SQ) function in ODBC query server 376
query, structured (SQ) function in query server 326
quiesce protocol (SNA server) 42

## R

R1 (retrieve LOG record) function in system manager server 509
random number generation (X'0040') in cryptographic interface 82
RC (read CICS) function in CICS interface server 433
RC (read code page) function in financial printer server 159
RC (run command) function in DDE access server 440
RD (read data)
  function in financial printer server 160
RD (read data) function in 4748 printer server 185
RD (read from the IBM 47xx MSR/E) function in MSR/E server 214
RD (read from the IBM 47xx PIN pad) function in PIN pad server 246
RD (receive data) function in PPC server 115
RD (register device and check status) function in object postbox server 583
RD (request data) function in DDE access server 447
RD (retrieve LANDP workgroup common data) function in system manager server 489
RDBMS information, obtain (RI) function in ODBC query server 374
RE (retrieve defined users list) function in system manager server 467
readers of this book xxv
RECEIVE command 3270 HLLAPI 655
receive data (RD) function in PPC server 115
redirecting data from parallel port to serial printer 161
redirection mode, set (RM) function in financial printer server 161
register device and check status (RD) function in object postbox server 583
related information xxviii
remote mode, change printer to (HP) function in local resource manager 548
remote sign-off (RF) function in system manager server 467
replace row (RR) function in ODBC query server 374
replace row (RR) function in query server 324
replicator server, shared-file 259
request data (RD) function in DDE access server 447
reset electronic journal file (RS) function 400
reset store-for-forwarding file (RS) function 417
resolution notification (UN) function in system manager server 503
response protocol (SNA server) 41
retrieval of defined record structures server functions 493
retrieve application program data (GU) function in system manager server 477
retrieve defined users list (RE) function in system manager server 467
retrieve LANDP workgroup common data (RD) function in system manager server 489
retrieve lists of users holding locks (RK) function in system manager server 468
retrieve LOG record (R1) function in system manager server 509
retrieve LOG record (RL) function in system manager server 507
retrieve record format (RR) function in system manager server 493
retrieve signed-on users list (RO) function in system manager server 469
retrieve system date and time (GD) function in system manager server 485
retrieving messages 577
return codes
  notation for xxvii
RF (remote sign-off) function in system manager server 467
RH (read header) function in shared-file server 290
RI (obtain RDBMS information) function in ODBC query server 374
RK (retrieve lists of users holding locks) function in system manager server 468
RL (release a parallel printer port) function in printer manager server 202
RL (release electronic journal) function in electronic journal server 398
RL (retrieve LOG record) function in system manager server 507
RM (read message) function in object postbox server 584
RM (set redirection mode) function in financial printer server 161
RN (read serial number) function in PIN pad server 247
RN (retrieve user profile by record number) function in system manager server 478
RO (retrieve signed-on users list) function in system manager server 469
rollback work (RW) function in ODBC query server 375
rollback work (RW) function in query server 325
router return codes
  notation for xxvii
row, delete (DR) function in ODBC query server. 370
row, delete (DR) function in query server 321
row, fetch (FR) function in ODBC query server. 372

row, fetch (FR) function in query server 323
row, insert (IR) function in ODBC query server. 373
row, insert (IR) function in query server 323
row, replace (RR) function in ODBC query server 374
row, replace (RR) function in query server 324
RR (replace row) function in ODBC query server 374
RR (replace row) function in query server 324
RR (retrieve record format) function in system manager server 493
RS (reset electronic journal file) function in electronic journal server 400
RS (reset store-for-forwarding file) function in store-for-forwarding server 417
RTR protocol (SNA server) 40
RU (retrieve user profile) function in system manager server 480
run command (RC) function in DDE access server 440
running LANDP for DOS applications in Linux workstations 563
running LANDP for DOS applications in OS/2 or Windows workstations 555
RW (rollback work) function in ODBC query server 375
RW (rollback work) function in query server 325
RX (retrieve user profile (Year-2000)) function in system manager server 481

# S

SA (set alternate mode) function in local resource manager 542
sample files for MVDM relays 557, 564
SC (set code page) function in financial printer server 162
scan code of keystroke in 3270 emulator 603
SD (send data) function in PPC server 116
SD (set system date and time) function in system manager server 486
SD (statistics on DBD) function in query server 349
SE (set exclusive mode) function in local resource manager 544
search
  hints in electronic journal server 390
SECY (cryptographic interface server) 79
segmentation protocol (SNA server) 38
select current electronic journal (SL) function 401
semaphore-handling
  Linux 5
semaphores
  association (AS) supervisor local function 7
  dissociation (DS) supervisor local function 9
  events 6
  managing 3
SEND command 3270 HLLAPI 655
send data (SD) function in PPC server 116
send data, process (PS) function in EXFS user-written server 108
sending alerts 499, 503
sequential access method (in shared-file server) 255
serial number, read (RN) function in PIN pad server 247
server information (ZI) function in SNA server 66
server programming
  name substitution 565
server return codes
  notation for xxvii
servers
  ODBC query server 353
servers supplied with LANDP
  query server 301
  system manager 459
service activation (AS) function in local resource manager 523
service marks 674
service suspension (SS) function in local resource manager 537
session initiation (IS) function in local resource manager 536
session release 557, 562, 564, 567
set alternate mode (SA) function in local resource manager 542
set code page (SC) function in financial printer server 162
set communication status (CS) function in store-for-forwarding server 410
set exclusive mode (SE) function in local resource manager 544
set parameters (SP) function in ODBC query server 375
set parameters (SP) function in query server 325
set redirection mode (RM) function in financial printer server 161
set system date and time (SD) function in system manager server 486
SF (sign-off) function in system manager server 471
SFORFORW (store-for-forwarding server) 408
shared-file distributor server 258
shared-file replicator server 259
shared-file server
  server functions
    begin transaction (BT) 264
    checkpoint (CP) 265
    close batch (CB) 264
    close on-line (CO) 265
    close session (CS) 266
    delete record (DL) 266
    enable logging (HL) 276
    end transaction (ET) 267
    erase shared-file data (ZD) 295
    fetch next record (FN) 268
    fetch previous record (FP) 269
    fetch unique record (FU) 271

shared-file server *(continued)*
server functions *(continued)*
get next record (GN) 273
get previous record (GP) 274
get unique record (GU) 275
grant (GF) 272
habilitate logging (HL) 276
hold next record (HN) 276
hold previous record (HP) 277
hold unique record (HU) 278
inhibit logging (IL) 281
initialize DBD (ID) 280
initialize pointers (IP) 281
insert record (IS) 282
keep next record (KN) 283
keep previous record (KP) 284
keep unique record (KU) 284
open batch (OB) 286
open on-line (OO) 286
open session (OS) 287
query PCB (QP) 287
read header (RH) 290
replace record (RP) 292
request exclusive use (EX) 268
revoke (RF) 289
rollback (RB) 288
statistic request (SR) 293
test status (TS) 294
using the server 262
SHFILE## (shared-file server) 262
shift state of keystroke in 3270 emulator 603
SI (initiate store) function in store-for-forwarding server 417
SI (system status information) supervisor local function 18
sign-on (SN) function in system manager server 471
sign-on (SO) function in system manager server 473
signed-on PC (GP) function in system manager server 465
signed-on PC (Year-2000) (GW) function in system manager server 466
signed-on user (GI) function in system manager server 464
signed-on user (GL) function in system manager server 463
signed-on users list, get (GO) function in system manager server 470
signed-on users list, retrieve (RO) function in system manager server 469
simulate hot-key (EE) supervisor local function 10
SL (select current electronic journal) function in electronic journal server 401
SM (send message and message header) function in object postbox server 584
SMGR (system manager server) 461
SN (sign-on) function in system manager server 471
SNA## (SNA server) 51
SO (sign-on) function in system manager server 473
SP (set parameters) function in ODBC query server 375
SP (set parameters) function in query server 325
SP (start posting events) supervisor local function 20
SPV (supervisor local function server) 4
SQ (structured query) function in ODBC query server 376
SQ (structured query) function in query server 326
SS (suspend service) function in local resource manager 537
SSCP–LU session (SNA server) 48
number of user sessions per workstation 49
user sessions per workstation 49
start posting events (SP) supervisor local function 20
start session (IS) function in local resource manager 536
start transmission (TB) function in store-for-forwarding server 418
status functions
information (SI) supervisor local function 18
status, get (GS) function in PPC server 113
stop posting events (TP) supervisor local function 23
stop transmission (TE) function in store-for-forwarding server 419
store initiation (SI) function in store-for-forwarding server 417
storing messages 577
structure of this book xxv
structured query (SQ) function in ODBC query server 376
structured query (SQ) function in query server 326
SU (update application program data) function in system manager server 482
supervisor local functions
activate and deactivate timers (T0–T8) 21
ask for asychronous events (AA) 6
associate semaphore(AS) 7
asynchronous request—asking for TT request (Z4) 29
asynchronous request—event notification or cancellation (Z5) 30
disconnect an application program (EJ) 11
dissociate semaphore (DS) 9
emulator activation 10
exit LANDP servers 11
extended asynchronous event notification or cancellation (ZN) 31
information (II) supervisor local function 17
initialize (IN) supervisor local function 16
inquire information (II) 14
preventing locking 28
simulate hot-key (EE) 10

supervisor local functions *(continued)*
start posting events (SP) 20
stop posting events (TP) 23
system status information (SI) 18
timer activation, deactivation (T0–T8) 21
unload LANDP (ES) 12
using 4
using a WM function timeout 25
wait for asynchronous events (WM) 24
supported data types
ODBC query server 366
suspend service (SS) function in local resource manager 537
switch printer mode within an application (Z6 function) 671
synchronization of date and time 484
system and user LOG management server functions 505
system LOG management 505
system manager server
operator interface, start 10
system status information (SI) supervisor local function 18
system status, get (GG) function in system manager server 487

## T

T0–T8 (activate and deactivate timers) supervisor local function 21
table, describe (DT) function in ODBC query server. 370
table, describe (DT) function in query server 321
TB (begin transmission) function in store-for-forwarding server 418
TC (terminate conversation) function in DDE access server 448
TCP/IP Wide Area communications server 137
TE (stop transmission) function in store-for-forwarding server 419
terminate conversation (TC) function in DDE access server 448
terms, definitions of 677
text copy (CT) function in DDE access server 449
text paste (PT) function in DDE access server 450
timeouts 624
timer activation and deactivation (T0–T8) supervisor local function 21
topic lock (LO) function in DDE access server 439
topic unlock (UL) function in DDE access server 441
TP (stop posting events) supervisor local function 23
trace tools, start 10, 623
trace tools, stop 623
trademarks 674
Transaction Security System (TSS) 75
translation routines
AE routine 573
EA routine 573
translation routines (ASCII-EBCDIC, EBCDIC-ASCII) 573
translation server (ASCII-EBCDIC, EBCDIC-ASCII) 569
transmission begin (TB) function in store-for-forwarding server 418
transmission end (TE) function in store-for-forwarding server 419
TSS (Transaction Security System) 75
TT (timer-generated request) asynchronous request for (Z4) supervisor local function 29

## U

UA (update alerts status) function in system manager server 489
UC (decompress data) function in SNA compression server exit 71
UD (update LANDP workgroup common data) function in system manager server 490
UI (update LANDP workgroup identification) function in system manager server 491
UL (unlock topic) function in DDE access server 441
UL (update logging status) function in system manager server 491
UM (update operator messages status) function in system manager server 492
UN (resolution notification) function in system manager server 503
UN (unadvise) function in DDE access server 448
unadvise (UN) function in DDE access server 448
uncompress data (UC) function in SNA compression server exit 71
unload LANDP (ES) supervisor local function 12
unlock topic (UL) function in DDE access server 441
UO (update message operator receiver) function in system manager server 492
update alerts status (UA) function in system manager server 489
update application program data (SU) function in system manager server 482
update LANDP workgroup common data (UD) function in system manager server 490
update LANDP workgroup identification (UI) function in system manager server 491
update logging status (UL) function in system manager server 491
update message operator receiver (UO) function in system manager server 492
update operator messages status (UM) function in system manager server 492
update user profile (UU) function in system manager server 483

update user profile (UX) function in system manager server 484
user characters download (DU) function in financial printer server 155
user input, arm operator panel for (AR) function in financial printer server 146
user-defined character, load (LD) function in 4748 printer server 169, 182
users of this book xxv
utility programs
  AE routine 573
  EA routine 573
utility services in 3270 emulator 652
UU (update user profile (Year-2000)) function in system manager server 484
UU (update user profile) function in system manager server 483

## V

VA (verify message authentication code) function in PIN pad server 248
validate record (VR) function in system manager server 497
validating data with defined record structures (system manager server) 497
variable length records (in shared-file server) 260
VDM 555
verify message authentication code (VA) function in PIN pad server 248
virtual DOS machine (OS/2 or Windows) 555
VisualAge C++ web site 706
VisualAge Generator books 706
VisualAge Smalltalk web site 707
VR (validate record) function in system manager server 497

## W

WD (write display) function in PIN pad server 249
wide area communication
  books 704
Windows
  supported releases xxvii
Windows virtual DOS machine 555
WL (write LOG record) function in system manager server 510
WM (write message data) function in object postbox server 585
workgroup identification, update (UI) function in system manager server 491
WR (write to IBM 47xx MSR/E device) function in MSR/E server 214
write display (WD) function in PIN pad server 249
write LOG record (WL) function in system manager server 510
write message data (WM) function in object postbox server 585
write status, check (CH) function in MSR/E server 208
write structured fields in 3270 emulator 655
WT (write to IBM 47xx MSR/E device) function in MSR/E server 215

## X

X25NAT## (X.25 communication server) 94
XN (retrieve user profile by record number (Year-2000)) function in system manager server 479

## Z

Z4 (asynchronous request—asking for TT request) supervisor local function 29
Z5 (asynchronous request—event notification or cancellation) supervisor local function 30
Z6 (switch printer mode) function, FBSI/2 671
ZI (server information) function in SNA server 66
ZN (extended asynchronous event notification or cancellation) supervisor local function 31



Printed in the USA

SC34-6045-00



Spine information:



LANDP® Family

LANDP Programming Reference

Version 6.0