Information Management for z/OS

Client Installation and User's Guide

Version 7.1

SC31-8738-00

Information Management for z/OS

Client Installation and User's Guide

Version 7.1

SC31-8738-00

Tivoli Information Management for z/OS Client Installation and User's Guide

Copyright Notice

© Copyright IBM Corporation 1981, 2001. All rights reserved. May only be used pursuant to a Tivoli Systems Software License Agreement, an IBM Software License Agreement, or Addendum for Tivoli Products to IBM Customer or License Agreement. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without prior written permission of IBM Corporation. IBM Corporation grants you limited permission to make hardcopy or other reproductions of any machine-readable documentation for your own use, provided that each such reproduction shall carry the IBM Corporation copyright notice. No other rights under copyright are granted without prior written permission of IBM Corporation. The document is not intended for production and is furnished "as is" without warranty of any kind. **All warranties on this document are hereby disclaimed, including the warranties of merchantability and fitness for a particular purpose.**

U.S. Government Users Restricted Rights—Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corporation.

Trademarks

The following product names are trademarks of International Business Machines Corporation in the United States, other countries, or both: IBM, the IBM logo, Tivoli, the Tivoli logo, AIX, APPN, CICS, CICS/ESA, DATABASE 2, DB2, DFSMS/MVS, IBMLink, Language Environment, MVS, MVS/ESA, NetView, OpenEdition, Operating System/2, OS/2, OS/2 WARP, OS/390, Presentation Manager, RACF, Redbooks, RISC System/6000, RMF, RS/6000, System/390, Tivoli Enterprise Console, TME 10, VisualAge, VTAM, z/OS.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names mentioned in this document may be trademarks or service marks of others.

Notices

References in this publication to Tivoli Systems or IBM products, programs, or services do not imply that they will be available in all countries in which Tivoli Systems or IBM operates. Any reference to these products, programs, or services is not intended to imply that only Tivoli Systems or IBM products, programs, or services can be used. Subject to valid intellectual property or other legally protectable right of Tivoli Systems or IBM, any functionally equivalent product, program, or service can be used instead of the referenced product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by Tivoli Systems or IBM, are the responsibility of the user. Tivoli Systems or IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, New York 10504-1785, U.S.A.

Programming Interface Information

This publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of Tivoli Information Management for z/OS.

Contents

Preface xxi

Who Should Read This Guide xxi

Prerequisite and Related Documentation xxii

What This Guide Contains xxiii

Contacting Customer Support xxiv

Chapter 1. Client/Server Computing 1

Supported Communication Protocols 2

Clients 2

- HLAPI Functions and Transactions 3
- Client Comparisons 4

Servers 5

- Remote Environment Server (RES) 5
- Multiclient Remote Environment Server (MRES) with APPC 6
- Multiclient Remote Environment Server (MRES) with TCP/IP 8

Java Applications 10

Chapter 2. Choosing a Server 13

RES Conversation Management 13

MRES with APPC Conversation Management 14

MRES with TCP/IP Conversation Management 15

Logical Unit (LU) Considerations 16

Socket Considerations 16

Accounting Considerations 16

- RES 16
- MRES with APPC 17

Performance Considerations 17

MRES with Pre-started API Sessions Considerations 18

MRES with APPC Cataloged Procedure Considerations 19

MRES with TCP/IP Cataloged Procedure Considerations 19

Transaction Logging by a RES and by an MRES Without Pre-started API Sessions 20

Transaction Logging by an MRES with Pre-started API Sessions 21

Security Considerations 21

- Security References 22
- Security for a RES 22
- Security Considerations When Using Pre-started API Sessions 23

Security for an MRES with APPC 23
Security for an MRES with TCP/IP 24

Chapter 3. Configuring and Running a Remote Environment Server (RES) 25

Using a RES 25
RES Configuration Tasks 26
Planning Your RES Configuration 26
Setting Up APPC/MVS 26
Creating a VSAM Data Set for the TP Profile 26
Making an Entry for the RES in the TP Profile Data Set 27
Defining Local LUs and Identifying Partner LUs 28
Defining a Schedule Class 29
Modifying VTAM 30
Defining the Local LU in VTAM 30
Defining the Log-on Mode 30
Defining Links to Clients 31
Defining Security Classes and Profiles 32
Starting and Stopping the APPC Environment 32
Determining Values Clients Need 33

Chapter 4. Configuring and Running a Multiclient Remote Environment Server (MRES) with APPC 35

Using an MRES with APPC 36
MRES Configuration Tasks 37
Planning an MRES with APPC Configuration 38
Defining a Procedure for an MRES with APPC 38
Coding the Parameters for an MRES with APPC 39
Sample MRES Parameters 45
Adding the Data Sets to the APF List 46
Defining an MRES with APPC to APPC/MVS 46
Creating a VSAM Data Set for the Side Information 46
Adding a Side Information Entry for an MRES 47
Defining a Nonscheduled APPC/MVS Logical Unit 48
Defining an MRES with APPC to VTAM 48
Implementing Security 49
Starting and Stopping an MRES with APPC 49
START Command Syntax 49

STOP Command Syntax 50
Determining Values Clients Need. 50

Chapter 5. Configuring and Running a Multiclient Remote Environment Server (MRES) with TCP/IP 53
Using an MRES with TCP/IP 54
MRES Configuration Tasks 56
Planning Your MRES with TCP/IP Configuration. 56
Defining a Procedure for an MRES with TCP/IP 56
Coding the Parameters for an MRES with TCP/IP 58
Sample MRES Parameters. 64
Adding the Data Sets to the APF List 65
Starting and Stopping an MRES with TCP/IP 65
START Command Syntax 65
STOP Command Syntax 66
Determining Values Clients Need. 66

Chapter 6. Using MVS Operator Commands. 67
Displaying Server Address Space Status. 67
Cancelling a Server Address Space 68
Forcing a Server Address Space. 68

Chapter 7. Introduction to the HLAPI/2. 69
A Typical Scenario 69
Server Overview 70
Requester Overview 71
HLAPI/2 C Language Binding. 71
Basic Transaction Flow 72
On the Workstation 72
Communication Link 73
On the Host 73
Back to the Workstation 73
The REXX HLAPI/2 Interface. 73
Client Workstation Requirements for the HLAPI/2 74
Software 74
Hardware 74

Chapter 8. Installing and Configuring HLAPI/2 77
Configuring a Communication Link to a Server 77

Configuring HLAPI/2 for APPC 77
Configuring HLAPI/2 for TCP/IP 81
Preparing to Install HLAPI/2 82
Installing HLAPI/2 on the Workstation 82
Installing HLAPI/2 in Attended Mode from CD-ROM 83
Installing HLAPI/2 on a LAN Server 84
Installing HLAPI/2 in Attended Mode from a LAN Server 85
Installing HLAPI/2 on a Workstation in Unattended Mode 86
Choosing the Appropriate Requester 90
Customizing the HLAPI/2 CONFIG.SYS File 90
Applying HLAPI/2 Maintenance 91
Restoring HLAPI/2 to the Previous Service Level 91
Restoring HLAPI/2 If It Was Installed from the HLAPI/2 CD-ROM 91
Restoring HLAPI/2 If It Was Installed from a LAN Server. 92
Deleting HLAPI/2 from Your Workstation 92
Diagnostic Assistance 93

Chapter 9. HLAPI/2 Profiles, Environment Variables, and Data Logging 95

Profile Syntax 95
System Profile Keywords 96
IDBINBOUNDBUFSIZE. 96
IDBOUTBOUNDBUFSIZE. 96
IDBSHARECMS. 96
System Profile Example. 97
Database Profile Keywords 97
IDBDataLogLevel 98
IDBLogFileSize. 98
IDBLogFileNameActive 98
IDBLogFileNameOld. 98
IDBCharCodePage 98
IDBServCharCodePage 99
IDBSymDestName 99
IDBServerHost 99
IDBServerService 99
Database Profile Example 100
HLAPI/2 OS/2 Environment Variables 101
Profile Override. 101

Profile Search Path . . . 101
IDBDBPATH . . . 101
IDBSMPATH . . . 102
Server Logging . . . 102
Transaction Logging . . . 102
HLAPI/2 Error Logging . . . 103
Chapter 10. The HLAPI/2 Requester . . . 105
Starting the Requester . . . 105
Stopping the Requester . . . 105
Diagnosis of Some Common HLAPI/2 Problems . . . 106
Changing the Profile and Its Effect on Program Operation . . . 106
Establishing a Conversation with the Host . . . 106
Establishing Too Many APPC Conversations . . . 107
Chapter 11. HLAPI/2 Transactions . . . 109
Transaction Operating Modes . . . 109
Synchronous Processing . . . 109
Asynchronous Processing . . . 110
Data Conversion Characteristics . . . 110
Database Profile Parameters . . . 111
Possible Truncation of Mixed SBCS/DBCS Data . . . 111
Differences between HLAPI/2 and HLAPI Transactions . . . 111
Initialize Tivoli Information Management for z/OS (HL01) . . . 111
Terminate Tivoli Information Management for z/OS (HL02) . . . 113
Retrieve Record (HL06) . . . 113
Create Record (HL08) . . . 114
Update Record (HL09) . . . 114
HLAPI/2 Code Pages . . . 114
ASCII SBCS Code Pages . . . 114
ASCII DBCS Code Pages . . . 115
EBCDIC SBCS Code Pages . . . 115
EBCDIC MIX Code Pages . . . 115
EBCDIC PURE DBCS Code Pages . . . 115
Chapter 12. Tips for Writing a HLAPI/2 Application . . . 117
Installation and Setup Summary for HLAPI/2 Sample Applications . . . 118
Chapter 13. HLAPI/2 C Language Application Program . . . 121
Allocating HICAs and PDBs . . . 121

Including the Header File in Your Program . . . 121
Allocating and Initializing a HICA . . . 122
Allocating and Initializing a PDB . . . 122
Binding Prototypes . . . 123
IDBTransactionSubmit . . . 124
IDBTransactionStatus . . . 124
Linking Your Program . . . 126
Sample HLAPI/2 C Program . . . 126
Steps Required to Run the HLAPI/2 C Sample Program . . . 127
HLAPI/2 Header Code . . . 127
Chapter 14. REXX HLAPI/2 Interface . . . 131
REXX HLAPI/2 Installation and Setup . . . 131
Prerequisite Setup . . . 132
Registering the REXX HLAPI/2 DLL . . . 132
REXX HLAPI/2 Interface Calls . . . 132
Deregistering the REXX HLAPI/2 DLL . . . 133
Differences between the REXX HLAPI/2 and the HLAPI/2 . . . 133
Differences between the REXX HLAPI/2 and the HLAPI/REXX . . . 133
REXX Reserved Variables . . . 133
Sample REXX HLAPI/2 Program . . . 134
Steps Required to Run the REXX HLAPI/2 Sample Program . . . 134
Transaction List . . . 135
Chapter 15. Introduction to the HLAPI/NT . . . 137
A Typical Scenario . . . 137
Server Overview . . . 138
Requester Overview . . . 138
HLAPI/NT C Language Binding . . . 139
Basic Transaction Flow . . . 139
On the Workstation . . . 140
Communication Link . . . 140
On the Host . . . 141
Back to the Workstation . . . 141
Client Workstation Requirements for the HLAPI/NT . . . 141
Software . . . 141
Hardware . . . 142

Chapter 16. Installing and Configuring HLAPI/NT 143

Configuring a Communication Link to a Server 143
Configuring HLAPI/NT for TCP/IP 143
Configuring HLAPI/NT for APPC 144
Preparing to Install HLAPI/NT 145
Installing HLAPI/NT in Attended Mode from CD-ROM. 145
Installing HLAPI/NT onto a Network Drive 146
Installing HLAPI/NT in Attended Mode from a Network Drive 147
Installing HLAPI/NT in Unattended Mode 148
HLAPI/NT Response File Keywords 148
Applying HLAPI/NT Maintenance 151
Deleting HLAPI/NT from a Workstation 151

Chapter 17. HLAPI/NT Profiles, Environment Variables, and Data Logging 153

Profile Syntax 153
System Profile Keywords. 154
IDBINBOUNDBUFSIZE. 154
IDBOUTBOUNDBUFSIZE 154
IDBSHARECMS. 154
System Profile Example. 155
Database Profile Keywords 155
IDBDataLogLevel 156
IDBLogFileSize. 156
IDBLogFileNameActive 156
IDBLogFileNameOld. 156
IDBCharCodeSet 156
IDBServCharCodeSet 157
IDBServerHost 157
IDBServerService 157
IDBSymDestName 157
Database Profile Example 158
Environment Variables. 158
Profile Override. 159
Profile Search Path 159
IDBDBPATH. 159
IDBSMPATH 160
Server Logging 160

Transaction Logging 160
HLAPI/NT Error Logging 161

Chapter 18. The HLAPI/NT Requester 163

Starting the Requester 163
Stopping the Requester 163
Diagnosis of Some Common HLAPI/NT Problems 164
Changing the Profile and Its Effect on Program Operation 164
Data Conversion Problems 164
Establishing a Conversation with the Host 164
Establishing Too Many APPC Conversations 165

Chapter 19. HLAPI/NT Transactions 167

Transaction Operating Modes 167
Synchronous Processing 167
Asynchronous Processing 168
Data Conversion Characteristics 168
Database Profile Parameters 169
Possible Truncation of Mixed SBCS/DBCS Data 169
Differences between HLAPI/NT and HLAPI Transactions 170
Initialize Tivoli Information Management for z/OS (HL01) 170
Terminate Tivoli Information Management for z/OS (HL02) 172
Retrieve Record (HL06) 172
Create Record (HL08) 172
Update Record (HL09) 172

Chapter 20. Tips for Writing a HLAPI/NT Application 175

Installation and Setup Summary for HLAPI/NT Sample Applications 176

Chapter 21. HLAPI/NT C Language Application Program 179

Allocating HICAs and PDBs 179
Including the Header File in Your Program 179
Allocating and Initializing a HICA 180
Allocating and Initializing a PDB 180
Binding Prototypes 181
IDBTransactionSubmit 182
IDBTransactionStatus 182
Linking Your Program 184
Sample HLAPI/NT C Program 184

Steps Required to Run the HLAPI/NT C Sample Program . . . 184

Chapter 22. Introduction to HLAPI/CICS . . . 187
HLAPI/CICS Overview . . . 187
Server Overview . . . 188
HLAPI/CICS Basic Transaction Flow . . . 188
Requirements. . . . 190
Software . . . 190
Hardware . . . 190

Chapter 23. Installing HLAPI/CICS and Customizing CICS/ESA . . . 191
Installing HLAPI/CICS . . . 191
Customizing CICS/ESA for HLAPI/CICS . . . 191
Customizing the System Initialization Table (DFHSIT). . . . 192
Customizing the Destination Control Table (DFHDCT) . . . 193
Customizing the Shut-Down Program Load Table (DFHPLT) . . . 194
Customizing the Startup JCL . . . 195
Customizing the CICS/ESA System Definition Data Set - JCL . . . 195
Customizing the CICS/ESA Systems Definition Data Set - Online . . . 196
Program Entries. . . . 196
Transaction Entries . . . 199
Connection Entries . . . 201
Session Entries . . . 202
Partner Entries. . . . 204
Add the Groups to a List . . . 205

Chapter 24. HLAPI/CICS Transaction Coding . . . 209
Linking to the HLAPI/CICS . . . 209
Control PDBs for HLAPI Transactions. . . . 210
CICS_User_ID PDB . . . 210
CICS_Partner_ID PDB . . . 210
CICS_CM_Time_Out_Value PDB . . . 210
CICS_Inter_Time_Out_Value PDB . . . 211

Chapter 25. Running the Sample CICS Application . . . 213
Sample Programs. . . . 213
Installing the Sample Programs . . . 214
Defining the Programs and Transactions to CICS . . . 215
Starting the Sample Application . . . 216
Entering the BLMM Transaction . . . 216

HL01 - Starting the Session . . . 217
Modifying the HL01 Panel . . . 218
HL01 Output - Main Menu . . . 219
HL08 - Creating a Record . . . 219
HL08 Output . . . 220
HL06 - Retrieving a Record. . . . 221
HL06 Output . . . 221
HL13 - Deleting a Tivoli Information Management for z/OS Record . . . 222
HL13 Input . . . 222
HL13 Output . . . 222
HL02 - Ending the Logical Session . . . 223
HL02 Input . . . 223
Ending the Sample Application . . . 223
Sample Closing Screen . . . 224
Running Multiple Environments. . . . 224

Chapter 26. Introduction to HLAPI/UNIX . . . 225

A Typical Scenario . . . 226
Server Overview . . . 226
Requester Overview. . . . 227
Client Interface Overview . . . 228
Communication Overview . . . 228
Basic Transaction Flow . . . 229
On the UNIX host running the client application program . . . 230
On the UNIX host running the requester . . . 230
On the MVS host running the server and Tivoli Information Management for z/OS . . . 230
On the UNIX host running the requester . . . 230
On the UNIX host running the client application program . . . 230
HLAPI/UNIX Configuration Considerations. . . . 231
Resources Needed for the Client Interface . . . 232
Resources Needed for the Requester . . . 233
Hardware and Software Requirements . . . 234
Hardware for HLAPI/UNIX. . . . 234
Software for HLAPI/UNIX . . . 234

Chapter 27. Installing and Setting Up HLAPI/UNIX . . . 237

Planning a HLAPI/UNIX Configuration . . . 237
Setting Up HLAPI/AIX . . . 238

Distributing HLAPI/AIX from a CD-ROM 238
Distributing HLAPI/AIX from a File System 238
Installing HLAPI/AIX on the RS/6000 System 239
Installing Options from a CD-ROM 239
Installing Options from a File System 240
Setting Up HLAPI/HP and HLAPI/Solaris 240
Distributing HLAPI/HP and HLAPI/Solaris from CD-ROM 240
Installing HLAPI/HP and HLAPI/Solaris 241
Configuring HLAPI/UNIX and Associated Software 241
Configuring HLAPI/AIX for APPC 242
Control Point Profile 242
Defining Side Information 243
Verifying Configuration 244
Starting and Stopping APPC 245
Determining Values 245
Configuring HLAPI/UNIX for TCP/IP 245
Defining the Client Interface to Requester Communication Link 246
Updating /etc/services on a Requester Host 246
Updating /etc/services and /etc/hosts on a Client Host 247
Removing HLAPI/UNIX Options 248

Chapter 28. HLAPI/UNIX Profiles, Environment Variables, and Data Logging 251

Profile Syntax 251
System Profile 252
IDBINBOUNDBUFSIZE 252
IDBMAXCMS 253
IDBOUTBOUNDBUFSIZE 253
IDBSERVICENAME 253
IDBSHARECMS 253
IDBTIMEOUT 254
System Profile Example 255
Database Profile 255
IDBCHARCODESET 255
IDBDATALOGLEVEL 256
IDBIDLECLIENTTIMEOUT 256
IDBLOGFILENAMEACTIVE 256
IDBLOGFILENAMEOLD 257
IDBLOGFILESIZE 257

IDBREQUESTERHOST . . . 257
IDBREQUESTERSERVICE . . . 257
IDBSERVCHARCODESET . . . 258
IDBSERVERHOST . . . 258
IDBSERVERSERVICE . . . 258
IDBSYMDESTNAME . . . 259
Database Profile Example . . . 259
Environment Variables . . . 260
IDBDATALOGLEVEL . . . 260
IDBREQUESTERHOST . . . 260
IDBREQUESTERSERVICE . . . 260
IDBDBPATH . . . 260
IDBSMPATH . . . 261
Transaction Logging . . . 261
Transaction Logging by a Client Interface . . . 261
Error Probe Logging by a Requester or Client Interface . . . 262
Chapter 29. The HLAPI/UNIX Requester . . . 263
Starting the Requester Manually . . . 263
Starting the Requester Automatically . . . 263
Stopping a Requester . . . 264
Diagnosing Some Common Problems . . . 264
Chapter 30. HLAPI/UNIX Transactions . . . 267
Validation of the Calling Process . . . 267
Transaction Processing Modes . . . 267
Synchronous Processing . . . 267
Asynchronous Processing . . . 268
Transaction Concurrency Limitations . . . 268
Data Conversion Characteristics . . . 269
Special DBCS Considerations . . . 269
Developing HLAPI/UNIX Client Applications . . . 270
Including the HLAPI/UNIX Header File idbh.h . . . 271
Including the HLAPI/UNIX Header File idbech.h . . . 272
Overview of HICA and PDB Data Structures . . . 272
Allocating and Initializing a HICA structure . . . 273
Allocating and Initializing a PDB Structure . . . 274
HLAPI/UNIX Functions . . . 275
IDBTransactionSubmit() . . . 275

IDBTransactionStatus() 276
Using HLAPI/UNIX Functions in a Transaction Sequence 278
Initialize Tivoli Information Management for z/OS (HL01). 278
Terminate Tivoli Information Management for z/OS (HL02). 279
Retrieve Record (HL06) 279
Create Record (HL08) 280
Update Record (HL09) 280
Linking Your Application to HLAPI/UNIX Services. 280
Planning Your HLAPI/UNIX Application 281
Converting HLAPI Programs to HLAPI/UNIX Programs 282
Using the REXX HLAPI/AIX Interface 283
REXX HLAPI/AIX Installation and Setup 283
Invoking REXX HLAPI/AIX. 283
REXX Reserved Variables 285
Other Considerations 285
REXX HLAPI/AIX Sample REXX Program 286
Sample Program BLMYRXSA 286

Chapter 31. Introduction to HLAPI/USS. 291

Server Overview 291
Requester Overview. 292
Client Interface Overview 292
Communication Overview 293
Basic Transaction Flow 293
On the USS host running the client application program. 294
On the USS host running the requester 294
On the MVS host running the MRES with TCP/IP server and Tivoli Information Management for z/OS 295
On the USS host running the requester 295
On the USS host running the client application program. 295
HLAPI/USS Configuration Considerations 295
Resources Needed for the Client Interface 296
Resources Needed for the Requester 297
Hardware and Software Requirements 297
Hardware for HLAPI/USS 298
Software for HLAPI/USS 298

Chapter 32. Installing and Setting Up HLAPI/USS. 299

Planning a HLAPI/USS Configuration 299

Configuring HLAPI/USS and Associated Software 299
Configuring HLAPI/USS for TCP/IP 299
Defining the Client Interface to Requester Communication Link 300
Updating /etc/services on a Requester Host 300
Updating /etc/services on a Client Host 301

Chapter 33. HLAPI/USS Profiles, Environment Variables, and Data Logging 303

Profile Syntax 303
System Profile 304
IDBINBOUNDBUFSIZE 304
IDBMAXCMS 305
IDBOUTBOUNDBUFSIZE 305
IDBSERVICENAME 305
IDBSHARECMS 305
IDBTIMEOUT 306
System Profile Example 307
Database Profile 307
IDBDATALOGLEVEL 307
IDBIDLECLIENTTIMEOUT 308
IDBLOGFILENAMEACTIVE 308
IDBLOGFILENAMEOLD 308
IDBLOGFILESIZE 308
IDBREQUESTERHOST 309
IDBREQUESTERSERVICE 309
IDBSERVERHOST 309
IDBSERVERSERVICE 310
Database Profile Example 310
Environment Variables 310
IDBDATALOGLEVEL 310
IDBREQUESTERHOST 311
IDBREQUESTERSERVICE 311
BLMDBPATH 311
BLMSMPATH 311
Transaction Logging 312
Transaction Logging by a Client Interface 312
Error Probe Logging by a Requester or Client Interface 313

Chapter 34. The HLAPI/USS Requester 315

Starting the Requester from the Shell . . . 315
Stopping a Requester from the Shell . . . 316
Starting a Requester by JCL . . . 316
Diagnosing Some Common Problems. . . . 316

Chapter 35. HLAPI/USS Transactions . . . 319

Validation of the Calling Process . . . 319
Transaction Processing Modes . . . 319
Transaction Concurrency Limitations . . . 319
Developing HLAPI/USS Client Applications . . . 320
Including the HLAPI/USS Header File blmh.h . . . 320
Including the HLAPI/USS Header File blmech.h . . . 321
Overview of HICA and PDB Data Structures . . . 321
Allocating and Initializing a HICA structure . . . 322
Allocating and Initializing a PDB Structure . . . 322
HLAPI/USS Function . . . 324
IDBTransactionSubmit() . . . 324
Using the HLAPI/USS Function in a Transaction Sequence . . . 325
Initialize Tivoli Information Management for z/OS (HL01). . . . 325
Terminate Tivoli Information Management for z/OS (HL02). . . . 326
Retrieve Record (HL06) . . . 326
Create Record (HL08) . . . 327
Update Record (HL09) . . . 327
Compiling and Linking Your Application to HLAPI/USS Services . . . 327
Planning Your HLAPI/USS Application . . . 328
Converting HLAPI Programs to HLAPI/USS Programs . . . 328
Using the REXX HLAPI/USS Interface . . . 329
REXX HLAPI/USS Installation and Setup . . . 330
Invoking REXX HLAPI/USS. . . . 330
REXX Reserved Variables . . . 332
Other Considerations . . . 332
REXX HLAPI/USS Sample REXX Program . . . 333
Sample Program blmyrxsa . . . 333

Appendix A. Components of Tivoli Information Management for z/OS Clients . . . 339

Components of HLAPI/2 . . . 339
Files on the Workstation . . . 339
Files on the LAN Server . . . 340

Components of HLAPI/CICS 341
Components of HLAPI/NT 342
Files on the Workstation 342
Files on the Network Server 347
Components of HLAPI/AIX 348
Requester Option 348
Client Interface Option 348
Components of HLAPI/HP 349
Requester Option 349
Client Interface Option 349
Other Files 350
Components of HLAPI/Solaris 350
Requester Option 350
Client Interface Option 351
Other Files 351
Components of HLAPI/USS 352
Directories 352
Files 352
Symbolic links 352

Appendix B. Tivoli Information Management for z/OS Java Wrappers (HLAPI for Java) 353
Class Hicao 354
Class Pdbo 361
Class Blmyjwc 363

Appendix C. HLAPI Service Call Return Codes 367
Return Codes 367

Appendix D. Relating Publications to Specific Tasks 369
Typical Tasks 369

Appendix E. Tivoli Information Management for z/OS Courses 373
Education Offerings 373
United States 373
United Kingdom 373

Appendix F. Where to Find More Information 375
The Tivoli Information Management for z/OS Library 375

Index 379

Preface

Tivoli® Information Management for z/OS extends your ability to gather, organize, and locate information about your company's data processing installation. The main purpose of the Tivoli Information Management for z/OS database is to hold problem, change, configuration, and user-defined data for your company. You can do these things from a remote environment workstation.

The remote environments supported by Tivoli Information Management for z/OS are Advanced Interactive Executive (AIX®), Customer Information Control System (CICS®), HP-UX, Operating System/2® (OS/2®), Sun Solaris, and Microsoft® Windows NT®. The communication protocols supported are advanced program-to-program communication (APPC) and Transmission Control Protocol/Internet Protocol (TCP/IP).

Please Note

Throughout this guide, the term UNIX® refers to the AIX, HP-UX, and Sun Solaris clients, unless otherwise specified. The term HLAPI/UNIX refers to HLAPI/AIX, HLAPI/HP, and HLAPI/Solaris.

In addition to those workstation environments, remote and local access to Tivoli Information Management for z/OS can be provided from an application program that runs on OS/390® UNIX System Services. Such user application programs can be local, running under OS/390 UNIX System Services on the same MVS™ host as Tivoli Information Management for z/OS, or they can be run on a remote MVS host. These local or remote environment user application programs can be thought of as the clients to the Tivoli Information Management for z/OS server.

This guide describes the servers Tivoli Information Management for z/OS offers to support remote clients and tells the system or application programmer how to configure them. This guide also describes the clients supported by Tivoli Information Management for z/OS and tells the system or application programmer how to install and set them up to communicate with the MVS host.

There may be references in this publication to versions of Tivoli Information Management for z/OS's predecessor products. For example:

- TME 10™ Information/Management Version 1.1
- Information/Management Version 6.3, Version 6.2, Version 6.1
- Tivoli Service Desk for OS/390 Version 1.2

Who Should Read This Guide

This guide is intended for system programmers and application programmers. It assumes that the system programmer installing the servers on MVS is familiar with MVS and the communication protocols to be used. It also assumes that the application programmer in each of the remote environments understands the client environment being used. For example, if you choose to access Tivoli Information Management for z/OS from OS/2, this guide assumes that the system programmer understands how to use OS/2 functions, commands,

and communications protocols. This guide also describes the client Application Programming Interfaces (APIs), and is of value to anyone who will be installing these APIs.

Prerequisite and Related Documentation

The library for Tivoli Information Management for z/OS Version 7.1 consists of these publications. For a description of each, see "The Tivoli Information Management for z/OS Library" on page 375.

Tivoli Information Management for z/OS Application Program Interface Guide, SC31-8737-00

Tivoli Information Management for z/OS Client Installation and User's Guide, SC31-8738-00

Tivoli Information Management for z/OS Data Reporting User's Guide, SC31-8739-00

Tivoli Information Management for z/OS Desktop User's Guide, SC31-8740-00

Tivoli Information Management for z/OS Diagnosis Guide, GC31-8741-00

Tivoli Information Management for z/OS Guide to Integrating with Tivoli Applications, SC31-8744-00

Tivoli Information Management for z/OS Integration Facility Guide, SC31-8745-00

Tivoli Information Management for z/OS Licensed Program Specification, GC31-8746-00

Tivoli Information Management for z/OS Master Index, Glossary, and Bibliography, SC31-8747-00

Tivoli Information Management for z/OS Messages and Codes, GC31-8748-00

Tivoli Information Management for z/OS Operation and Maintenance Reference, SC31-8749-00

Tivoli Information Management for z/OS Panel Modification Facility Guide, SC31-8750-00

Tivoli Information Management for z/OS Planning and Installation Guide and Reference, GC31-8751-00

Tivoli Information Management for z/OS Problem, Change, and Configuration Management, SC31-8752-00

Tivoli Information Management for z/OS Program Administration Guide and Reference, SC31-8753-00

Tivoli Information Management for z/OS Reference Summary, SC31-8754-00

Tivoli Information Management for z/OS Terminal Simulator Guide and Reference, SC31-8755-00

Tivoli Information Management for z/OS User's Guide , SC31-8756-00

Tivoli Information Management for z/OS World Wide Web Interface Guide, SC31-8757-00

Note: Tivoli is in the process of changing product names. Products referenced in this manual may still be available under their old names (for example, TME 10 Enterprise Console instead of Tivoli Enterprise Console®).

What This Guide Contains

"Client/Server Computing" on page 1 introduces the Tivoli Information Management for z/OS clients and servers. It lists the HLAPI transactions the clients support and illustrates some possible network configurations.

"Choosing a Server" on page 13 compares the Tivoli Information Management for z/OS servers on several key characteristics. Read this chapter when you are planning a client/server configuration.

"Configuring and Running a Remote Environment Server (RES)" on page 25 provides instructions for setting up and running a Remote Environment Server (RES). Read this chapter to become familiar with the setup of Tivoli Information Management for z/OS's server that supports one client per MVS address space.

"Configuring and Running a Multiclient Remote Environment Server (MRES) with APPC" on page 35 provides instructions for setting up and running a Multiclient Remote Environment Server with APPC (MRES with APPC). Read this chapter to become familiar with the setup of Tivoli Information Management for z/OS's server that uses APPC and can support multiple clients per MVS address space.

"Configuring and Running a Multiclient Remote Environment Server (MRES) with TCP/IP" on page 53 provides instructions for setting up and running a Multiclient Remote Environment Server with TCP/IP (MRES with TCP/IP). Read this chapter to become familiar with the setup of Tivoli Information Management for z/OS's server that uses TCP/IP and can support multiple clients per MVS address space.

"Using MVS Operator Commands" on page 67 lists some useful support commands.

Information on the clients is grouped into sections, with several chapters providing information specific to each client:

- Information on the HLAPI/2 client begins with "Introduction to the HLAPI/2" on page 69.
- Information on the client for Windows NT begins with "Introduction to the HLAPI/NT" on page 137.
- Information on the CICS client begins with "Introduction to HLAPI/CICS" on page 187.
- Information on the HLAPI/UNIX clients begins with "Introduction to HLAPI/UNIX" on page 225.
- Information on the HLAPI/USS client begins with "Introduction to HLAPI/USS" on page 291.

"Components of Tivoli Information Management for z/OS Clients" on page 339 lists the components of the clients.

"Tivoli Information Management for z/OS Java Wrappers (HLAPI for Java)" on page 353 is a listing of Java™ classes that are provided to allow programmers to more easily write HLAPI applications using Java. These classes are based on the existing HLAPI programming model, but simplify that model by providing some of the common programming functions.

"HLAPI Service Call Return Codes" on page 367 lists the HLAPI Service Call Return Codes.

Contacting Customer Support

For support inside the United States, for this or any other Tivoli product, contact Tivoli Customer Support in one of the following ways:

- Send e-mail to **support@tivoli.com**
- Call 1-800-TIVOLI8
- Navigate our Web site at **http://www.support.tivoli.com**

For support outside the United States, refer to your Customer Support Handbook for phone numbers in your country. The Customer Support Handbook is available online at **http://www.support.tivoli.com**.

When you contact Tivoli Customer Support, be prepared to provide identification information for your company so that support personnel can assist you more readily.

The latest downloads and fixes can be obtained at **http://www.tivoli.com/infoman**.

1

Client/Server Computing

Tivoli Information Management for z/OS extends your ability to gather, organize, and locate information about your company's data processing installation. The main purpose of the Tivoli Information Management for z/OS database is to hold problem, change, configuration, and user-defined data for your company. If your Tivoli Information Management for z/OS system is part of a communications network, you may want to access Tivoli Information Management for z/OS data from user application programs running on other platforms in remote environments, such as OS/2 or AIX. Tivoli Information Management for z/OS provides interfaces for OS/2, UNIX, CICS, Windows NT, and OS/390 UNIX System Services that enable application programs on those platforms to do just that. These interfaces are called *clients* because they request services from a network.

Tivoli Information Management for z/OS also provides interfaces on MVS to receive client requests and pass them along to Tivoli Information Management for z/OS for processing. These interfaces are called *servers* because they respond to requests for service from clients.

In order for an application program running on a different platform, or running on MVS under CICS or OS/390 UNIX System Services, to retrieve data from Tivoli Information Management for z/OS, a communication link between Tivoli Information Management for z/OS and the application program has to be established. Tivoli Information Management for z/OS clients initiate the establishment of the communication link. They send identifying information to MVS to verify their authorization to use the communication link and the Tivoli Information Management for z/OS database.

Data from application programs in remote environments such as OS/2 or UNIX or Windows NT also must be translated into a code page that MVS understands. Then, when Tivoli Information Management for z/OS returns data to a remote environment, the data has to be translated back into the code page that the remote environment understands. This code page translation is done by the Tivoli Information Management for z/OS client.

On the Tivoli Information Management for z/OS end of the communication link, the application program has a choice of servers that support access to Tivoli Information Management for z/OS data through the High-Level Application Program Interface (HLAPI). The servers are MVS-based programs that reside on the host system and provide a way for a remote application program to communicate with a Tivoli Information Management for z/OS database.

The Tivoli Information Management for z/OS clients for OS/2, UNIX, Windows NT, and OS/390 UNIX System Services consist of two parts: a client interface and a requester. A *requester* is a transaction program that runs on the client platform; it is the counterpart to the server on the host. A client's requester must be up and running before an application program can request services from a Tivoli Information Management for z/OS server.

This chapter introduces the Tivoli Information Management for z/OS clients and servers.

Supported Communication Protocols

Tivoli Information Management for z/OS provides servers that support Advanced Program-to-Program Communication (APPC) and Transmission Control Protocol/Internet Protocol (TCP/IP) communication protocols. The OS/2, AIX, CICS, and Windows NT clients all support communication with a Tivoli Information Management for z/OS server using APPC protocol. The OS/2, UNIX, Windows NT, and OS/390 UNIX System Services clients also support communication with a Tivoli Information Management for z/OS server using TCP/IP. See “Choosing a Server” on page 13 for a comparison of the servers.

Clients

Like the HLAPI, each client is a transaction-based application programming interface. User application programs interact with Tivoli Information Management for z/OS from a remote environment in basically the same way as they do from MVS using the HLAPI. The client environments offer a subset of HLAPI transactions, which are listed in Table 1 and described in the *Tivoli Information Management for z/OS Application Program Interface Guide*.

The user application communicates with the Tivoli Information Management for z/OS system by creating a high-level application communication area (HICA) and its related parameter data blocks (PDBs). The HICA must include some client-specific PDBs to identify its source. The user application then submits the HICA transaction by making client program service calls. Return and reason codes are returned in the transaction HICA.

For those clients that require language bindings, such as the OS/2 client, these are provided to assist you in writing application programs that request services from a Tivoli Information Management for z/OS server.

Three of the remote clients (OS/2, AIX, and USS) also provide a REXX interface which allows you to write REXX application programs. The interfaces are the same as HLAPI/REXX, which is described in *Tivoli Information Management for z/OS Application Program Interface Guide*.

Clients are provided for the following remote environments:

- AIX

  This client is referred to as the HLAPI/AIX. It can communicate with a RES or an MRES using the APPC protocol, or an MRES using the TCP/IP protocol. This client consists of two parts, a client interface and a requester. The requester can be run on the same client platform as the client interface or on a different platform. The client interface can also use a requester running on a different UNIX platform.

- CICS

  This client is referred to as the HLAPI/CICS. It can communicate with a RES or an MRES using the APPC protocol.

- HP-UX

  This client is referred to as the HLAPI/HP. It can communicate with an MRES using the TCP/IP protocol. This client consists of two parts, a client interface and a requester. The

requester can be run on the same client platform as the client interface or on a different platform. The client interface can also use a requester running on a different UNIX platform.

- OS/2

  This client is referred to as the HLAPI/2. It can communicate with a RES or an MRES using the APPC protocol, or an MRES using the TCP/IP protocol. This client consists of two parts, a client interface and a requester. The requester and client interface must run on the same host.

- Sun Solaris

  This client is referred to as the HLAPI/Solaris. It can communicate with an MRES using the TCP/IP protocol. This client consists of two parts, a client interface and a requester. The requester can be run on the same client platform as the client interface or on a different platform. The client interface can also use a requester running on a different UNIX platform.

- Windows NT

  This client is referred to as the HLAPI/NT. It can communicate with a RES or an MRES using the APPC protocol, or an MRES using the TCP/IP protocol. This client consists of two parts, a client interface and a requester. The requester and client interface must run on the same host.

- OS/390 UNIX System Services (USS)

  This client is referred to as the HLAPI/USS. It can communicate with an MRES using the TCP/IP protocol. This client consists of two parts, a client interface and a requester. The requester can be run under OS/390 UNIX System Services on the same MVS host as the client interface or on a different MVS host. The client interface can also use a requester running under OS/390 UNIX System Services on a different MVS host.

HLAPI Functions and Transactions

Table 1. HLAPI Functions and Transaction Numbers Supported by Clients

HLAPI Function Supported by Clients	Transaction Number	Transaction Type
Initialize Tivoli Information Management for z/OS	HL01	Environment Control
Terminate Tivoli Information Management for z/OS	HL02	Environment Control
Obtain External Record ID	HL03	Interface Service
Check Out Record	HL04	Interface Service
Check In Record	HL05	Interface Service
Retrieve Record	HL06	Database Access
Reserved	HL07	—
Create Record	HL08	Database Access
Update Record	HL09	Database Access
Change Record Approval	HL10	Interface Service
Record Inquiry	HL11	Database Access
Add Record Relations	HL12	Database Access
Delete Record	HL13	Database Access

Table 1. HLAPI Functions and Transaction Numbers Supported by Clients (continued)

HLAPI Function Supported by Clients	Transaction Number	Transaction Type
Start User TSP or TSX	HL14	Interface Service
Get Data Model	HL31	Database Access

Client Comparisons

Table 2 compares the clients on the following characteristics:

- The communication protocols they support
- Which servers they can be served by
- Which language bindings are shipped with Tivoli Information Management for z/OS for each client
- Whether they require security verification
- Which client control PDBs are required on HL01 transactions
- Other characteristics

Table 2. Comparison of Tivoli Information Management for z/OS HLAPI Clients

Client	Protocol	Served by	Language Bindings	Verify	Control PDBs	Other
HLAPI/AIX	TCP/IP APPC/APPN	RES MRES with APPC MRES with TCP/IP	C Java REXX	YES	DATABASE_PROFILE SECURITY_ID PASSWORD	Client and requester can run on separate machines. A conversation can carry a maximum of 10 logical sessions.
HLAPI/CICS	APPC/APPN	RES MRES with APPC	Sample requires VS COBOL II. See "Other" column.	NO	CICS_User_ID CICS_Partner_ID CICS_CM_Time_Out_Value CICS_Inter_Time_Out_Value	Any language supported by both CICS and Tivoli Information Management for z/OS .
HLAPI/HP	TCP/IP	MRES with TCP/IP	C Java	YES	DATABASE_PROFILE SECURITY_ID PASSWORD	Client and requester can run on separate machines. A conversation can carry a maximum of 10 logical sessions.
HLAPI/2	TCP/IP APPC/APPN	RES MRES with APPC MRES with TCP/IP	C Java REXX	YES	DATABASE_PROFILE SECURITY_ID PASSWORD	Client and requester must run on the same machine.
HLAPI/Solaris	TCP/IP	MRES with TCP/IP	C Java	YES	DATABASE_PROFILE SECURITY_ID PASSWORD	Client and requester can run on separate machines. A conversation can carry a maximum of 10 logical sessions.

Table 2. Comparison of Tivoli Information Management for z/OS HLAPI Clients (continued)

Client	Protocol	Served by	Language Bindings	Verify	Control PDBs	Other
HLAPI/NT	TCP/IP APPC/APPN	RES MRES with APPC MRES with TCP/IP	C Java	YES	DATABASE_PROFILE SECURITY_ID PASSWORD	Client and requester must run on the same machine. A conversation can carry a maximum of 10 logical sessions.
HLAPI/USS	TCP/IP	MRES with TCP/IP	C REXX	YES	DATABASE_PROFILE SECURITY_ID PASSWORD	Client and requester can run on separate hosts. A conversation can carry a maximum of 10 logical sessions.

Servers

Tivoli Information Management for z/OS servers are MVS/ESA™ programs that handle all communication between a client and any Tivoli Information Management for z/OS databases that reside on the MVS system where the server is installed. The server cannot be accessed directly from a user application program. The application must use one of the Tivoli Information Management for z/OS clients to access a Tivoli Information Management for z/OS server.

A Tivoli Information Management for z/OS server must be set up on every MVS/ESA machine running a Tivoli Information Management for z/OS database that an application from a client needs to access.

You can choose between Tivoli Information Management for z/OS servers that can:

- Serve a single client application program at a time using APPC protocol. The application program can start multiple Tivoli Information Management for z/OS logical sessions on one APPC conversation. This server is called a Remote Environment Server (RES). See "Remote Environment Server (RES)" for an overview of the RES.
- Serve several client application programs concurrently using APPC protocol. Each application program can start multiple Tivoli Information Management for z/OS logical sessions on its APPC conversation. This server is called a Multiclient Remote Environment Server with APPC (MRES with APPC). See "Multiclient Remote Environment Server (MRES) with APPC" on page 6 for an overview of the MRES.
- Serve multiple client application programs concurrently using TCP/IP communication protocol. Each application program can start multiple Tivoli Information Management for z/OS logical sessions on its TCP/IP conversation. This server is called a Multiclient Remote Environment Server with TCP/IP (MRES with TCP/IP). See "Multiclient Remote Environment Server (MRES) with TCP/IP" on page 8 for an overview of the MRES with TCP/IP.

Remote Environment Server (RES)

One of the means by which Tivoli Information Management for z/OS supports remote client access is via the Remote Environment Server (RES). The RES runs as a transaction program (TP) that is started by APPC/MVS 4.2 (or later) when a connection request (an *allocate* verb) is received from a Tivoli Information Management for z/OS client. For each

connection request, the APPC scheduler starts a new address space with a RES dedicated to servicing that client's HLAPI transactions. Each RES can access only one BLX-Service Provider (BLX-SP).

A RES must be associated with a logical unit (LU) that is defined as scheduled (*sched*) to APPC/MVS. The minimum software required to define a RES is:

- OS/390 Version 2.5, or a later version

Figure 1 illustrates a client/server configuration that could be achieved using the RES.

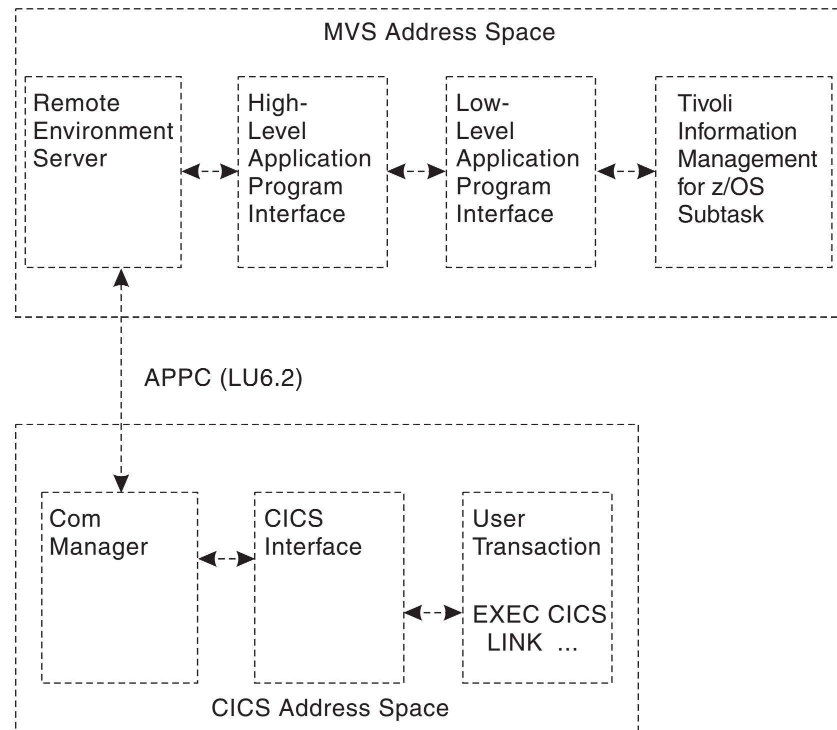


Figure 1. Client/Server Configuration with RESs

The HLAPI/AIX, HLAPI/CICS, HLAPI/2, and HLAPI/NT clients can communicate with a RES.

See "Configuring and Running a Remote Environment Server (RES)" on page 25 for information on installing and running a RES.

Multiclient Remote Environment Server (MRES) with APPC

This server is the Multiclient Remote Environment Server with APPC (MRES with APPC). The address space for MRES with APPC runs as a task started by the MVS operator. Unlike the RES, the MRES with APPC bypasses the APPC scheduler and receives the connection request (an *allocate* verb) directly from a Tivoli Information Management for z/OS client. This allows a single address space for MRES with APPC to receive and process transactions from multiple Tivoli Information Management for z/OS clients concurrently. It also removes the burden of having a separate address space for each client connection. Address spaces for both RES and MRES with APPC can be active on an MVS machine at the same time. Each MRES with APPC can access only one BLX-SP.

An MRES with APPC requires an LU defined to APPC/MVS as nonscheduled (*Nosched*). The minimum software required to define an MRES with APPC is:

- OS/390 Version 2.5, or a later version

Figure 2 illustrates a client/server configuration that could be achieved using an MRES with APPC.

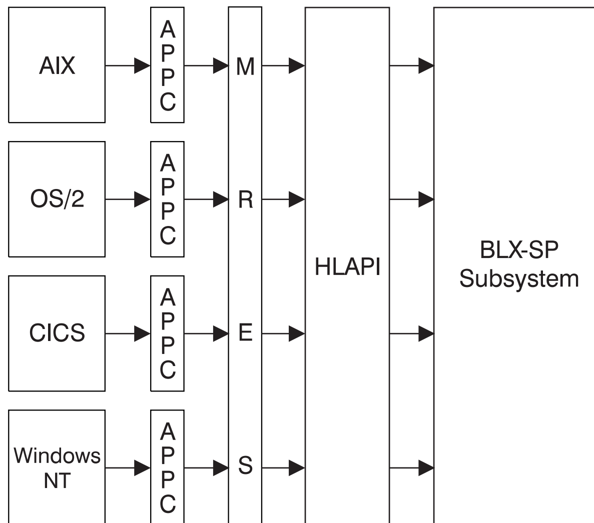


Figure 2. Client/Server Configuration Using an MRES with APPC

The HLAPI/AIX, HLAPI/CICS, HLAPI/2, and HLAPI/NT clients can communicate with an MRES with APPC.

See "Configuring and Running a Multiclient Remote Environment Server (MRES) with APPC" on page 35 for information on installing and starting an MRES with APPC.

Figure 3 on page 8 illustrates a client/server configuration that could be achieved using RESs and an MRES with APPC.

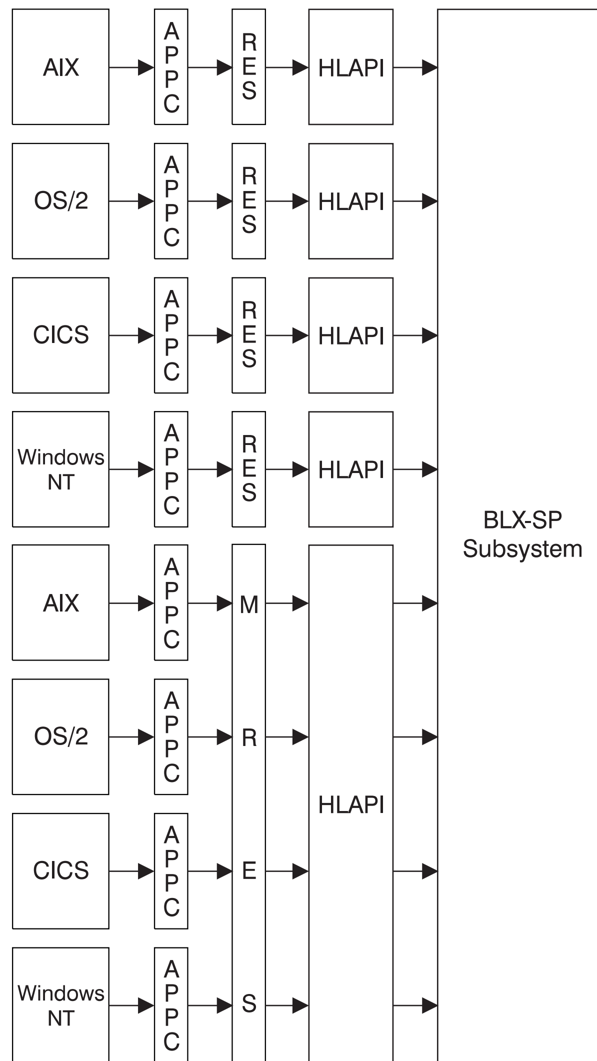


Figure 3. Client/Server Configuration Using RESs and an MRES with APPC

Multiclient Remote Environment Server (MRES) with TCP/IP

Tivoli Information Management for z/OS also provides a server that uses TCP/IP communication protocol. This server is the Multiclient Remote Environment Server with TCP/IP (MRES with TCP/IP). Like the address space for MRES with APPC, the address space for MRES with TCP/IP runs as a task started by the MVS operator. The MRES with TCP/IP uses socket interfaces to communicate with TCP/IP. A single address space for MRES with TCP/IP can receive and process transactions from multiple Tivoli Information Management for z/OS clients concurrently. Address spaces for RES, MRES with APPC, and MRES with TCP/IP can be active on an MVS machine at the same time. Each MRES with TCP/IP can access only one BLX-SP.

The minimum software required to define an MRES with TCP/IP is:

- OS/390 Version 2.5, or a later version

Figure 4 illustrates a client/server configuration that could be achieved using an MRES with TCP/IP. The OS/2, UNIX, Windows NT, and USS clients can communicate with an MRES with TCP/IP.

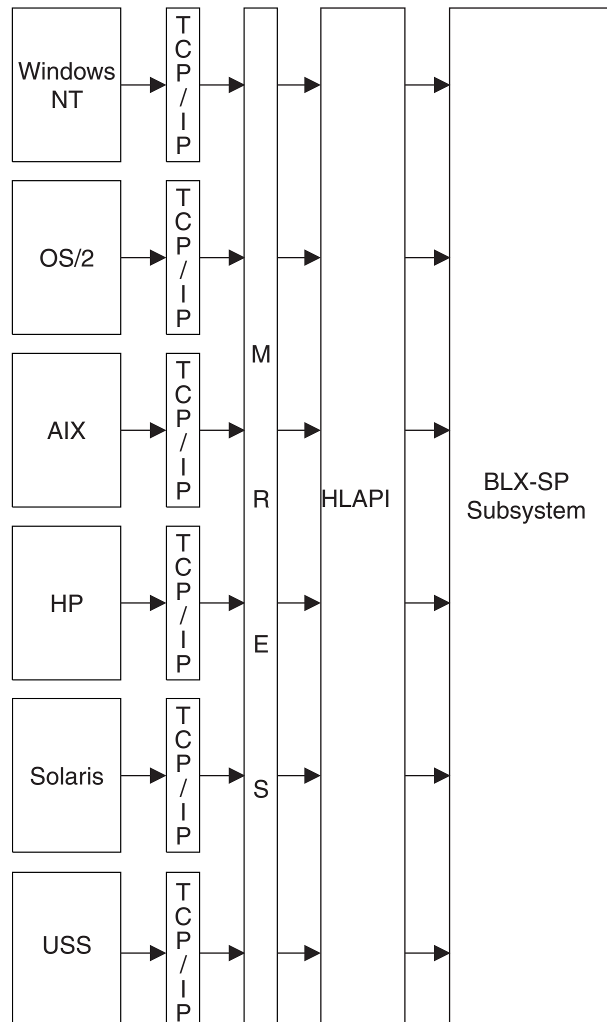


Figure 4. Client/Server Configuration Using an MRES with TCP/IP

See "Configuring and Running a Multiclient Remote Environment Server (MRES) with TCP/IP" on page 53 for information on installing and starting an MRES with TCP/IP.

Figure 5 on page 10 illustrates a client/server configuration that could be achieved using the RES, MRES with APPC, and MRES with TCP/IP. In this example, they all use the same BLX-SP.

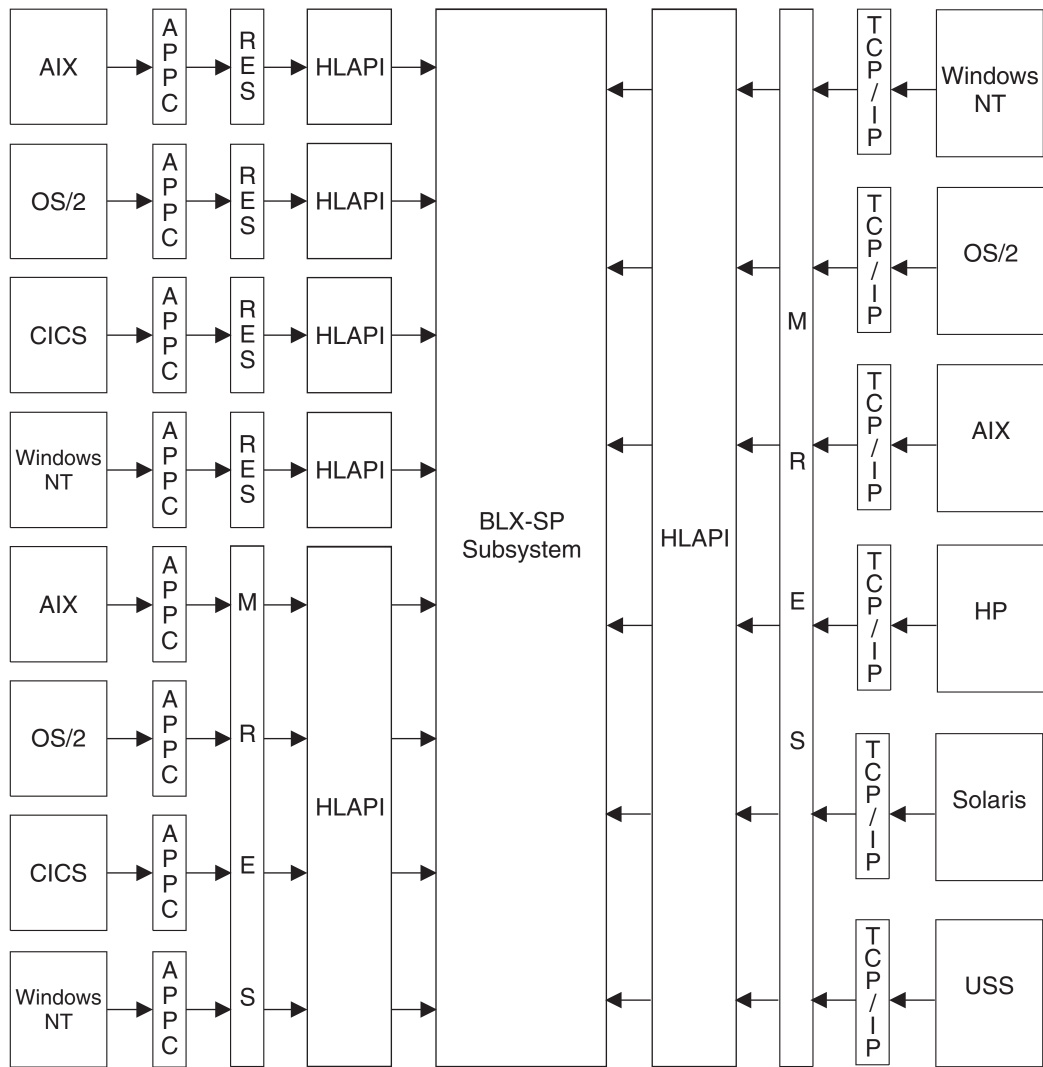


Figure 5. Client/Server Configuration Using RESs, an MRES with APPC, and an MRES with TCP/IP

Java Applications

Java programs can interface with Tivoli Information Management for z/OS clients on the following operating systems, which must also support the Java Runtime Environment (JRE) Version 1.1:

- AIX
- HP-UX
- Sun Solaris
- OS/2
- Windows®/NT

A sample Java program that illustrates the use of the Java class objects is also provided with the clients. "Tivoli Information Management for z/OS Java Wrappers (HLAPI for Java)" on page 353 contains additional information on the following classes and their methods:

- The **Hicao** class which contains methods and data to handle HICA field data and collections of PDBs.

- The **Pdbo** class which contains methods to store and retrieve data from the PDB data structure.
- The **Blmyjwc** class which defines constants used for control PDB names.

2

Choosing a Server

This chapter compares the servers that are supported and discusses characteristics that you need to consider while planning your configuration.

The servers are compared on the following characteristics:

- How they manage conversations, including
  - Which communication protocol they support
  - What type of LU-to-LU conversation they require, if any
  - How many concurrent clients they can support in one server address space (ASID)
- Where you can define security controls
- How you can determine who used a server
- How each server is started

Some performance issues and transaction logging are also discussed.

Table 3 summarizes the comparisons.

Table 3. Comparison of Servers

Server	Protocol	Conversation	Clients per ASID	Security	Accounting	Startup
RES	APPC	SCHEDULED	1	APPC/MVS, VTAM®, RACF® (or comparable security product), and TP profile	SMF type 33, subtype 1 records	Initiated by client
MRES with APPC	APPC	NONSCHEDULED	1 to 50, as specified by MAXCONNECT in the parameters data set	APPC/MVS, VTAM, and RACF (or comparable security product)	SMF type 33, subtype 2 records	Started by MVS operator
MRES with TCP/IP	TCP/IP	Terminology does not apply. Uses sockets.	1 to 50, as specified by MAXCONNECT in the parameters data set	RACF (or comparable security product)	None	Started by MVS operator

RES Conversation Management

The RES supports APPC protocol. It uses the APPC scheduler to schedule conversations and start an address space to serve a client's request for a conversation. Therefore, the APPC and VTAM definitions for a RES must be for a scheduled logical unit.

A single APPC conversation is established for each combination of security user ID, client requester, and symbolic destination name. The symbolic destination name identifies an MVS system where a server resides. Once established, all database connections with the same

combination use this single conversation. (Some clients, though, can support no more than 10 logical sessions per conversation. This is a limitation of the sockets interface.)

As the number of logical sessions associated with a specific server address space increases, the address space might run out of storage. If this happens, an MVS abend ends all of the logical sessions in the address space. This includes any logical sessions that might correspond to HICAs from multiple client application programs.

The number of logical sessions that an address space can support depends on many factors:

- The MVS configuration
- The number of client sessions
- The specific transaction being performed
- What data is in the Tivoli Information Management for z/OS database
- If the client application specifies MVS HLAPI/LLAPI logging

It is suggested that you limit the number of HICAs per conversation. Generally, each MVS host has one conversation with each combination of security user ID, client requester, and symbolic destination name.

Conversations are established based on unique symbolic destination names and security IDs. If more than one symbolic destination name refers to exactly the same system and TP name, more than one conversation is established to that system and TP name. How many different conversations you can establish from one workstation to each symbolic destination is limited by the session limits your system administrator has defined in APPC/MVS and VTAM on the host for the server's logical unit.

MRES with APPC Conversation Management

The MRES with APPC also supports APPC protocol. Facilities available on z/OS enable the MRES to support conversations with multiple clients in a single MVS address space and to bypass the APPC scheduler when it processes LU 6.2 inbound transaction program requests. Therefore, the APPC and VTAM definitions for an MRES must be for a nonscheduled LU.

The MRES address space runs as a task started by the MVS operator. The number of LU-to-LU conversations this address space can serve simultaneously is determined by the number of *client conversation processors* the task starts. A client conversation processor is a program that connects a client conversation to a HLAPI session. Transactions from the client flow through this connection to the HLAPI. The results from the HLAPI flow back through this connection to the client.

When an MRES is started, it registers with APPC/MVS to receive inbound conversations. Thereafter, APPC/MVS monitors all the inbound conversation requests for those requests for which the MRES has registered. Instead of routing the requests to the APPC scheduler, APPC/MVS places the requests on a queue to await further processing by the MRES. The queue is called an *allocate queue*. The MRES processes conversation requests by receiving them from the allocate queue and performing the function requested by the client transaction program. This allows a single MRES address space to receive and process transactions from multiple Tivoli Information Management for z/OS clients concurrently. This removes the

burden of having a separate address space for each client connection. Both RES and MRES address spaces can be active on an MVS machine at the same time. Also, each MRES can access only one BLX-SP.

If memory abends are a problem with this server, you can either increase the region size in the started procedure or reduce the number of client conversation processors this server supports.

The number of client conversation processors that an address space can support depends on many factors:

- The size of the address space
- The MVS configuration
- The number of logical sessions flowing on each conversation
- The specific transactions being performed
- What data is in the Tivoli Information Management for z/OS database
- If the client application specifies MVS HLAPI/LLAPI logging

MRES with TCP/IP Conversation Management

The MRES with TCP/IP supports the TCP/IP protocol. Like the MRES with APPC, it can support multiple clients in a single MVS address space. It uses sockets as the interface between the client and Tivoli Information Management for z/OS. The MRES with TCP/IP uses the standard OS/390 UNIX System Services socket interface. You must create and define a generic user to OS/390 UNIX System Services and associate the user with the name of the MRES to be started.

Note: If you are migrating from an earlier release of Tivoli Information Management for z/OS, you should review your specifications. Some of the parameters may have changed. For example, the parameter **WTO** that was valid in earlier releases should now be specified as **WRITEOPER**. In a like manner, in earlier releases, the TCP/IP address space name was specified with the **ASN** parameter. This parameter is not valid in this release, and specifying it will result in an error. In addition to the changed parameters, you need to specify your MRES startup parameters in a data set rather than inline in your cataloged procedure, as was done in earlier releases. A list and description of valid parameters for this release can be found in "Coding the Parameters for an MRES with TCP/IP" on page 58.

The MRES with TCP/IP address space runs as a task started by the MVS operator. The number of clients this address space can serve simultaneously is determined by the number of client communication processors this task starts. RES, MRES with APPC, and MRES with TCP/IP address spaces can all be active on an MVS machine at the same time.

Each MRES with TCP/IP can access only one BLX-SP.

If memory abends are a problem with this server, you can either increase the region size in the started procedure or reduce the number of client conversation processors the server supports.

The number of client conversation processors that an address space can support depends on many factors:

- The size of the address space
- The MVS configuration
- The number of logical sessions flowing on each conversation
- The specific transactions being performed
- What data is in the Tivoli Information Management for z/OS database
- If the client application specifies MVS HLAPI/LLAPI logging

Logical Unit (LU) Considerations

The RES and the MRES with APPC both require logical unit definitions in APPC/MVS and VTAM. The LU for a RES must be scheduled. The LU for an MRES must be nonscheduled. Therefore, you cannot use the same LU for both a RES and an MRES.

The clients you use may have requirements that you need to consider when defining LUs also. For example, HLAPI/CICS clients may not need password verification, but HLAPI/2, HLAPI/NT, and HLAPI/AIX clients do. If you have both HLAPI/2 and HLAPI/CICS clients, one way to handle the security issue is to define LUs for the exclusive use of each client and create security definitions to restrict the use of each LU to a particular client.

Refer to the publications listed in “Security References” on page 22 for more information.

Refer to *MVS/ESA Planning: APPC Management* for more information about defining LUs that implement security mechanisms.

Socket Considerations

The MRES with TCP/IP uses sockets. The client must know the port number for the server’s socket. So, if you decide to change the port number when you start an MRES with TCP/IP, you must inform the clients that use the server of the change. The client also has to know the IP address or the host name of the server. The MRES with TCP/IP uses the standard OS/390 UNIX System Services socket interface. You must create and define a generic user to OS/390 UNIX System Services and associate the user with the name of the MRES to be started.

Accounting Considerations

System Management Facilities (SMF) is the MVS component that collects and records system and job related information for accounting, configuration, performance, and security management purposes.

RES

Your installation can track the use of RES resources through the SMF record type 33, subtype 1. This record provides data that includes the following information about inbound conversations scheduled by APPC:

- TP name, TP profile name, and TP class
- Local and partner LU names
- Queueing and processing times for the scheduled program
- Number of sends and receives for the request

You can also use an SMF account validation exit to validate the accounting information for TPs whose profiles specify **TAILOR_ACCOUNT(YES)**. The exit receives control when a new unit of work is prepared for processing and is passed the user ID and account number information. The exit can validate that information and approve or deny the processing of the related TP instance.

The *OS/390 MVS System Management Facilities (SMF)* contains additional information about SMF.

MRES with APPC

Your installation can track the use of MRES with APPC resources through the SMF record type 33, subtype 2. This record provides data that includes the following information:

- Conversation user ID and conversation ID
- TP name for the conversation
- Local and partner LU names
- The date and time a request was:
  - Received by APPC/MVS
  - Queued for processing
  - Delivered to the processing program
  - Deallocated
- Number of sends and amount of data sent
- Number of receives and amount of data received

For more information, *OS/390 System Management Facilities (SMF)* has further discussion of SMF accounting for APPC/MVS servers.

Performance Considerations

If you use an MRES with APPC or an MRES with TCP/IP, the total elapsed time required to perform the first HL01 (initialization) transaction received from the client will be less than if you use a RES. This is because the MRES address space has already been started and the client conversation processing task is waiting to handle the first transaction. If you are using an MRES with pre-started API sessions, you may obtain better HL01 performance because session initialization is done when the MRES is started; this is described in "MRES with Pre-started API Sessions Considerations" on page 18.

The symbolic destination name, specified when an MRES with APPC is started, defines the APPC allocate queue that will be monitored. More than one MRES with APPC can be started and assigned to the same APPC allocate queue. As client conversations arrive in the queue, they are received by the waiting MRESs. When all of the MRESs assigned to an allocate queue have received their maximum number of active conversations, any additional conversations arriving in the queue must wait until an MRES becomes available to receive it. This wait time, which does not occur in RES processing, can be significant, thus impacting performance of the waiting conversations. The MVS operator can use APPC commands to monitor the activity of the allocate queue. Another solution is for the operator to start a sufficient number of MRESs beforehand to handle the volume of active conversations.

The port number and internet protocol address, specified when an MRES with TCP/IP is started, defines the socket that receives the connection request. More than one MRES with TCP/IP can be started, but each must have a unique port number. As client connection requests arrive at the socket, they are received by the waiting MRES with TCP/IP. When the MRES with TCP/IP has received its maximum number of connection requests, any additional requests arriving at the socket are rejected.

MRES with Pre-started API Sessions Considerations

You may improve performance by using pre-started API sessions. When you use pre-started API sessions, the initialization (HL01) time for a client connecting to Tivoli Information Management for z/OS through the MRES may be shortened. This is because each client conversation processor (CCP) issues the HL01 transaction when the MRES is first started and before clients begin to connect in. If some of your clients typically connect to the MRES using the same HL01 control values, the clients should achieve a faster response when they submit the HL01 to a Tivoli Information Management for z/OS that is using pre-started sessions.

With either MRES with APPC or MRES with TCP/IP, the system administrator can request that an MRES pre-start its API sessions by providing the appropriate keywords in the data set specified on the **BLMYPRM DD** statement in the cataloged procedure for the MRES. The parameters that are specified have the same names as the corresponding control PDBs for the HL01 HLAPI transaction.

The MRES parameter **PRESTARTSESSIONS** (described on page 41) is used to indicate whether the MRES should pre-start API sessions. Based on the specification of this parameter, either none of the CCPs will pre-start an API session or else all of the CCPs will pre-start an API session.

Existing clients can use an MRES with pre-started API sessions by ensuring that the values, if specified, for **SESSION_MEMBER**, **DATABASE_ID**, and **BYPASS_PANEL_PROCESSING**, match those specified for the MRES. If these values are different, the HL01 is rejected.

New clients can be designed to take advantage of the pre-started API sessions. In doing so, the HL01 transaction is still needed. As stated in the preceding paragraph, if values are specified for **SESSION_MEMBER**, **DATABASE_ID**, and **BYPASS_PANEL_PROCESSING**, they must match the values specified for the MRES. For workstation and HLAPI/USS clients, **SECURITY_ID** and **PASSWORD** are required because these are used by RACF to ensure proper access authority to MVS. Tivoli Information Management for z/OS also checks access authority by using the specifications for **APPLICATION_ID** and **PRIVILEGE_CLASS**. The value for **APPLICATION_ID** must be provided by the client on either the HL01 or on the first transaction sent after the HL01. A specification for **PRIVILEGE_CLASS** is optional; if a privilege class is not specified by the client, the one used to pre-start the API session is used. Finally, workstation clients must also specify a value for **DATABASE_PROFILE**.

Note: The values specified by the client application remain in effect until changed on a subsequent transaction by the same application or another application.

Because the API sessions will be open for as long as the MRES is running, an attempt will be made to restart a session if it is down when a client's transaction is being processed. One

reason why the session may be down is that the BLX-SP may have been stopped and restarted. The API return codes will be analyzed to determine if the session can be restarted. If the API session cannot be restarted, the error codes will flow back to the client attempting the transaction.

When a client conversation sends an HL01 transaction to a pre-started MRES session, the parameters **SESSION_MEMBER**, **DATABASE_ID**, and **BYPASS_PANEL_PROCESSING**, if specified, are compared with the pre-started values. If they are different, Return Code 12, Reason Code 200 is returned to the client. If any of these three PDBs are not specified on the HL01 transaction, then the MRES assumes that the client wants to inherit the values specified for the pre-started session. For example, if the session is pre-started with **DATABASE_ID=8** and the **DATABASE_ID** is not specified on the HL01, then the HL01 value is assumed to be **8**, and no error occurs.

Note: Note that the Client HL01 uses the specified value of the pre-started **DATABASE_ID** parameter and does not take the HLAPI default of **5**.

When a client conversation sends an HL01 transaction to an MRES with pre-started API session, the **APPLICATION_ID**, **PRIVILEGE_CLASS**, and **DATE_FORMAT** values are saved. If the next transaction received from the client does not specify these PDBs, then the values from the HL01 are appended to the transaction. If the **PRIVILEGE_CLASS** or **DATE_FORMAT** is not specified by the client on the HL01 or the next transaction, the values used by the MRES to pre-start the session are appended to the transaction. If the client does not provide an application ID, the transaction does not run and Return Code 12, Reason Code 201 is returned to the client. This ensures that the privilege class check is made and that the correct application ID is used for the audit trail.

When data model records or PIDTs are updated and you want to force the cache to get the new updates, use the BRDCST operator command with the TABLES keyword to pick up the updates. More information on using the BRDCST command can be found in *Tivoli Information Management for z/OS Operation and Maintenance Reference*.

MRES with APPC Cataloged Procedure Considerations

You can start multiple MRESs with APPC using the same cataloged procedure. If you do that, you will be able to distinguish one MRES from another by its address space identification (ASID). Additional information on starting an MRES with APPC can be found in “Starting and Stopping an MRES with APPC” on page 49; you may also wish to see the JCL provided in “Defining a Procedure for an MRES with APPC” on page 38.

HLAPI/2, HLAPI/CICS, HLAPI/AIX, and HLAPI/NT client application programs can all access the same MRES at the same time. If you define an LU for the exclusive use of each type of client or for the exclusive use of an application program, though, you must define a corresponding MRES cataloged procedure to start the address space that uses the LU.

MRES with TCP/IP Cataloged Procedure Considerations

You can start only one MRES with TCP/IP using the same cataloged procedure. You must create a separate cataloged procedure, each with a unique port number. Additional information on starting an MRES with TCP/IP can be found in “Starting and Stopping an MRES with TCP/IP” on page 65; you may also wish to see the JCL provided in “Defining a Procedure for an MRES with TCP/IP” on page 56.

Transaction Logging by a RES and by an MRES Without Pre-started API Sessions

The content of the server log entries produced by a client is similar to the content of log entries produced by the HLAPI. If logging is enabled for a client's logical session, the transactions of the logical session are logged when they are processed by the HLAPI. Log entries include both transaction request data and transaction reply data. Transactions are logged in the order in which they are completed on the MVS host.

For client transaction sequences, the server's transaction program or cataloged procedure JCL allocates the **DDNAME HLAPILOG**. Therefore, a client application program cannot control allocation of a log in some of the ways a HLAPI application program can. For example, a client application program cannot use a CLIST, logging on, or executing JCL to control allocation of a log.

The client application program can control the content and size of the server log through control PDBs. This conforms to the HLAPI logging implementation. An application program can control the server log in the following ways:

- Turn logging on or off for a logical session. An application can turn logging off for a session by omitting the **SPOOL_INTERVAL** control PDB on the HL01 transaction of the transaction sequence.
- Set the server log spool interval. An application can control the reuse of the server log by specifying a time interval in a **SPOOL_INTERVAL** control PDB on an HL01 transaction. When the specified time interval elapses, logging continues at the beginning of the log and previous entries are overwritten.
- Control the messages. An application can use the **APIMSG_OPTION** control PDB on an HL01 transaction to control the writing of a logical session's Low-Level Application Program Interface (LLAPI) messages to the server log. An application can use the **HLIMSG_OPTION** control PDB on an HL01 transaction to control the writing of a logical session's High-Level Application Program Interface (HLAPI) messages to the server log.

The server log entries for each logical session are labeled with the identifier specified by the **HLAPILOG_ID** control PDB passed to the HLAPI client on the HL01 transaction that initiated the session.

When two or more logical sessions use the same conversation, and when multiple conversations use the same MRES, the following rules apply:

- If the **DDNAME HLAPILOG** is allocated to a data set, transactions are logged only for the logical session that most recently submitted an HL01 transaction with logging specified. Transactions of other logical sessions are not logged in the server log file.
- If the **DDNAME HLAPILOG** is allocated to **SYSOUT**, transactions from all of the logical sessions are interleaved in the server log file.
- When different spool intervals are specified by different logical sessions, the spool interval applied to the server log is the spool interval specified most recently by any logical session.

The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information about transaction logging.

Transaction Logging by an MRES with Pre-started API Sessions

The MRES controls the content and size of the server log through the values specified for the **HLIMSG_OPTION** and the **SPOOL_INTERVAL** pre-start parameters. These parameters control the server log for all of the pre-started tasks as a unit and cannot be varied for individual CCP sessions. The log entries in the HLAPILOG are labeled with the character string **MRESCP** followed by the 2-digit number of the CCP task, starting with 01. If you are using pre-started API sessions, the client application has no control over transaction logging. Any values that you specify in a client PDB for **SPOOL_INTERVAL**, **HLIMSG_OPTION**, or **APIMSG_OPTION** are ignored.

The MRES can control the server log in the following ways:

- Turn logging on or off for a logical session. The MRES can turn logging off for a session by omitting the **SPOOL_INTERVAL** pre-start parameter on the **BLMYPRM DD** statement.
- Set the server log spool interval. The MRES can control the reuse of the server log by specifying a time interval in a **SPOOL_INTERVAL** pre-start parameter on the **BLMYPRM DD** statement. When the specified time interval elapses, logging continues at the beginning of the log and previous entries are overwritten.
- Control the messages. The MRES can use the **APIMSG_OPTION** pre-start parameter on the **BLMYPRM DD** statement to control the writing of a logical session's Low-Level Application Program Interface (LLAPI) messages to the server log. The MRES can use the **HLIMSG_OPTION** pre-start parameter on the **BLMYPRM DD** statement to control the writing of a logical session's High-Level Application Program Interface (HLAPI) messages to the server log.

The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information about transaction logging.

Security Considerations

The clients all send identifying information to the server. You can use this identifying information to implement security for your servers. For the HLAPI/UNIX, HLAPI/2, HLAPI/NT, and HLAPI/USS clients using the TCP/IP protocol, this information includes a user ID and password. For the HLAPI/2, HLAPI/AIX, and HLAPI/NT clients using the APPC/APPN® protocol, this information includes a user ID, password, and symbolic destination name. For the HLAPI/CICS client, this information includes a user ID and a partner ID. The HLAPI/CICS does not require a password because it assumes CICS has already checked the user ID's password. The symbolic destination name and partner ID identify the LUs, side information entries, and any other APPC resources the client application program wants to access. The user ID and password identify an authorized user of those APPC resources.

If you are using the APPC protocol, APPC performs the security verifications you have set up, and then the RES or MRES calls RACF (or a comparable security product) to verify that the user ID is authorized to initialize a Tivoli Information Management for z/OS session. Each server, whether using the APPC protocol or the TCP/IP protocol, uses RACF (or a comparable security product) to verify that the user ID is authorized to initialize a Tivoli Information Management for z/OS session. If the user ID passes that verification, Tivoli Information Management for z/OS verifies that the application ID is a member of a privilege class authorized to access the Tivoli Information Management for z/OS database. For more

information about the **APPLICATION_ID** control PDB, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*. Security checking ensures that a user has the authority to use the value specified in **PICAUSRN**. (The value specified in **PICAUSRN** ensures that the MVS user ID(s) running your application are allowed to use this application ID). This security checking is optional and is implemented by the system administrator. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information about this security feature.

If you plan to secure your servers, you need to use parameters on your APPC/MVS and VTAM definitions that enable security checking. You also need to define security classes and profiles in a security product, such as RACF. The classes you define depend on the server. The following sections briefly discuss some of the security measures you can take to provide security for the Tivoli Information Management for z/OS servers that use APPC/APPN.

Security References

The following publications provide more information about security:

- *OS/390 Version 2 Release 5 Security Server (RACF) Planning: Installation and Migration*
- *OS/390 Version 2 Release 5 MVS Planning: APPC/MVS Management*
- *VTAM Resource Definition Reference*
- *AIX for RISC System/6000® General Concepts and Procedures*

Security for a RES

You enable security in the following places:

- **APPC** on the **LUADD** definition and the **TP profile** definition. Also, by adding database tokens to the administrative data sets (the **TP profile** and side information data sets).
- **VTAM** on the **APPL** and **MODE** definitions.
- **RACF** or a comparable security product.
- **APISECURITY** in the BLX startup parameters

APPC/MVS Definitions That Enable Security

In APPC/MVS, you can specify a search order for the TP profile. You do this with **TPLEVEL** on the **LUADD** statement. **GROUP** means to look for profiles associated with a group of users. The default level of the TP profile is **SYSTEM**, which means that any user ID on the MVS host can use this TP. However, if your MVS host has a security product such as RACF installed, consider using **GROUP** or **USER** to control access to the TP. If you do that, you can specify the name of the group or user in the TP profile on the **GROUPID** parameter.

For example, HLAPI/2 users need to have their user IDs and passwords verified. So you could create a TP profile for them that specifies **GROUPID(IMOS2)** and have your security administrator define the IMOS2 group in your security product and associate the HLAPI/2 user IDs with that group.

A single TP can have profiles for all three levels: one available for all users on the system (highest level), one for a specified group of users, and one for an individual user (lowest level). When APPC/MVS receives an incoming allocate request for a TP with more than one profile, it uses the TP profile with the lowest level to which the user ID has access.

If you defined the TP as **SYSTEM** when you created the profile, define the **LU** as **SYSTEM** also on the **TPLEVEL** statement. This will prevent unnecessary searching for profiles, because if the LU is defined as **GROUP**, APPC searches for a TP profile that specifies **GROUPID**.

Refer to the publications listed in “Security References” on page 22 for more information.

VTAM Definitions That Enable Security

Values that you specify for **SECACPT** and **VERIFY** on the **VTAM APPL** statement enable security checking. If you are defining an LU for a CICS client, specify **SECACPT=ALREADYV** because the user is already verified. If you are defining an LU for an OS/2 or AIX or Windows NT client, specify **SECACPT=CONV** because users of these clients are not yet verified.

You can customize the conversation security levels accepted for inbound requests. The **CONVSEC** operand in the **SESSION** segment of the RACF **APPCLU** class implements this in conjunction with VTAM.

VTAM also provides data encryption facilities you may want to consider using.

Refer to the publications listed in “Security References” on page 22 for more information.

RACF Definitions

RACF offers several classes to implement APPC/MVS security. The **APPCLU** class enables customization of conversation security. The **APPCTP** class controls access to TPs. The **APPCPORT** class controls access to remote LUs. The **APPL** general resource class controls what groups of APPC/MVS applications the user has access to.

Refer to the publications listed in “Security References” on page 22 for more information.

Security Considerations When Using Pre-started API Sessions

When the BLX-SP has been started with the **APISECURITY** parameter set **ON**, RACF checking is performed whenever the application ID is different from the user ID of the task starting the API session. When an operator starts the MRES task, the system default is to not assign a user ID. Thus, a pre-started API session will fail with a RACF error. It is possible to define a started procedures table in the operating system which names the MRES and a user ID to be assigned when the task is started. By doing this, the session can be pre-started.

Security for an MRES with APPC

The MRES with APPC is a started procedure. It must be able to access any RACF-protected resources it needs. If your site has a started procedures table, your security administrator can associate the names of the MRES cataloged procedures with specific RACF user IDs and group names. This ensures that the MRES started procedures can access the resources they need.

You can use APPC and VTAM definitions to protect the MRES with APPC the same way that you use them to protect a RES. However, you cannot use the **APPCTP** class because the MRES with APPC does not use a TP profile. The MRES with APPC uses the side information data set on MVS to resolve the symbolic destination name on conversation requests to a TP name. RACF provides the **APPCSERV** class that you can use to verify that a server is authorized to serve a particular TP.

RACF also provides an **APPCSI** class. However, it controls access to side information entries used on outbound requests. (The RES, MRES with APPC, and MRES with TCP/IP handle only inbound requests.) You might want to implement this class, though, to protect the side information data set from unauthorized changes.

Refer to the publications listed in “Security References” on page 22 for more information.

Security for an MRES with TCP/IP

The MRES with TCP/IP is a started procedure. It must be able to access any RACF-protected resources it needs. If your site has a started procedures table, your security administrator can associate the names of the MRES cataloged procedures with specific RACF user IDs and group names. This ensures that the MRES started procedures can access the resources they need.

All of the Tivoli Information Management for z/OS HLAPI clients, except for the HLAPI/CICS, can use the TCP/IP protocol. When one of these clients wants to use TCP/IP, the MVS IP address and port number are used to find the MRES.

The MRES with TCP/IP calls RACF to verify that the user ID is authorized to initialize a Tivoli Information Management for z/OS session. If the user ID passes that verification, Tivoli Information Management for z/OS verifies that the application ID is a member of a privilege class authorized to access the Tivoli Information Management for z/OS database. For more information about the **APPLICATION_ID**, **SECURITY_ID** and **PASSWORD** control PDBs, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*.

3

Configuring and Running a Remote Environment Server (RES)

This chapter tells you how to configure a Remote Environment Server (RES) on a network that has the following software:

- OS/390 Version 2.5, or a later version
- Tivoli Information Management for z/OS Version 7.1

The configuration described in this chapter is based on a zSeries host connected to a 3745 Communication Controller with a token-ring attachment that serves as a gateway to the rest of the network. Figure 6 illustrates a RES that was started in response to a request from an OS/2 client.

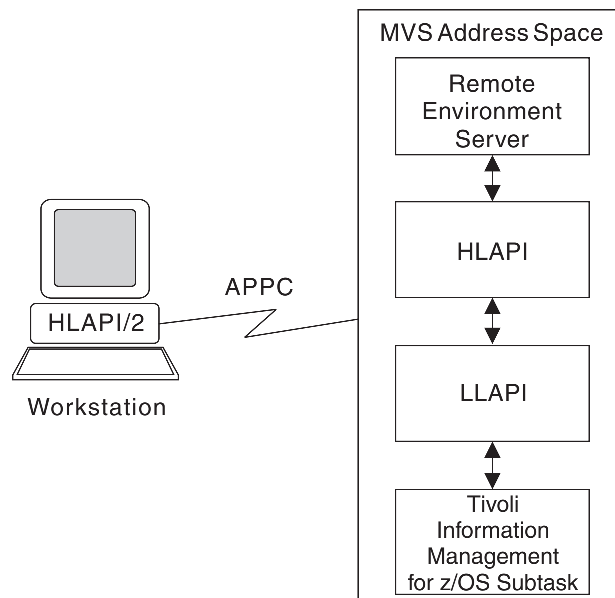


Figure 6. RES Started by an OS/2 Client

Using a RES

The Remote Environment Server (RES) runs as a transaction program (TP) that is started by APPC/MVS when a connection request is received from a Tivoli Information Management for z/OS client application program. For each unique connection request, the APPC scheduler starts a new address space with a RES dedicated to servicing that client's HLAPI transactions. Each RES can access only one BLX-Service Provider (BLX-SP).

RES Configuration Tasks

To configure a RES, do the following tasks:

1. Plan your configuration. "Planning Your RES Configuration" discusses this task.
2. Set up APPC/MVS. "Setting Up APPC/MVS" discusses this task.
   a. Create a VSAM data set for the APPC transaction program profile. "Creating a VSAM Data Set for the TP Profile" discusses this task.
   b. Make an entry for the RES in the transaction program profile. "Making an Entry for the RES in the TP Profile Data Set" on page 27 discusses this task.
   c. Define a local LU for the RES and identify its partner LUs in **APPCPM***xx* of **SYS1.PARMLIB**. "Defining Local LUs and Identifying Partner LUs" on page 28 discusses this task.
   d. Define a schedule class for the RES in **ASCHPM***xx* of **SYS1.PARMLIB**. "Defining a Schedule Class" on page 29 discusses this task.
3. Set up VTAM. "Modifying VTAM" on page 30 discusses this task.
   a. Define APPC/MVS local LUs in **SYS1.VTAMLST**. "Defining the Local LU in VTAM" on page 30 discusses this task.
   b. Define the log-on mode in **SYS1.VTAMLIB**. "Defining the Log-on Mode" on page 30 discusses this task.
4. Define a link to each client requester you plan to use with the RES. "Defining Links to Clients" on page 31 discusses this task.
5. Define security classes and profiles. "Defining Security Classes and Profiles" on page 32 discusses this task.

This server is started in response to a request from a client. Both APPC and the APPC scheduler must be started and running before requests can be received. See "Starting and Stopping the APPC Environment" on page 32 for information about starting the APPC environment.

Planning Your RES Configuration

Setting up a RES is independent of setting up a client. However, you need to consider which clients you want to use a RES. The clients will need to know some of the values you declare when you configure the RES. And you will need to know some of the client's values when you configure the RES. See "Choosing a Server" on page 13 for a discussion of factors you should consider while planning your configuration.

Setting Up APPC/MVS

Either you or the APPC/MVS system administrator of your communications network does the tasks outlined in this section. The information in this section is presented to help you plan your configuration. Refer to the *OS/390 MVS Planning: APPC/MVS Management* manual for more information.

Creating a VSAM Data Set for the TP Profile

Use IDCAMS to create a VSAM key sequenced data set for the transaction program profile. APPC/MVS needs this data set to be able to associate TP names with particular job streams

when an incoming request to start a TP is received. You must define this file first, before you associate it to an LU in **SYS1.PARMLIB**.

You can use the JCL in the following example to create a VSAM data set for TP profiles. Change *volname* to the name of your local volume.

```
//jobname  JOB your-job-card
//JOBPARM LINES=9999,TIME=1440
//TPSAMPLE EXEC PGM=IDCAMS
//volname  DD   DISP=OLD,UNIT=3380,VOL=SER=volname
//SYSPRINT DD   SYSOUT=*
//SYSABEND DD   SYSOUT=*
//AMSDUMP  DD   SYSOUT=*
//SYSIN    DD   *
     DEFINE CLUSTER (NAME(SYS1.APPCTP) -
          VOLUMES (volname) -
          INDEXED REUSE -
          SHAREOPTIONS(3 3) -
          RECORDSIZE(3824 7024) -
          KEYS(112 0) -
          RECORDS(300 150)) -
        DATA -
          (NAME(SYS1.APPCTP.DATA)) -
        INDEX -
          (NAME(SYS1.APPCTP.INDEX))
/*
```

Making an Entry for the RES in the TP Profile Data Set

A TP profile contains the identification, security, and scheduling information for a transaction program. You can use the APPC/MVS administration utility, program ATBSDFMU, to add a TP profile entry for each client application program that can start a RES. The utility commands are issued in the SYSIN data stream. Refer to the *MVS/ESA Planning: APPC Management* guide for more information about how to define the TP profile. Set the following parameters as indicated:

1. The value for **TPNAME** must match the value specified for **TPNAME** in the CPI-C side information entry on the client. In on page 28, the **TPNAME** is **INFOTP1**.

   Note: You must create a CPI-C side information entry on each workstation or CICS system that uses a RES. See the sections on installing the individual clients.

2. The default level of a TP profile is **SYSTEM**, which means that any user ID on the MVS host can use the TP. **SYSTEM** is allowed for a RES. However, if your MVS host has a security product such as RACF installed, consider using **GROUPID** or **USER** to control access to the TP.

   A single TP can have profiles for all three levels: one available for all users on the system (highest level), one for a specified group of users, and one for an individual user (lowest level). When APPC/MVS receives an incoming allocate request for a TP with more than one profile, it uses the TP profile with the lowest level to which the user ID has access.

3. The value for **TPSCHED_EXIT** must be **ASCH** for scheduled. This is the default.

4. The value for **TPSCHED_TYPE** must be **STANDARD**. This is the default.

5. To better control the processing environment of each TP, assign each TP to its own **CLASS**. In the following example, the **CLASS** is **FAST**. See “Defining a Schedule Class” on page 29 for information on defining a class.

6. Specify **BLMYSCSC** as the program name on the **IKJAACNT EXEC** statement. A **STEPLIB DD** statement must point to the library containing the load modules used by Tivoli Information Management for z/OS. Refer to the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* for more information about Tivoli Information Management for z/OS load modules and how to allocate them.

Note: For the purposes of the remote clients, you can continue to define the **RFT** data sets the same way you normally do for API use. Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* and the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* for more information about **RFT** definition.

```
//jobname  JOB your-job-card
//************************************************************
//***  ADD USERS TO TP PROFILE          **********************
//***                                   **********************
//************************************************************
//* IN THIS EXAMPLE, A GROUP-LEVEL TP PROFILE
//* FOR INFOTP1 IS ADDED TO THE TP PROFILE FILE
//* SYS1.APPCTP.  WHEN ADDING TP PROFILES, THE TP PROFILE
//* KEY PRECEDES THE TP PROFILE. THE TP PROFILE CONSISTS OF:
//*       I) A TP ATTRIBUTES SECTION
//*      II) A TRANSACTION SCHEDULER SECTION (DELIMITED)
//* THE FOLLOWING DEFAULTS ARE ASSUMED:
//*       I) TYPRUN=RUN
//*      II) TPSCHED_EXIT(ASCH)
//************************************************************
//STEP     EXEC PGM=ATBSDFMU
//SYSPRINT DD   SYSOUT=*
//SYSSDLIB DD   DSN=SYS1.APPCTP,DISP=SHR
//SYSSDOUT DD   SYSOUT=*
//SYSIN    DD   DATA,DLM=XX
     TPADD                                <-- Utility command to add a TP profile
          TPNAME(INFOTP1)                 <-- Transaction program name
          GROUPID(IMCICS)                 <-- Name of group of authorized userids
          ACTIVE(YES)                     <-- Attribute to allow access
          TPSCHED_DELIMITER(##)           <----,
             CLASS(FAST)                       | TP scheduler section
             TPSCHED_TYPE(STANDARD)            |
             JCL_DELIMITER(END_OF_JCL)         |
//INFOAPI1 JOB  MSGLEVEL=(1,1)                 | Job name
//IKJAACNT EXEC PGM=BLMYSCSC              <----| Program to run to start server
//STEPLIB  DD DSN=BLM.V1R2M0.SBLMMOD1,DISP=SHR |
//APIPRINT DD SYSOUT=*                    <--- | Send LLAPI messages to SYSOUT class
//HLAPILOG DD SYSOUT=*                    <--- | Send HLAPI messages to SYSOUT class
//SYSPRINT DD  SYSOUT=*,FREE=CLOSE             |
END_OF_JCL                                     |
   KEEP_MESSAGE_LOG(NEVER)                     |
##                                        <----| End of TP scheduler section
XX                                        <----| End of SYSIN data stream
/*
//
```

The sample profile uses the Tivoli Information Management for z/OS database subsystem **BLX1**, which is the default. If you use a subsystem other than this, you must add a **DD** statement to the **STEPLIB** concatenation for the **BLXx LOAD** data set. This statement must be inserted before the DD statement that specifies the **SBLMMOD1** data set.

Defining Local LUs and Identifying Partner LUs

You associate the local LU with the TP by an **LUADD** statement in the **APPCPM***xx* member of **SYS1.PARMLIB**. In the **LUADD** statement, you specify the TP profile data set name, the search level for the TP profile, and scheduling information.

The following example illustrates the used of an **LUADD** statement.

- **ACBNAME** specifies the local LU name, **BLMSRV01**.
- **SCHED** specifies the transaction scheduler. The value for this parameter must be **ASCH**, which specifies the APPC transaction scheduler.
- **TPDATA** specifies the VSAM data set that contains the TP profiles for this LU. In on page 28, the TP profile was put in **SYS1.APPCTP**.
- **TPLEVEL** specifies the search order for the TP profile. **GROUP** means to look for profiles associated with a group of users.

  If you defined the TP as **SYSTEM** when you created the profile, define the LU as **SYSTEM** also on the **TPLEVEL** statement. This will prevent unnecessary searching for profiles, because if the LU is defined as **GROUP**, APPC searches for a TP profile that specifies **GROUPID**.

```
LUADD
 ACBNAME(BLMSRV01),
 SCHED(ASCH),
 TPDATA(SYS1.APPCTP),
 TPLEVEL(GROUP),
```

Defining a Schedule Class

A class of transaction initiators for the APPC transaction scheduler must be defined in an **ASCHPM***aa* member of **SYS1.PARMLIB**. This member also contains default scheduling information that is used when it is missing from a TP profile. The class name is specified for the TP in the TP profile on the **TPADD** statement in the scheduler section. The sample profile on page 28 specifies **FAST** as the class.

The following example shows the statements that define a class of transaction initiators named **FAST**. Comment lines explain each of the statements.

```
CLASSADD CLASSNAME(FAST), /* Specify the name of the class to be     */
                          /* added                                    */
   MAX(10),               /* Specify that the maximum number of      */
                          /* transaction initiators allowed for      */
                          /* this class is 10                        */
   MIN(2),                /* Specify that the minimum number of      */
                          /* transaction initiators to be brought    */
                          /* up for this class is 2                  */
   RESPGOAL(1),           /* Specify that the response time goal     */
                          /* for transaction programs running        */
                          /* in this class is 1 second               */
   MSGLIMIT(500)          /* Specify that the maximum size of        */
                          /* the job logs for TPs is 500 messages    */
OPTIONS DEFAULT(FAST)     /* Specify the default class.  See Note 1      */
   SUBSYS(yourjes)        /* Name of the subsystem under which all   */
                          /* newly created APPC transaction          */
                          /* are stored                              */
TPDEFAULT REGION(4M)      /* See Note 2                                  */
/*
   OUTCLASS(A)            /* See Note 3                         */
```

Notes

1. The **DEFAULT** (classname) is an optional parameter that specifies the default class of transaction initiators in which to run a TP when a class name is not specified in a TP profile. If the TP profile does not specify a class name, and there is no default defined by this parameter, the request to run the TP is denied. If the default parameter names a class that does not exist, an error message is displayed on the console.

2. The region size assigned to TPs that do not specify a region size in their TP profile is 4MB.
3. The message class used for TPs whose profiles do not specify the **MSGCLASS** keyword in their job statements is **A**. When the **SYSOUT** keyword does not include a specific output class, the value of **OUTCLASS** can be used as a default.

Modifying VTAM

The system administrator for your communications network can modify VTAM so that you can use APPC at your installation on each Tivoli Information Management for z/OS system using a RES. The *OS/390 MVS Planning: APPC/MVS Management* manual contains more information about these modifications, including planning samples. Have your system administrator define the local LU for your RES and define the log-on mode to VTAM. (If an APPC/MVS LU is not defined to VTAM, the LU can only process local requests to and from itself.)

Defining the Local LU in VTAM

Use the **VTAM APPL** statement to define the **BLMSRV01** LU defined to APPC/MVS by the **LUADD** statement in the LUADD example on page 29.

This is an example of a **VTAM APPL** definition for an OS/2 client's use. If this LU were for a CICS client's use, you would specify **SECACPT=ALREADYV**. This definition goes in the **SYS1.VTAMLST** library.

```
APPCAPP VBUILD TYPE=APPL
BLMSRV01       APPL ACBNAME=BLMSRV01,APPC=YES,AUTOSES=1,
                    DDRAINL=NALLOW,DMINWNL=1,DMINWNR=1,DRESPL=NALLOW,DSESLIM=20,
                    EAS=509,MODETAB=ISTINCLM,SECACPT=CONV,VPACING=0,
                    VERIFY=NONE,SRBEXIT=YES,DLOGMOD=#INTER
```

- **ACBNAME** specifies the name of the LU for the RES. This value must be the same as the value you declared for **ACBNAME** on the **LUADD** statement in the **APPCPM***xx* member of **SYS1.PARMLIB**.
- **MODETAB** specifies the name of the VTAM log mode table that contains the definition of **#INTER**, the mode you declared on the **LMADD** statement in the **APPCPM***xx* member of **SYS1.PARMLIB**.
- **DLOGMOD** specifies the name of a compiled log-on mode to use for the conversation. This logon mode is in the VTAM log mode table declared on the **MOTETAB** parameter and is the same one you declared on the **LMADD** statement in the **APPCPM***xx* member of **SYS1.PARMLIB**, **#INTER**. See "Defining the Log-on Mode" for information on defining a mode.
- **SECACPT** specifies whether the client must send a password when requesting a conversation. Use **ALREADYV** if you are defining an LU for a CICS client to use. Use **CONV** if you are defining an LU for an OS/2, an AIX, or a Windows NT client to use.

  Refer to *MVS/ESA Planning: APPC Management* and *VTAM Resource Definition Reference* and *APPC Security: MVS/ESA, CICS/ESA®, and OS/2* for more information about security and defining LUs to VTAM.

Defining the Log-on Mode

You must define the parameters and protocols that determine the communication characteristics of **#INTER**. Log-on modes are entries in a log-on mode table, a compiled version of which exists in the **SYS1.VTAMLIB**.

The following example is a sample of uncompiled source statements to define logmode **#INTER**. This definition is for interactive sessions on resources capable of acting as LU 6.2 devices. Place this definition in the logmode table **ISTINCLM** because that is the mode table you specified when you defined the LU.

This sample is provided in the **BLXVTAML** member of the **SBLMSAMP** library. Refer to the *VTAM Resource Definition Reference* for information on each of the parameters.

```
#INTER   MODEENT LOGMODE=#INTER,FMPROF=X'13',TSPROF=X'07',
               ENCR=B'0000',SSNDPAC=7,RUSIZES=X'F7F7',
               SRCVPAC=7,PSNDPAC=7,APPNCOS=#INTER
```

Defining Links to Clients

The system administrator of your communications network defines the links to clients or to intermediate nodes in NCP and VTAM. The link definition in the following example is for a link to a partner on a token-ring connection.

```
TO30T2PG GROUP ECLTYPE=(PHYSICAL,ANY)
TO30T2PL  LINE  ADDRESS=(1089,FULL),LOCADD=ncplanaddr,PORTADD=2,    X
               RCVBUFC=4095,MAXTSL=2044,ADAPTER=TIC2,TRSPEED=4
*
TO30TRLO GROUP ECLTYPE=LOGICAL,AUTOGEN=10,PHYPORT=2,CALL=INOUT
```

The value you specify on the **LOCADD** parameter (*ncplanaddr*) is the local LAN address that is defined in NCP on a **LINE** macro using the **LOCADD** parameter.

If you want to provide for the use of independent LUs on the token-ring connection, add a statement like the following one. It enables the NCP to support up to 100 independent LUs that are using dial connections, either token-ring or switched lines.

```
POOL1    LUDRPOOL  NUMILU=100,NUMTYP1=20,NUMTYP2=20
```

VTAM has to be able to associate a client’s LU (known as a ***partner LU*** here) with a physical unit. You establish this association in the **SYS1.VTAMLST** library. You can place the statements in the following example in a switched major node definition provided the partner LU is not on another VTAM system. If the partner LU is on another VTAM system, then the link definitions above needs to be for a subarea link, or the cross-domain links need to be defined so that normal LU routing within the subarea network will resolve the locations of the session partners.

```
WSN         VBUILD TYPE=SWNET,MAXGRP=2,MAXNO=2
PARTPU      PU    ADDR=04,                                      X
            CPNAME=partnerlu,                                   X
            PUTYPE=2,                                           X
            MAXDATA=2012,                                       X
            MAXPATH=1
partnerlu   LU       LOCADDR=0,MODETAB=ISTINCLM,DLOGMOD=#INTER
```

Replace *partnerlu* with the name that you defined for the client’s local LU. This example uses **MODETAB=ISTINCLM** and **DLOGMOD=#INTER** because the logon mode **#INTER** is defined in the VTAM logon mode table **ISTINCLM**.

Defining Security Classes and Profiles

If you specified a **GROUPID** in the TP profile you added for your client application program, you can ask your security administrator to add a profile in the **APPCTP** class of RACF. Give your security administrator the following information:

- **APPCTP** profile name for the client application in the format of *dbtoken.level.tpname*
  - *dbtoken* is the database token associated with the VSAM data set that contains the TP profile.

    Refer to the *OS/390 MVS Planning: APPC/MVS Management* for information on using the **DBRETRIEVE** command provided by the APPC/MVS administration utility to retrieve a database token. If a database token has not been defined, use the **DBMODIFY** command provided by the APPC/MVS administration utility to add a database token to the profile data set.
  - *level* is one of the following:
    - *system* if the transaction program is available to all users who can access the LU.
    - *groupid* if the transaction program is available to a group.
    - *userid* if the transaction program is for a specific user.
  - *tpname* is the name of the transaction program, INFOTP1 in the example on page 28.

  For example, the **APPCTP** profile name for the TP profile added in the example on page 28 would be *dbtoken*.**IMCICS**.**INFOTP1**.
- A list of user IDs who will run the transaction program. These IDs need **EXECUTE** access to the TP.
- A list of user IDs who will need to view the TP. These IDs need **READ** access to the TP.
- A list of user IDs who will create or modify the TP. These IDs need **UPDATE** access to the TP.

Your security administrator must make these modifications on each Tivoli Information Management for z/OS system using a RES. The section about network security in the *OS/390 MVS Planning: APPC/MVS Management* manual contains more information about security, including information about other security classes and profiles you may want to have your security administrator set up.

Starting and Stopping the APPC Environment

The APPC environment must be started to use the server functions. The APPC environment consists of the APPC address space and the APPC scheduler address space. Issue the **START** operator command for the APPC and ASCH address spaces as follows:

- To start the APPC address space with **PARMLIB** member **APPCPMAA**, and to have the master subsystem process the task, enter the statement

  ```
  START APPC,SUB=MSTR,APPC=aa
  ```
- To start the APPC scheduler address space with **PARMLIB** member **ASCHPMAA**, and have the master subsystem process the task, enter the statement

  ```
  START ASCH,SUB=MSTR,ASCH=aa
  ```

After both address spaces are started, when APPC receives a request for a conversation with the TP, **INFOTP1**, it starts a new address space dedicated to the session between the local LU **BLMSRV01** and the client requester's LU. Then it runs **BLMYSCSC**, the program specified in the TP, to access Tivoli Information Management for z/OS through the HLAPI.

To stop the APPC environment, issue the following commands:

```
C APPC
C ASCH
```

Determining Values Clients Need

When you set up the communication links on the clients, you will need the following values:

- LU name for the RES

  For APPC/MVS, this is configured in **SYS1.PARMLIB(APPCPM**xx). The **LUADD** parameter includes an **ACBNAME**. The **ACBNAME** defines an APPC/MVS LU name. This same LU must also be included on an **APPL** definition statement in an **APPL** major node of VTAM. The examples in this chapter define an LU named **BLMSRV01**.

- Control Point (CP) name for the RES

  VTAM does not have a CP name for APPN use. It does have an **SSCPNAME** (system services control point name). If you need a CP name when you configure a client, use the **SSCPNAME** instead. It is defined in the **ATCSTR***xx* member of a data set with the **DDN**name of **VTAMLST** in the VTAM startup procedure. The *xx* is two digits, and the default is 00. **ATCSTR***xx* contains the VTAM startup options.

- Network name

  The **NETID** parameter defines the network name. It is also set in the **ATCSTR***xx* member of the **VTAMLST** data set.

- LAN address

  The **LAN** address is coded in **NCP** on a **LINE** macro using the **LOCADD** parameter.

- TPNAME

  The **TPNAME** is on the **TPNAME** parameter of the **TPADD** statement in the TP profile entry. The value for **TPNAME** must match the one that comes in on a client request to start a conversation. In the example on page 28, the **TPNAME** is **INFOTP1**.

4

Configuring and Running a Multiclient Remote Environment Server (MRES) with APPC

This chapter tells you how to configure a Multiclient Remote Environment Server with APPC (MRES with APPC) on a network that has the following software:

- OS/390 Version 2.5, or a later version
- Tivoli Information Management for z/OS Version 1.2

The configuration described in this chapter is based on a System 390 host connected to a 3745 Communication Controller with a token-ring attachment that serves as a gateway to the rest of the network.

Figure 7 on page 36 illustrates an MRES. The collector in this figure represents an application program that handles several client connections. An example of a collector is the AIX requester.

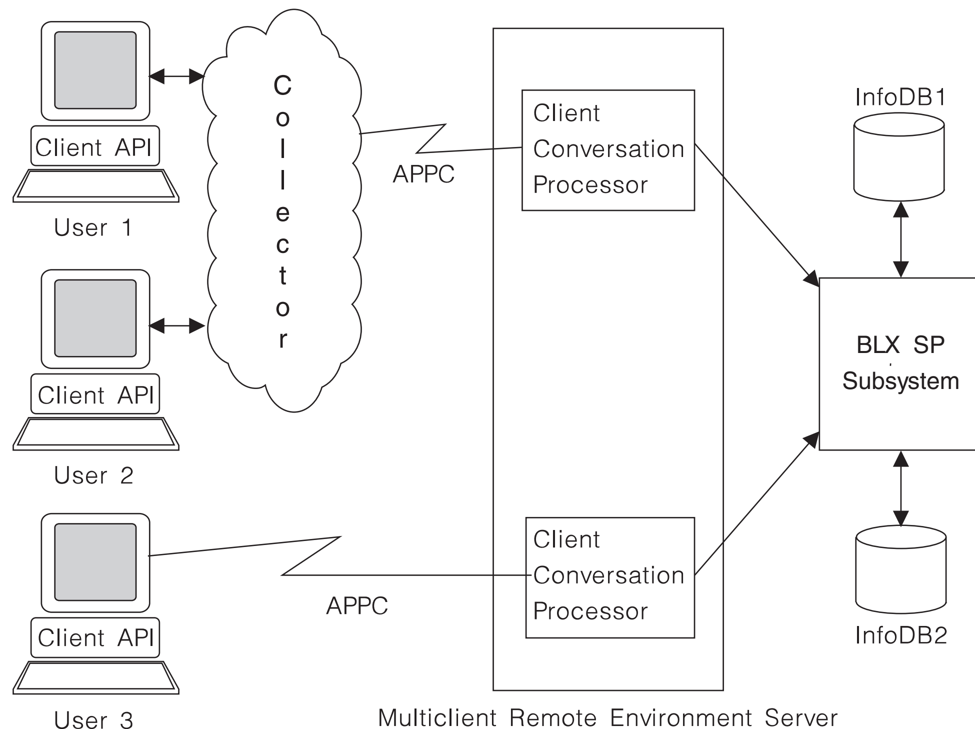


Figure 7. MRES with APPC with Multiple Client Conversation Processors Started

Using an MRES with APPC

The systems programmer creates one or more MRES procedures with APPC cataloged procedures in **SYS1.PROCLIB**. Each procedure contains the MVS JCL required to start the MRES address space. The systems programmer must also create a data set pointed to by the procedure which contains the values for the MRES with APPC parameters.

The systems programmer works with the security administrator and the APPC/APPN communications programmer to determine the appropriate security mechanisms to use for securing the MRES and ensuring that only authorized TPs use the MRES. Then the Tivoli Information Management for z/OS systems programmer works with the APPC/APPN communications programmer to define APPC/MVS resources for the MRES to APPC/MVS and VTAM that implement the security mechanisms. They also work together to define LUs for the clients on the clients' operating systems and to add definitions to NCP to enable routing conversation requests from clients to the MRES with APPC.

While the APPC and VTAM definitions are being done, the security administrator defines the agreed upon security classes and profiles for the APPC and VTAM resources that the communications programmer is defining.

After the cataloged procedure, MRES parameters, APPC, VTAM, and security definitions are complete, the system programmer and the communications programmer verify that each client can establish an LU-to-LU session with the MRES with APPC and exchange data.

The operator starts an MRES with APPC address space with the MVS **START** command. The MRES parameter **APPCDESTNAME** specifies the symbolic destination name with which the MRES registers with APPC/MVS that it is ready to receive client conversations. APPC/MVS creates an allocate queue for the MRES. The MRES parameter **MAXCONNECT** specifies the maximum number of client conversations (connections) that the MRES can process concurrently. Additional MRES address spaces can also be started using the same cataloged procedure.

APPC/MVS monitors the allocate queue for inbound client conversation requests that contain the appropriate symbolic destination name. If the maximum number of conversations for the MRES has not been reached, the conversation is received and the client transactions processed. When a client ends its last or only HLAPI session, the conversation is ended. When the specified maximum number of conversations are active, the MRES does not attempt to access additional conversations from the allocate queue. If another MRES is not started to access this queue, these conversations will wait until the MRES can receive them.

The operator uses the MVS **STOP** command to stop an MRES with APPC address space. Upon receipt of the stop command, a write-to-operator (WTO) message is sent saying how many client connections are still active. Once the **STOP** command is issued, the MRES will not accept any more conversation requests. After a period of time (specified in a startup parameter) expires or when the MRES has no more client connections, the address space ends. Additional WTO messages may be sent at a specified time interval while the MRES is waiting to end.

MRES Configuration Tasks

To configure an MRES, do the following tasks:

1. Plan your configuration. “Planning an MRES with APPC Configuration” on page 38 discusses this task.
2. Define a cataloged procedure for the MRES. “Defining a Procedure for an MRES with APPC” on page 38 discusses this task.
3. Create a data set and specify the MRES parameters. This data set is referenced by the **BLMYPRM DD** statement in the cataloged procedure. “Coding the Parameters for an MRES with APPC” on page 39 describes this task.
4. Add the data sets that contain the MRES load modules to the authorized program facility (APF) library. “Adding the Data Sets to the APF List” on page 46 discusses this task. If the data sets are specified in the link list or if you have them in the link pack area, you can skip this step.
5. Define an LU for the MRES to APPC/MVS. “Defining an MRES with APPC to APPC/MVS” on page 46 discusses this task.
6. Define the MRES LU to VTAM. “Defining an MRES with APPC to VTAM” on page 48 discusses this task.
7. Define classes and profiles in your security product. “Implementing Security” on page 49 discusses this task.

When you are ready to start an MRES with APPC, see “Starting and Stopping an MRES with APPC” on page 49 for the syntax.

Planning an MRES with APPC Configuration

Setting up an MRES with APPC is independent of setting up a client. However, you need to consider which clients will be using the MRES and whether you want them all to use the same MRES or whether you want each client to use a different MRES. Also, the clients will need to know some of the values you declare when you configure the MRES. And you will need to know some of the client’s values when you configure the MRES. See “Choosing a Server” on page 13 for a discussion of other factors you should consider while planning your configuration.

Defining a Procedure for an MRES with APPC

You must define a procedure for an MRES with APPC in SYS1.PROCLIB. The following example lists the JCL statements to include in an MRES with APPC procedure. This JCL is shipped as **BLMMRES** in the **SBLMSAMP** sample library.

```
//*******************************************************************
//*
//*  THIS SAMPLE CATALOGUED PROCEDURE STARTS THE MULTICLIENT REMOTE
//*  ENVIRONMENT SERVER (MRES) ADDRESS SPACE USING THE SAMPLE
//*  PARAMETERS DATA MEMBER.
//*
//*-----------------------------------------------------------------*
//*  UPDATE THE FOLLOWING WITH UPPERCASE TEXT:
//*
//*      FROM                    TO
//* ================================================================
//* BLM.SBLMMOD1         - THE ACTUAL NAME OF THE INFORMATION MANAGEMENT
//*                        FOR Z/OS DATA SET USED AT YOUR INSTALLATION.
//* BLM.SBLMSAMP         - THE NAME OF THE MRES PARAMETERS DATA SET
//*                        USED AT YOUR INSTALLATION.
//* BLM.SBLMTSX           - THE NAME OF THE TSX DATA SET USED AT YOUR
//*                         INSTALLATION.
//* BLMMRESP              - THE NAME OF YOUR MRES PARAMETERS MEMBER IF
//*                         YOU ARE USING A PARTITIONED DATA SET FOR
//*                         YOUR MRES PARAMETERS
//*
//*  STORE THE UPDATED PROCEDURE AS A MEMBER IN YOUR SYSTEM'S PROCEDURE
//*  LIBRARY, SUCH AS SYS1.PROCLIB.
//*
//*******************************************************************
//BLMMRES   PROC PRM=BLMMRESP                        PARAMETER DATA MEMBER
//MRESPGM   EXEC PGM=BLMYSM00,REGION=6M,TIME=1440
//STEPLIB   DD   DISP=SHR,DSN=BLM.SBLMMOD1                  APF AUTHORIZED
//BLMYPRM   DD   DISP=SHR,DSN=BLM.SBLMSAMP(&PRM)            PARAMETER DATA
//BLGTSX    DD   DISP=SHR,DSN=BLM.SBLMTSX                   TSX REXX EXECS
//APIPRINT  DD   SYSOUT=*                                   LLAPI LOG
//HLAPILOG  DD   SYSOUT=*                                   HLAPI LOG
//SYSTSPRT  DD   SYSOUT=*                                   TSX OUTPUT
//          PEND
```

A description of the JCL statements follows:

BLMMRES
This is the name of the instream catalogued procedure. The **PRM=** parameter specifies the name of a data set member that contains the parameters unique to this MRES. It is set to a default value of **BLMMRESP**, the name of a sample MRES parameters member supplied in the SBLMSAMP data set. If a value for **PRM=** is not specified on the

START command when you start the MRES, the values contained in the sample parameter data member **BLMMRESP** are used. To override the value contained in the cataloged procedure, you can specify a **PRM=** on the **START** command.

MRESPGM
This statement specifies the program to be processed, its region size, and how long it can run. The value for **PGM=** must be **BLMYSM00** to call the MRES program. The **REGION=** parameter specifies how much memory to reserve for the MRES program. The larger the region size, the more HLAPI sessions can run concurrently. Declaring **TIME=1440** tells the operating system not to time out the MRES. If this parameter and value are not specified, the MRES will receive an out-of-time abend.

STEPLIB
This statement specifies the program libraries used by the MRES. You must specify the data sets in a **STEPLIB DD** statement if you do not specify them in the link list or if you do not have the load modules in the link pack area. The data sets must be authorized program facility (APF) libraries.

BLMYPRM
This statement specifies the data set or PDS member containing the MRES parameters. Both fixed and variable length record formats are supported.

BLGTSX
This statement points to the PDS (or PDSs) containing the TSX REXX execs.

APIPRINT
This statement specifies the LLAPI log. If any messages about LLAPI transaction activity are generated, they are to be sent to the job's **SYSOUT** class. Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* for more information about these messages.

HLAPILOG
This statement specifies the HLAPI log. If any messages about HLAPI transaction activity are generated, they are to be sent to the job's **SYSOUT** class. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information about these messages.

SYSTSPRT
This **SYSOUT** file can be written to by a TSX.

Coding the Parameters for an MRES with APPC

You must specify the MRES parameters in the data set or PDS member that you specify on the **BLMYPRM DD** statement. A sample that you can use for your MRES parameters is shipped as **BLMMRESP** in the **SBLMSAMP** library; this is described in "Sample MRES Parameters" on page 45.

Several basic rules must be followed when you code the parameter data:

- Comments must begin with /* and end with */
- Comments can be in any column between 1 and 72, inclusive
- Nothing (except comments) can be in column 1
- Nothing can be in any column greater than column 72
- Begin the parameters data with a statement identifier of **BLMYPRM**

- You can separate parameters with a comma. Parameters need not appear on separate lines
- The final parameter should end with a semicolon

These are the MRES parameters that you can specify in your **BLMYPRM** data set:

```
[APMINSTRUMENT={OFF|ON}]
[MAXCONNECT={10|n}]
SHUTDOWNTRY=hhmmss
SHUTDOWNWT=hhmmss
[WRITEOPER={1|code}]
COMMTYPE=APPC
APPCDESTNAME=id
[PRESTARTSESSIONS={NO|YES}]
[APPLICATION_ID=id]
[PRIVILEGE_CLASS=id]
[SESSION_MEMBER=session member name]
[APIMSG_OPTION={C|P|B}]
[BYPASS_PANEL_PROCESSING={NO|YES}]
[CLASS_COUNT={0|n}]
[DATABASE_ID={5|n}]
[DEFAULT_DATA_STORAGE_SIZE={1024|n}]
[DATE_FORMAT={DATABASE|pattern}]
[DEFAULT_OPTION={ALL|REQUIRED|NONE}]
[HLIMSG_OPTION={C|P|B}]
[MULTIPLE_RESPONSE_FORMAT={SEPARATOR|PHRASE}]
[PDB_TRACE={NO|YES}]
[SPOOL_INTERVAL={0|n}]
[TABLE_COUNT={0|n}]
[TIMEOUT_INTERVAL={300|n}]
```

The values that you can specify for these parameters are as follows:

APMINSTRUMENT={OFF|ON}

This parameter specifies the APM Instrumentation flag. A specification of **ON** indicates that the MRES should generate heartbeats or pulses to allow Tivoli Global Enterprise Manager (GEM) Application Policy Manager (APM) to monitor the status of the MRES. To have pulses generated, specify **APMINSTRUMENT=ON**. If a value is not specified, the default value is **OFF**. This parameter is optional. This parameter is provided for those users who are using Tivoli Global Enterprise Manager (GEM) to manage Tivoli Information Management for z/OS in a Tivoli Management Environment. The *Tivoli Information Management for z/OS Guide to Integrating with Tivoli Applications* contains additional information about using the Tivoli Global Enterprise Manager.

If you are not using Tivoli Information Management for z/OS in a Tivoli GEM environment, you can omit this parameter.

MAXCONNECT={10|n}

This parameter specifies the maximum number of client conversation processors this MRES can run concurrently. Valid values are from 1 to 50. The default value is 10. This parameter is optional.

SHUTDOWNTFY=hhmmss

A required parameter that specifies the time interval between shutdown notification messages. The value specified indicates the amount of time between operator notification messages that follow the first message. When the operator issues the MVS **STOP** command, the MRES sends a message to the operator indicating the number of conversations that are still connected. After this first notification message is sent, notification messages are sent at the interval specified on this parameter. For example, if you specify a value of *000000*, you will receive only one notification message. If you

specify a value of *000500*, you will receive the first notification message and additional messages at 5-minute intervals until either the **SHUTDOWNTFY** time expires or all the conversations have ended.

You must specify this interval as **HHMMSS** where
HH Hours
MM Minutes
SS Seconds

SHUTDOWNWT=hhmmss
A required parameter that specifies the shutdown wait time period. The value specified indicates a period of time that the MRES is to continue processing after receiving the MVS **STOP** command. This time period permits client conversations to complete any processing that was active when the operator issued the **STOP** command. Additional conversations are not accepted during this time, but those already connected can continue until the time expires. If no clients are connected when the **STOP** command is issued, the MRES stops processing immediately regardless of the interval specified on this parameter. Also, when all the conversations have stopped, the MRES stops processing immediately.

You must specify this interval as **HHMMSS** where
HH Hours
MM Minutes
SS Seconds

WRITEOPER={1|code}
This parameter specifies the default write-to-operator (WTO) routing code. Valid values are from 1 to 128. If not specified, the default value is 1. All WTOs that are not a result of command responses are automatically routed to this code. To determine the routing codes for a console, you can do one of the following:

- Display console characteristics by issuing the `DISPLAY CONSOLES,A` command from the console.
- Review the **ROUTCODE** parameter of the **CONSOLE** statements in the **CONSOL.***xx* member of **SYS1.PARMLIB**.

Refer to *OS/390 Operations: System Commands* for more information about consoles and routing codes. This parameter is optional.

COMMTYPE=APPC
A required parameter that specifies that APPC communications protocol is to be used.

APPCDESTNAME=id
Specifies the APPC symbolic destination name. When **COMMTYPE=APPC** is specified, this is a required parameter. The id you specify indicates the symbolic destination name that represents the transaction program and local LU to be registered. This name must match a symbolic destination name defined in the active side information data set. The name can be from 1 to 8 characters in length and composed of the uppercase letters **A** through **Z** and the numerals **0** through **9**.

PRESTARTSESSIONS={NO|YES}
Pre-start API sessions indicator. Set this to **YES** to specify that the API sessions are to be pre-started. If you specify **NO** or omit this parameter, the API sessions are not pre-started. This parameter is optional.

APPLICATION_ID=id
Contains a 1- to 8-character uppercase application ID that Tivoli Information

Management for z/OS uses for this session. The application ID is specified on the HL01 transaction and can be specified on many other HLAPI transactions, so it can vary over the life of the HLAPI session. The ID must be an eligible user of the privilege class being used. This keyword is required when **PRESTARTSESSIONS=YES** is specified; otherwise, it is optional.

PRIVILEGE_CLASS=id
Contains a 1- to 8- byte privilege class name, which can contain DBCS characters enclosed by an SO/SI pair. A privilege class remains in effect until your application specifies a different privilege class name. An application can specify an initial privilege class that grants all authority required for the duration of the Tivoli Information Management for z/OS session. This keyword is required when **PRESTARTSESSIONS=YES** is specified; otherwise, it is optional.

SESSION_MEMBER=session member name
Contains a 7- or 8-character load library session parameter member name that Tivoli Information Management for z/OS uses for this session. Session member names begin with the character string **BLGSES** and cannot contain imbedded blanks. This keyword is required when **PRESTARTSESSIONS=YES** is specified; otherwise, it is optional.

APIMSG_OPTION={C|P|B}
Specify a 1-character LLAPI message option parameter **P**, **C**, or **B**.

- A value of **P** specifies that the LLAPI writes messages to the **APIPRINT** data set.
- A value of **C** specifies that the LLAPI chains messages and passes them from the LLAPI to the HLAPI for conversion into message PDBs.
- A value of **B** specifies that the LLAPI performs both **P** and **C**.

This parameter is optional and is used only if **PRESTARTSESSIONS=YES**. This parameter is used only if **SPOOL_INTERVAL** is specified and is not set to zero. If you omit this parameter, the LLAPI performs option **C**.

BYPASS_PANEL_PROCESSING={NO|YES}
Bypass panel processing indicator. Set this to **YES** to specify that no panels be used in record processing other than those used by the delete transaction. If you specify **NO**, the HLAPI uses panel processing mode.

If you specify **BYPASS_PANEL_PROCESSING = YES**, you must use data model records for the following transactions:

- HL08 Create record
- HL09 Update record
- HL12 Add record relation

This parameter is optional and is used only if **PRESTARTSESSIONS=YES**.

CLASS_COUNT={0|n}
Contains a character number that indicates the maximum number of Tivoli Information Management for z/OS privilege class records that can be maintained in storage during the life of this Tivoli Information Management for z/OS session. If you omit this parameter or enter zero as its value, the Tivoli Information Management for z/OS session operates with a single privilege class record in storage at a time. This parameter is optional and is used only if **PRESTARTSESSIONS=YES**.

DATABASE_ID={5|n}
A character number containing a 1-character ID number of the database to be used. For

Tivoli Information Management for z/OS records, the database ID number is 5. If you omit this parameter, the HLAPI automatically sets the database ID to 5. This parameter is optional and is used only if **PRESTARTSESSIONS=YES**.

DEFAULT_DATA_STORAGE_SIZE={1024|n}
Contains a character number specifying how much additional storage is allocated to hold default response data from an alias table when your application is creating records. When the HLAPI creates records, it calculates the size of the response buffer it needs by totaling the lengths of all the input data PDBs and adding the specified default data storage size. If you omit the default data storage size, the HLAPI adds a default of 1024 bytes. When the HLAPI performs create response processing, it always checks to make sure the response will not overlay storage. If the response will overlay storage, the HLAPI transaction will end with an error code. You use this parameter with the **DEFAULT_OPTION** parameter. This parameter is optional and is used only if **PRESTARTSESSIONS=YES**.

DATE_FORMAT={DATABASE|pattern}
Contains a character field that specifies how your application uses dates. If the value **DATABASE** is specified, the default format from the database is assumed. If you omit this parameter, the value **DATABASE** is assumed. Valid values are:

MM/DD/YY
MM-DD-YY
MM.DD.YY
MM/DD/YYYY
MM-DD-YYYY
MM.DD.YYYY
DD/MM/YY
DD-MM-YY
DD.MM.YY
DD/MM/YYYY
DD-MM-YYYY
DD.MM.YYYY
YY/MM/DD
YY-MM-DD
YY.MM.DD
YYYY/MM/DD
YYYY-MM-DD
YYYY.MM.DD
DDMMMYY
DDMMMYYYY
YYDDD
YYYYDDD

This parameter is optional and is used only if **PRESTARTSESSIONS=YES**.

DEFAULT_OPTION={ALL|REQUIRED|NONE}
Contains a character field that specifies how the HLAPI performs create default data response processing in this session. The valid data values for **DEFAULT_OPTION** are **ALL**, **REQUIRED**, and **NONE**. **ALL** specifies that all response fields specified in a PIDT are candidates for default responses. **REQUIRED** specifies that only required fields are candidates for default responses. The HLAPI does not perform default response processing if you omit this field or specify it as **NONE**. You can override the initial default processing option when creating records by respecifying the default option on the control chain. After the create transaction completes, the HLAPI reverts to the

initial default specification for record creation unless overridden again. This parameter is optional and is used only if **PRESTARTSESSIONS=YES**. If you omit this parameter, the value NONE is assumed.

HLIMSG_OPTION={C|P|B}
Contains a 1-character HLAPI message option parameter **P**, **C**, or **B**.

- A value of **P** specifies that the HLAPI writes messages to the **HLAPILOG** data set.
- A value of **C** specifies that the HLAPI chains messages on the PDB message chain.
- A value of **B** specifies that the HLAPI performs both P and C.

If you omit this parameter, then the HLAPI performs option **C**. The HLAPI writes messages passed back from the LLAPI to the **HLAPILOG** data set. This parameter is used only if **SPOOL_INTERVAL** is specified and is not set to zero. This parameter is optional and is used only if **PRESTARTSESSIONS=YES**.

MULTIPLE_RESPONSE_FORMAT={SEPARATOR|PHRASE}
A specification of **PHRASE** permits you to use spaces to separate multiple response fields. The default value, **SEPARATOR**, requires that a separator character be specified in the control PDB named **SEPARATOR_CHARACTER** on the HLAPI transaction submitted by the client API application. This parameter is optional and is used only if **PRESTARTSESSIONS=YES**.

PDB_TRACE={NO|YES}
This parameter specifies whether the HLAPI should perform PDB data tracing for debugging purposes or PDB data logging to the HLAPILOG output file. Setting this parameter value to **YES** causes the logging of up to 32 bytes of PDBDATA information for each PDB used throughout the session. A value of **NO** causes no PDB logging to be performed. This parameter is optional and is used only if **PRESTARTSESSIONS=YES**. If you omit this parameter, the value **NO** is assumed.

SPOOL_INTERVAL={0|n}
Contains a character specifying the number of minutes that the HLAPI spools the activity logs **HLAPILOG** and **APIPRINT** when messages are printed. If the HLAPI is spooling to a data set and this time interval has passed, the activity logs are recycled and new log information is written starting at the top of the data set, writing over any existing information. If you omit this parameter, the HLAPI does not log messages and the settings in **APIMSG_OPTION** and **HLIMSG_OPTION** are ignored. This parameter is optional and is used only if **PRESTARTSESSIONS=YES**.

TABLE_COUNT={0|n}
Contains a character value that indicates the maximum number of alias tables and PIDTs and anchored PIPTs that the HLAPI can maintain in storage during the life of a Tivoli Information Management for z/OS session. Static PIDTs and PIDTs generated from data view records are treated the same for caching purposes. It can take a significant amount of time to generate a PIDT from data view records. The length of time depends on the number of data attribute records (and validation records they reference) contained in the data view record. Therefore, it can be especially important to direct the HLAPI to maintain PIDTs in storage if you are using data models. If you specify this value as zero or omit it, the Tivoli Information Management for z/OS session will not process **ALIAS_TABLE** parameters or cache PIDTs. Alias table and PIDT processing can increase transaction run time due to the increased I/O time of loading and unloading tables. By balancing the table count to alias table and PIDT usage, you can reduce to

zero the additional I/O overhead for long-running applications. This parameter is optional and is used only if **PRESTARTSESSIONS=YES**.

TIMEOUT_INTERVAL={300|n}
Contains a character value specifying the number of seconds that a transaction can run before a timer interrupt occurs. If you specify a value between 0 and 45 seconds, the HLAPI uses a value of 45 seconds. If you specify a value of 0 or omit this parameter, the HLAPI uses a default value of 300 seconds (five minutes). This parameter is optional and is used only if **PRESTARTSESSIONS=YES**.

Sample MRES Parameters

A sample of the MRES parameters is provided in **BLMMRESP** in the **SBLMSAMP** data set. If you rename this member or create a new one, be sure to change the member name in the **PRM=** parameter in the JCL for your MRES with APPC procedure or be sure to specify **PRM=** on the **START** command when you start the MRES with APPC procedure. Refer to “Defining a Procedure for an MRES with APPC” on page 38 for an example of the JCL. The following are sample parameters for an MRES with APPC.

```
/*********************************************************************/
/*                                                                   */
/*              MRES START UP PARAMETERS                             */
/*                                                                   */
/*********************************************************************/

 BLMYPRM                         /* SPECIFY MRES PARAMETERS          */

   /******************************************************************/
   /*                                                                */
   /*     PARAMETERS TO CONTROL THE GENERAL MRES SESSION             */
   /*                                                                */
   /******************************************************************/

   APMINSTRUMENT=OFF,          /* APM INSTRUMENTATION (ON OR OFF)    */
   MAXCONNECT=10,              /* MAXIMUM NUMBER OF CONNECTIONS      */
   SHUTDOWNTFY=000200,         /* SHUTDOWN NOTIFY TIME (HHMMSS)      */
   SHUTDOWNWT=000500,          /* SHUTDOWN WAIT TIME (HHMMSS)        */
   WRITEOPER=1,                /* WRITE-TO-OPERATOR ROUTING CODE     */

   /******************************************************************/
   /*                                                                */
   /*     PARAMETERS TO CONTROL THE COMMUNICATIONS SESSION           */
   /*                                                                */
   /******************************************************************/

   COMMTYPE=APPC,              /* COMMUNICATIONS (APPC OR TCPIP)     */
   APPCDESTNAME=XXXXXXXX,      /* APPC SYMBOLIC DESTINATION NAME     */

   /******************************************************************/
   /*                                                                */
   /*     PARAMETERS TO CONTROL THE PRE-STARTING OF API SESSIONS     */
   /*                                                                */
   /******************************************************************/

   PRESTARTSESSIONS=NO,             /* PRE-START (YES OR NO)         */
   APPLICATION_ID=XXXXXXXX,         /* APPLICATION ID                */
   PRIVILEGE_CLASS=XXXXXXXX,        /* PRIVILEGE CLASS NAME          */
   SESSION_MEMBER=BLGSESXX,         /* SESSION MEMBER NAME           */
   APIMSG_OPTION=C,                 /* LLAPI MESSAGES (C, P, OR B)   */
   BYPASS_PANEL_PROCESSING=NO,      /* BYPASS PANELS (YES OR NO)     */
   CLASS_COUNT=0,                   /* MAXIMUM NUMBER TO BE CACHED   */
   DATABASE_ID=5,                   /* DATABASE ID                   */
   DEFAULT_DATA_STORAGE_SIZE=1024,  /* DEFAULT SIZE IN BYTES         */
   DATE_FORMAT=DATABASE,            /* FORMAT OF DATE DATA           */
```

```
DEFAULT_OPTION=NONE,             /* (ALL, REQUIRED, OR NONE)      */
HLIMSG_OPTION=C,                 /* HLAPI MESSAGES (C, P, OR B)   */
MULTIPLE_RESPONSE_FORMAT=SEPARATOR,  /* (PHRASE OR SEPARATOR)     */
PDB_TRACE=NO,                    /* PDB DEBUG TRACE (YES OR NO)   */
SPOOL_INTERVAL=0,                /* MAXIMUM TIME IN MINUTES       */
TABLE_COUNT=0,                   /* MAXIMUM TABLE ENTRIES         */
TIMEOUT_INTERVAL=300;            /* MAXIMUM TIME IN SECONDS       */
```

Adding the Data Sets to the APF List

If you neither specified the data sets that contain the MRES load modules in the link list nor have them in the link pack area, you must add the data sets to the **APF** library.

To define the data sets that contain the load modules as **APF** libraries, make an entry for each data set in the appropriate **PROG***xx* members of **SYS1.PARMLIB**. Each entry in a **PROG***xx* member includes the data set name (*dsn*) and the volume serial number (*volser*) of the library.

The changes will be activated the next time you IPL the system, or you can use the MVS operator console command **SET PROG=xx** to dynamically activate the changes.

Defining an MRES with APPC to APPC/MVS

To define an MRES to APPC/MVS, do the following tasks:

1. Create a VSAM data set for the CPI-C side information if the MVS system does not already have one.
2. Add an entry to the side information file for the MRES LU.
3. Define an MRES LU in **APPCPM***xx* of **SYS1.PARMLIB**.

You do not have to define a class in **ASCHPM***xx* because classes define scheduling information and the MRES bypasses the transaction scheduler. For the same reason, you do not need to define a TP profile. If you plan to use the **APPCSERV** class provided by RACF for security, verify that a TP profile data set exists.

Creating a VSAM Data Set for the Side Information

If you do not have a VSAM data set for the CPI-C side information, use IDCAMS to create one. APPC/MVS uses this data set to translate symbolic destination names. You must define this file first, before you reference it in **SYS1.PARMLIB** when you define local LUs.

If you already have a VSAM data set for CPI-C side information, add the side information entry for the MRES to that file. APPC/MVS allows only one active side information data set on an MVS system.

You can use the JCL in this example to create a VSAM data set for the CPI-C side information. Change *volname* to the name of your local volume. Change *sidefile* to the name you want to use for your local side information data set.

```
//jobname  JOB your-job-card
//JOBPARM LINES=9999,TIME=1440
//SISAMPLE EXEC PGM=IDCAMS
//volname  DD   DISP=OLD,UNIT=3380,VOL=SER=volname
//SYSPRINT DD   SYSOUT=*
//SYSABEND DD   SYSOUT=*
//AMSDUMP  DD   SYSOUT=*
//SYSIN    DD   *
```

```
        DEFINE CLUSTER (NAME(sidefile) - /*side info data set name = SYS1.APPCSI */
             VOLUMES(volname) -
             INDEXED REUSE -
             SHAREOPTIONS(3 3) -
             RECORDSIZE(248 248) -
             KEYS(112 0) -
             RECORDS(50 25)) -
           DATA -
             (NAME(sidefile.DATA)) -  /*side info data set name = SYS1.APPCSI */
           INDEX -
             (NAME(sidefile.INDEX)) - /*side info data set name = SYS1.APPCSI */
/*
```

Adding a Side Information Entry for an MRES

Because the MRES uses a nonscheduled LU, it does not require a transaction program profile. (Transaction program profiles contain scheduling information.) For nonscheduled LUs, APPC/MVS searches the side information file for a **DESTNAME** that matches the one in the request from a client to initiate a conversation. If a match is found, the conversation request is routed to the LU specified on the **PARTNER_LU** parameter of that side information entry.

You can modify the JCL shown in the following example to add an entry to the side information file.

```
//jobname  JOB your-job-card
//STEP     EXEC PGM=ATBSDFMU
//SYSPRINT DD   SYSOUT=*
//SYSSDLIB DD   DSN=SYS1.APPCSI,DISP=SHR
//SYSSDOUT DD   SYSOUT=*
//SYSIN    DD   *
     SIADD
          DESTNAME(symbolic_destination_name)
          TPNAME(tpname)
          MODENAME(mode)
          PARTNER_LU(mres_lu)
/*
```

Set the parameters in the side information entry as follows:

DESTNAME
: Specifies the symbolic destination name for the MRES. Use a symbolic destination name you will use on the **APPCDESTNAME** parameter in the MRES parameters data set.

TPNAME
: Specifies the name of the incoming TP that can use the associated **DESTNAME**. *tpname* must match the value coming from the client.

MODENAME
: Specifies the name of a compiled log-on mode to use for the conversation. The log-on mode is an entry in **SYS1.VTAMLIB**. See “Defining the Log-on Mode” on page 30 for information on defining a mode.

PARTNER_LU
: Specifies the name of a nonscheduled LU that is defined in the **APPCPM***xx* member of **SYS1.PARMLIB** on this MVS system. This LU is on the same MVS system as this side information file. The example in “Defining a Nonscheduled APPC/MVS Logical Unit” on page 48 shows the statements to use to define a nonscheduled LU.

You must have a separate side information entry for each symbolic destination name you use when you start an MRES procedure. If this MRES is allowed to serve more than one TP, you must also have a separate side information entry for each unique TPname.

Defining a Nonscheduled APPC/MVS Logical Unit

You must define one or more nonscheduled APPC/MVS LUs to serve as partner LUs for your client's transaction programs. The definitions go in member **APPCPM***xx* of **SYS1.PARMLIB**.

The following example shows the statements to define a nonscheduled LU to APPC. Only two of the statements shown are required, **LUADD ACBNAME**(*mres_lu*) and **NOSCHED**. The remaining statements default to the values shown. Because the MRES does not use a TP profile, the only **TPLEVEL** available is **SYSTEM**, the default.

```
LUADD  ACBNAME(mres_lu)
       NOSCHED                   /* Declares the LU as nonscheduled
       TPDATA(SYS1.APPCTP)       /* This is the APPC default
       TPLEVEL(SYSTEM)           /* This is the APPC default
SIDEINFO DATASET(SYS1.APPCSI)    /* Specifies that VSAM data set
                                 /* SYS1.APPCSI is the CPI-C
                                 /* side information file
                                 /* This is the APPC default
```

Defining an MRES with APPC to VTAM

The logical units that you defined to APPC/MVS must also be defined to VTAM in a **VTAMLST** library. This example illustrates the statements needed to define an LU for an MRES to VTAM.

```
APPCAPP VBUILD TYPE=APPL
mres_lu  APPL ACBNAME=mres_lu,APPC=YES,AUTOSES=1,
                    DDRAINL=NALLOW,DMINWNL=1,DMINWNR=1,DRESPL=NALLOW,DSESLIM=20,EAS=509,
                    MODETAB=modetab,SECACPT=security,VPACING=0,VERIFY=NONE
                    SRBEXIT=YES,DLOGMOD=mode
```

Substitute the variables in the example as follows:

ACBNAME
: Specifies the name of the LU for the MRES. This value must be the same as the value you declared for **ACBNAME** on the **LUADD** statement and **PARTNER_LU** on the **SIADD** statement.

MODETAB
: Specifies the name of the VTAM log mode table, for example **ISTINCLM**, that contains the definition for the value you declare for *mode* on the **DLOGMOD** parameter.

DLOGMOD
: Specifies the name of a compiled logon mode to use for the conversation. This log-on mode is in the VTAM log mode table declared on the **MODETAB** parameter. See "Defining the Log-on Mode" on page 30 for information on defining a mode.

SECACPT
: Specifies whether the client must send a password when requesting a conversation. Use **ALREADYV** if you are defining an LU for a CICS client to use. Use **CONV** if you are defining an LU for an OS/2, an AIX, or a Windows NT client to use.

: The following documents contain additional information about security and defining LUs to VTAM:

OS/390 MVS Planning: APPC/MVS Management
VTAM Resource Definition Reference
OS/390 Security Server (RACF) Planning: Installation and Migration

Implementing Security

Define the security profiles needed to implement the APPC security mechanisms your LU definitions require. Refer to your security product's documentation for information on creating the definitions.

Starting and Stopping an MRES with APPC

APPC/MVS and the BLX-SP server must be running before you start an MRES with APPC. The APPC/MVS scheduler does not have to be running because the MRES does not use it.

To start an MRES, issue the MVS system operator **START** command. To stop an MRES, issue the MVS system operator **STOP** command. For full descriptions of the MVS **START** and **STOP** operator commands, refer to *OS/390 Operations: System Commands* The **START** command accepts the parameters listed in the following section.

START Command Syntax

The syntax of the MVS system operator **START** command for an MRES with APPC is as follows:

```
S blmmres[,JOBNAME=job_name][,PRM=member_name]
```

S The MVS system operator **START** command.

blmmres
The name of the cataloged procedure for this MRES with APPC.

Note: The name of the cataloged procedure becomes the name of the job unless it is overridden by the **JOBNAME=** parameter.

JOBNAME=*job_name*
The value assigned to this keyword becomes the name of the newly started MRES. If this parameter is not specified, the name of the cataloged procedure becomes the name of the job.

Note: For the MRES with APPC, multiple MRESs can be started with the same or different job names and with either the same or different parameter data.

PRM=*MRES_parameters_member_name*
In the sample procedure **BLMMRES** on page "Defining a Procedure for an MRES with APPC" on page 38), **PRM=** specifies the name of the data set member that contains the MRES parameters. If you specify **PRM=** on the **START** command, the MRES parameters in that member are used. They override the values contained in the MRES parameters data set member referenced by **PRM=** in the catalogued procedure for this MRES with APPC. If **PRM=** is not specified on the **START** command, the MRES parameters contained in the data set member referenced by the cataloged procedure are used. Refer to "Defining a Procedure for an MRES with APPC" on page 38 for a sample of the cataloged procedure. The sample uses the MRES parameters in **SBLMSAMP** data set member **BLMMRESP**.

STOP Command Syntax

The syntax of the MVS system operator **STOP** command for an MRES with APPC is as follows:

```
P job_name[,A=n]
```

P The MVS system operator **STOP** command.

job_name
The name of the cataloged procedure for this MRES with APPC.

Note: Note that **job_name** is the name of the cataloged procedure that was started, unless the parameter **JOBNAME** was specified on the **START** command.

A=*n*
An optional parameter that specifies the address space number consisting of from 1 to 4 hexadecimal digits (0–F). You can obtain this number in several ways:

- Use the Display Jobs operator command (*D J,BLMMRES* displays information about all jobs with the name **BLMMRES**).
- Look for **ASID=*n*** in message **BLM03170I**. This message is written to the console when the MRES is successfully started.
- Use other display commands. (See “Using MVS Operator Commands” on page 67 for information about the **DISPLAY** command.)

If you do not use this parameter, all jobs with the name **job_name** are stopped. If you do use this parameter, only the job you specify is stopped.

Determining Values Clients Need

When you set up the communication links on the clients, you will need the following values:

- LU name for MRES with APPC

  For APPC/MVS, this is configured in **SYS1.PARMLIB(APPCPM*xx*)**. The **LUADD** parameter includes an **ACBNAME**. The **ACBNAME** defines an APPC/MVS LU name. This same LU must also be included on an **APPL** definition statement in an APPL major node of VTAM. This is the value you specified for *mres_lu*.

- Control Point (CP) name for MRES

  VTAM does not have a CP name for APPN use. It does have an **SSCPNAME** (system services control point name). If you need a CP name when you configure a client, use the **SSCPNAME** instead. It is defined in the **ATCSTR*xx*** member of a data set with the **DDN**name of **VTAMLST** in the VTAM startup procedure. The *xx* is two digits, and the default is 00. **ATCSTR*xx*** contains the VTAM startup options.

- Network name

  The **NETID** parameter defines the network name. It is also set in the **ATCSTR*xx*** member of the **VTAMLST** data set.

- LAN address

  The LAN address is coded in **NCP** on a **LINE** macro using the **LOCADD** parameter.

- Symbolic destination name

  The **DESTNAME** parameter on the **SIADD** statement you use to add side information for the MRES defines the symbolic destination name. (This value also matches the

APPCDESTNAME value used when starting the MRES with APPC cataloged procedure.) In the JCL example on page “Adding a Side Information Entry for an MRES” on page 47, the symbolic destination name is the value you declare for *symbolic_destination_name*.

- TPname

  The **TPNAME** parameter on each **SIADD** statement identifies a TP that can be served by the MRES associated with that entry. In the JCL example on page 47, the **TPname** is the value you declare for *tpname*.

5

Configuring and Running a Multiclient Remote Environment Server (MRES) with TCP/IP

This chapter tells how to configure a Multiclient Remote Environment Server with TCP/IP (MRES with TCP/IP) on a network that has the following software:

- OS/390 Version 2.5, or a later version
- IBM® Transmission Control Protocol (TCP/IP) for MVS Version 3 Release 2 (5655–HAL) with PTF UN98840, or equivalent.
- Tivoli Information Management for z/OS Version 1.2

You may need other software, depending on the configuration of your network.

Figure 8 on page 54 illustrates an MRES with TCP/IP.

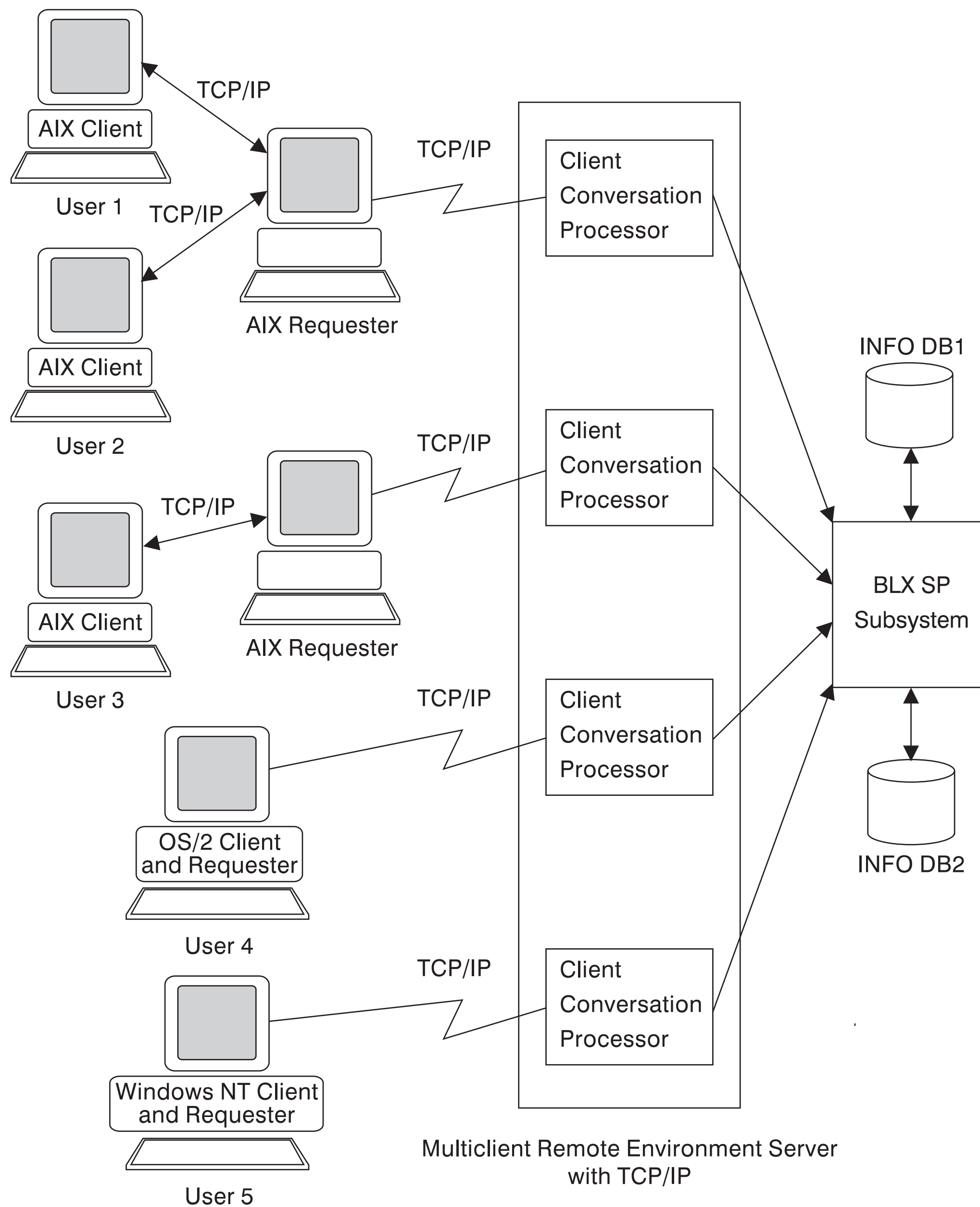


Figure 8. MRES with TCP/IP and Multiple Client Conversation Processors Started

Using an MRES with TCP/IP

The systems programmer creates one or more MRES with TCP/IP cataloged procedures in **SYS1.PROCLIB**. Each procedure contains the MVS JCL required to start the MRES with TCP/IP address space. The systems programmer must also create a data set pointed to by the procedure which contains the values for MRES with TCP/IP startup parameters.

The systems programmer works with the security administrator and the TCP/IP communications programmer to determine the appropriate security mechanisms to use for securing the MRES with TCP/IP and the transaction programs it serves. The Tivoli Information Management for z/OS systems programmer then works with the TCP/IP communications programmer to define resources for the MRES with TCP/IP.

While the TCP/IP definitions are being done, the security administrator defines the agreed upon security classes and profiles. For example, when using RACF as the security product with OS/390 UNIX System Services (OS/390 UNIX System Services was previously called OpenEdition®), you need to add the MRES started procedure name to the **RACF STARTED** class, specifying the following information for **STDATA**:

USER= specify a userid that has an OMVS segment.

Note: You may consider using an OMVS default user. In the *OpenEdition Planning Guide*, see the section entitled "Setting Up Default OMVS Segments" for additional information on creating an OMVS default user.

TRUSTED*(NO)*

PRIVILEGED*(YES)*

The commands to perform this are:

```
RDEFINE STARTED mres STDATA(USER(OEDFLT) TRUSTED(NO) PRIVILEGED(YES)

SETROPTS RACLIST(STARTED) REFRESH
```

where `mres` is the name of the started procedure. In a RACF environment, the first **SETROPTS** command causes the OEDFLT user ID to be assigned to a started task which has a name of `mres`. The second **SETROPTS** command performs a refresh of the RACF list.

After the cataloged procedure, MRES parameters, TCP/IP, and security definitions are complete, the system programmer and the communications programmer verify that each client user ID can communicate with the MRES with TCP/IP and exchange data.

The operator starts an MRES with TCP/IP address space with the MVS **START** command, described in "START Command Syntax" on page 65. The MRES parameter **PORT** specifies the unique port number for TCP/IP to use to communicate with the MRES address space. The MRES parameter **IPADDRESS** specifies the IP address used to access the MRES address space. The MRES parameter **MAXCONNECT** specifies the maximum number of client conversations (connections) that the MRES can process concurrently. Additional address spaces for MRES for TCP/IP communication can also be started. Each address space must have a unique IP address and port number combination.

Note: You can start multiple MRESs using the same procedure. You will need to create an MRES parameters member for each **PORT** number you plan to use because each MRES must be started with a unique **PORT** number. To start an MRES using the same procedure, specify the name of the procedure on the **START** command along with the **PRM=** parameter identifying the particular MRES parameters member that should be used for that MRES. The name of an MRES address space is the same name as that of the catalogued procedure that started it unless the **JOBNAME=** parameter is specified.

When the MRES address space receives a communication request, it grants the request provided it has not reached its maximum number of communication connections. When a client ends its last or only HLAPI session, the communication connection is ended. If the MRES receives a request and all its communication connections are being used, the request is rejected.

The operator uses the MVS **STOP** command to stop an MRES address space. Upon receipt of the stop command, a write-to-operator (WTO) message is sent saying how many client connections are still active. After the **STOP** command is issued, the MRES does not accept any more connection requests. After a period of time (specified in a startup parameter) expires or when the MRES has no more client connections, the address space ends. Additional WTO messages may be sent at a specified time interval while the MRES is waiting to end.

MRES Configuration Tasks

To configure an MRES with TCP/IP, do the following tasks:

1. Plan your configuration. "Planning Your MRES with TCP/IP Configuration" discusses this task.
2. Define a cataloged procedure for the MRES with TCP/IP. "Defining a Procedure for an MRES with TCP/IP" discusses this task.
3. Create a data set and specify the MRES parameters. This data set is referenced by the **BLMYPRM DD** statement in the cataloged procedure; "Coding the Parameters for an MRES with TCP/IP" on page 58 describes this task.
4. Add the data sets that contain the MRES load modules to the authorized program facility (APF) library. "Adding the Data Sets to the APF List" on page 65 discusses this task. If the data sets are specified in the link list, or if you have them in the link pack area, you can skip this step.
5. Implement your security plan. Refer to your security product's documentation for information on this task.

When you are ready to start an MRES with TCP/IP, see "Starting and Stopping an MRES with TCP/IP" on page 65 for the syntax.

Planning Your MRES with TCP/IP Configuration

Setting up an MRES is independent of setting up a client. You need to consider which clients will be using the MRES and whether you want them all to use the same MRES or whether you want each client to use a different MRES. You will need to tell the clients what TCP/IP port numbers and addresses to use. See "Choosing a Server" on page 13 for a discussion of other factors you should consider while planning your configuration.

Defining a Procedure for an MRES with TCP/IP

You must define a procedure for an MRES with TCP/IP in **SYS1.PROCLIB**. This example lists the JCL statements to include in an MRES with TCP/IP procedure. This JCL is shipped as **BLMMRES** in the **SBLMSAMP** sample library.

```
//*******************************************************************
//*
//*  THIS SAMPLE CATALOGUED PROCEDURE STARTS THE MULTICLIENT REMOTE
//*  ENVIRONMENT SERVER (MRES) ADDRESS SPACE USING THE SAMPLE
//*  PARAMETERS DATA MEMBER.
//*
//*-----------------------------------------------------------------*
//*  UPDATE THE FOLLOWING WITH UPPERCASE TEXT:
//*
//*      FROM                    TO
//* ================================================================
```

```
//* BLM.SBLMMOD1        - THE ACTUAL NAME OF THE INFORMATION MANAGEMENT
//*                       FOR Z/OS DATA SET USED AT YOUR INSTALLATION.
//* BLM.SBLMSAMP        - THE NAME OF THE MRES PARAMETERS DATA SET
//*                       USED AT YOUR INSTALLATION.
//* BLM.SBLMTSX         - THE NAME OF THE TSX DATA SET USED AT YOUR
//*                       INSTALLATION.
//* BLMMRESP            - THE NAME OF YOUR MRES PARAMETERS MEMBER IF
//*                       YOU ARE USING A PARTITIONED DATA SET FOR
//*                       YOUR MRES PARAMETERS
//*
//*  STORE THE UPDATED PROCEDURE AS A MEMBER IN YOUR SYSTEM'S PROCEDURE
//*  LIBRARY, SUCH AS SYS1.PROCLIB.
//*
//*********************************************************************
//BLMMRES   PROC PRM=BLMMRESP                     PARAMETER DATA MEMBER
//MRESPGM   EXEC PGM=BLMYSM00,REGION=6M,TIME=1440
//STEPLIB   DD   DISP=SHR,DSN=BLM.SBLMMOD1               APF AUTHORIZED
//BLMYPRM   DD   DISP=SHR,DSN=BLM.SBLMSAMP(&PRM)         PARAMETER DATA
//BLGTSX    DD   DISP=SHR,DSN=BLM.SBLMTSX                TSX REXX EXECS
//APIPRINT  DD   SYSOUT=*                                LLAPI LOG
//HLAPILOG  DD   SYSOUT=*                                HLAPI LOG
//SYSTSPRT  DD   SYSOUT=*                                TSX OUTPUT
//          PEND
```


A description of the JCL statements follows:

BLMMRES
: This is the name of the instream catalogued procedure.The **PRM=** parameter specifies the name of a data set member that contains the parameters unique to this MRES. It is set to a default value of **BLMMRESP**, the name of a sample MRES parameters member supplied in the SBLMSAMP data set. If **PRM=** is not specified on the **START** command, the values contained in the sample parameter data member **BLMMRESP** are used. To override the value contained in the cataloged procedure, you can specify a **PRM=** on the **START** command.

MRESPGM
: This statement specifies the program to be processed, its region size, and how long it can run. The value for **PGM=** must be *BLMYSM00* to call the MRES program. The **REGION=** parameter specifies how much memory to reserve for the MRES program. The larger the region size, the more HLAPI sessions can run concurrently. Declaring **TIME=***1440* tells the operating system not to time out the MRES. If this parameter and value are not specified, the MRES will receive an out-of-time abend.

STEPLIB
: This statement specifies the program libraries used by the MRES. You must specify the data sets in a **STEPLIB DD** statement if you do not specify them in the link list or if you do not have the load modules in the link pack area. The data sets must be authorized program facility (APF) libraries.

BLMYPRM
: This statement specifies the data set or PDS member containing the MRES parameters. Both fixed and variable length record formats are supported.

BLGTSX
: This statement points to the PDS (or PDSs) containing the TSX REXX execs.

APIPRINT
: This statement specifies the LLAPI log. If any messages about LLAPI transaction

activity are generated, they are to be sent to the job's **SYSOUT** class. Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* for more information about these messages.

HLAPILOG
This statement specifies the HLAPI log. If any messages about HLAPI transaction activity are generated, they are to be sent to the job's **SYSOUT** class. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information about these messages.

SYSTSPRT
This **SYSOUT** file can be written to by a TSX.

Coding the Parameters for an MRES with TCP/IP

You must specify the MRES parameters in the data set or PDS member that you specify on the **BLMYPRM DD** statement. A sample that you can use for your MRES parameters is shipped as **BLMMRESP** in the **SBLMSAMP** library; this is described in "Sample MRES Parameters" on page 64.

Several basic rules must be followed when you code the parameter data:

- Comments must begin with /* and end with */
- Comments can be in any column between 1 and 72, inclusive
- Nothing (except comments) can be in column 1
- Nothing can be in any column greater than column 72
- Begin the parameters data with a statement identifier of *BLMYPRM*
- You can separate parameters with a comma. Parameters need not appear on separate lines
- The final parameter should end with a semicolon

These are the MRES parameters that you can specify in your **BLMYPRM** data set:

```
[APMINSTRUMENT={OFF|ON}]
[MAXCONNECT={10|n}]
SHUTDOWNTRY=hhmmss
SHUTDOWNWT=hhmmss
[WRITEOPER={1|code}]
COMMTYPE=TCPIP
[IPADDRESS={0.0.0.0|n.n.n.n}]
[PORT={1451|n}]
[PRESTARTSESSIONS={NO|YES}]
[APPLICATION_ID=id]
[PRIVILEGE_CLASS=id]
[SESSION_MEMBER=session member name]
[APIMSG_OPTION={C|P|B}]
[BYPASS_PANEL_PROCESSING={NO|YES}]
[CLASS_COUNT={0|n}]
[DATABASE_ID={5|n}]
[DEFAULT_DATA_STORAGE_SIZE={1024|n}]
[DATE_FORMAT={DATABASE|pattern}]
[DEFAULT_OPTION={ALL|REQUIRED|NONE}]
[HLIMSG_OPTION={C|P|B}]
[MULTIPLE_RESPONSE_FORMAT={SEPARATOR|PHRASE}]
```

```
[PDB_TRACE={NO|YES}]
[SPOOL_INTERVAL={0|n}]
[TABLE_COUNT={0|n}]
[TIMEOUT_INTERVAL={300|n}]
```

The values that you can specify for these parameters are as follows:

APMINSTRUMENT={OFF|ON}
This parameter specifies the APM Instrumentation flag. A specification of **ON** indicates that the MRES should generate heartbeats or pulses to allow Tivoli Global Enterprise Manager (GEM) Application Policy Manager (APM) to monitor the status of the MRES. To have pulses generated, specify **APM=ON**. If a value is not specified, the default value is **OFF**. This parameter is optional. This parameter is provided for those users who are using Tivoli Global Enterprise Manager (GEM) to manage Tivoli Information Management for z/OS in a Tivoli Management Environment. The *Tivoli Information Management for z/OS Guide to Integrating with Tivoli Applications* contains additional information about the Global Enterprise Manager.

If you are not using Tivoli Information Management for z/OS in a Tivoli GEM environment, you can omit this parameter.

MAXCONNECT={10|n}
This parameter specifies the maximum number of client conversation processors this MRES can run concurrently. Valid values are from 1 to 50. The default value is 10. This is an optional parameter.

SHUTDOWNTFY=hhmmss
A required parameter that specifies the time interval between shutdown notification messages. The value specified indicates the amount of time between operator notification messages that follow the first message. When the operator issues the MVS **STOP** command, the MRES sends a message to the operator indicating the number of conversations that are still connected. After this first notification message is sent, notification messages are sent at the interval specified on this parameter. For example, if you specify a value of 000000, you will receive only one notification message. If you specify a value of 000500, you will receive the first notification message and additional messages at 5-minute intervals until either the **SHUTDOWNTFY** time expires or all the conversations have ended.

You must specify this interval as **HHMMSS** where
HH Hours
MM Minutes
SS Seconds

SHUTDOWNWT=hhmmss
A required parameter that specifies the shutdown wait time period. The value specified indicates a period of time that the MRES is to continue processing after receiving the MVS **STOP** command. This time period permits client conversations to complete any processing that was active when the operator issued the **STOP** command. Additional conversations are not accepted during this time, but those already connected can continue until the time expires. If no clients are connected when the **STOP** command is issued, the MRES stops processing immediately regardless of the interval specified on this parameter. Also, when all the conversations have stopped, the MRES stops processing immediately.

You must specify this interval as **HHMMSS** where
HH Hours

MM Minutes
SS Seconds

WRITEOPER={1|code}
This parameter specifies the default write-to-operator (WTO) routing code. Valid values are from 1 to 128. If not specified, the default value is 1. All WTOs that are not a result of command responses are automatically routed to this code. To determine the routing codes for a console, you can do one of the following:

- Display console characteristics by issuing the **DISPLAY CONSOLES,A** command from the console.
- Review the **ROUTCODE** parameter of the **CONSOLE** statements in the **CONSOL**.*xx* member of **SYS1.PARMLIB**.

This parameter is optional.

COMMTYPE=TCPIP
A required parameter that specifies the communications protocol to be used. You must specify **TCPIP** for the MRES with TCP/IP.

IPADDRESS={0.0.0.0|n.n.n.n}
This parameter specifies the Internet or IP address of the network interface for accessing the MRES with TCP/IP. It is the unique address of the host on a network and is specified in dotted decimal format. This consists of four numbers with valid values from 0 to 255, separated by periods. Some hosts have more than one network address. If you want to allow this MRES to receive connection requests from any of the network interfaces, specify the dotted decimal string of all zeros (**000.000.000.000)**. If a value is not specified, the default value is **0.0.0.0** (this is the equivalent of **000.000.000.000**).

PORT={1451|n}
If you specified **COMMTYPE=TCPIP**, this parameter specifies the unique port number used to communicate between TCP/IP and the MRES address space. This number must not be used by any other application, including another MRES or any HLAPI/USS requester that may be running on the same MVS host. Valid values are from **1** to **65534**. The value you specify here becomes the default port number for this MRES address space. This parameter is optional. If a value is not specified, the default value is **1451**.

PRESTARTSESSIONS={NO|YES}
Pre-start API sessions indicator. Set this to **YES** to specify that the API sessions are to be pre-started. If you specify **NO** or omit this parameter, the API sessions are not pre-started. This parameter is optional.

APPLICATION_ID=id
Contains a 1- to 8-character uppercase application ID that Tivoli Information Management for z/OS uses for this session. The application ID is specified on the HL01 transaction and can be specified on many other HLAPI transactions, so it can vary over the life of the HLAPI session. The ID must be an eligible user of the privilege class being used. This keyword is required when **PRESTARTSESSIONS=YES** is specified; otherwise, it is optional.

PRIVILEGE_CLASS=id
Contains a 1- to 8- byte privilege class name, which can contain DBCS characters enclosed by an SO/SI pair. A privilege class remains in effect until your application specifies a different privilege class name. An application can specify an initial privilege class that grants all authority required for the duration of the Tivoli Information

Management for z/OS session. This keyword is required when **PRESTARTSESSIONS=YES** is specified; otherwise, it is optional.

SESSION_MEMBER=session member name
Contains a 7- or 8-character load library session parameter member name that Tivoli Information Management for z/OS uses for this session. Session member names begin with the character string *BLGSES* and cannot contain imbedded blanks. This keyword is required when **PRESTARTSESSIONS=YES** is specified; otherwise, it is optional.

APIMSG_OPTION={C|P|B}
Contains a 1-character LLAPI message option parameter **P**, **C**, or **B**.

- A value of **P** specifies that the LLAPI writes messages to the **APIPRINT** data set.
- A value of **C** specifies that the LLAPI chains messages and passes them from the LLAPI to the HLAPI for conversion into message PDBs.
- A value of **B** specifies that the LLAPI performs both **P** and **C**.

If you omit this parameter, then the LLAPI performs option **C**. This parameter is used only if **SPOOL_INTERVAL** is specified and is not set to zero. This parameter is optional and is used only if **PRESTARTSESSIONS=YES**.

BYPASS_PANEL_PROCESSING={NO|YES}
Bypass panel processing indicator. Set this to **YES** to specify that no panels be used in record processing other than those used by the delete transaction. If you specify **NO** or omit this parameter, the HLAPI performs panel processing.

If you specify **BYPASS_PANEL_PROCESSING = YES**, you must use data model records for the following transactions:

- HL08 Create record
- HL09 Update record
- HL12 Add record relation

This parameter is optional and is used only if **PRESTARTSESSIONS=YES**.

CLASS_COUNT={0|n}
Contains a character number that indicates the maximum number of Tivoli Information Management for z/OS privilege class records that can be maintained in storage during the life of this Tivoli Information Management for z/OS session. If you omit this parameter or enter zero as its value, the Tivoli Information Management for z/OS session operates with a single privilege class record in storage at a time. This parameter is optional and is used only if **PRESTARTSESSIONS=YES**.

DATABASE_ID={5|n
A 1-byte character field containing the 1-character ID number of the database to be used. For Tivoli Information Management for z/OS records, the database ID number is **5**. If you omit this parameter, the HLAPI automatically sets the database ID to **5**. This parameter is optional and is used only if **PRESTARTSESSIONS=YES**.

DEFAULT_DATA_STORAGE_SIZE={1024|n}
Contains a character specifying how much additional storage is allocated to hold default response data from an alias table when your application is creating records. When the HLAPI creates records, it calculates the size of the response buffer it needs by totaling the lengths of all the input data PDBs and adding the specified default data storage size. If you omit the default data storage size, the HLAPI adds a default of 1024 bytes. When the HLAPI performs create response processing, it always checks to make sure the

response will not overlay storage. If the response will overlay storage, the HLAPI transaction will end with an error code. You use this parameter with the **DEFAULT_OPTION** parameter. This parameter is optional and is used only if **PRESTARTSESSIONS=YES**.

DATE_FORMAT={DATABASE|pattern}
Contains a character field that specifies how your application uses dates. If the value **DATABASE** is specified, the default format from the database is assumed. If you omit this parameter, the value **DATABASE** is assumed. Valid values are:

MM/DD/YY
MM-DD-YY
MM.DD.YY
MM/DD/YYYY
MM-DD-YYYY
MM.DD.YYYY
DD/MM/YY
DD-MM-YY
DD.MM.YY
DD/MM/YYYY
DD-MM-YYYY
DD.MM.YYYY
YY/MM/DD
YY-MM-DD
YY.MM.DD
YYYY/MM/DD
YYYY-MM-DD
YYYY.MM.DD
DDMMMYY
DDMMMYYYY
YYDDD
YYYYDDD

This parameter is optional and is used only if **PRESTARTSESSIONS=YES**.

DEFAULT_OPTION={ALL|REQUIRED|NONE}
Contains a character field that specifies how the HLAPI performs create default data response processing in this session. The valid data values for **DEFAULT_OPTION** are **ALL**, **REQUIRED**, and **NONE**. **ALL** specifies that all response fields specified in a PIDT are candidates for default responses. **REQUIRED** specifies that only required fields are candidates for default responses. The HLAPI does not perform default response processing if you omit this field or specify it as **NONE**. After the create transaction completes, the HLAPI reverts to the initial default specification for record creation unless overridden again. This parameter is optional. If you omit this parameter, the value **NONE** is assumed. This parameter is optional and is used only if **PRESTARTSESSIONS=YES**.

HLIMSG_OPTION={C|P|B
Contains a 1-character HLAPI message option parameter **P**, **C**, or **B**.

- A value of **P** specifies that the HLAPI writes messages to the HLAPILOG data set.
- A value of **C** specifies that the HLAPI chains messages on the PDB message chain.
- A value of **B** specifies that the HLAPI performs both **P** and **C**.

If you omit this parameter, then the HLAPI performs option **C**. The HLAPI writes messages passed back from the LLAPI to the **HLAPILOG** data set. This parameter is used only if **SPOOL_INTERVAL** is specified and is not set to zero. This parameter is optional and is used only if **PRESTARTSESSIONS=YES**.

MULTIPLE_RESPONSE_FORMAT={SEPARATOR|PHRASE}
A specification of **PHRASE** permits you to use spaces to separate multiple response fields. The default value, **SEPARATOR**, requires that a separator character be specified in the control PDB named **SEPARATOR_CHARACTER** on the HLAPI transaction submitted by the client application. This parameter is optional and is used only if **PRESTARTSESSIONS=YES**.

PDB_TRACE={NO|YES}
This parameter specifies whether the HLAPI should perform PDB data tracing for debugging purposes or PDB data logging to the HLAPILOG output file. Setting this parameter value to **YES** causes the logging of up to 32 bytes of PDBDATA information for each PDB used throughout the session. A value of **NO** causes no PDB logging to be performed. This parameter is optional and is used only if **PRESTARTSESSIONS=YES**. If you omit this parameter, the value **NO** is assumed.

SPOOL_INTERVAL={0|n}
Contains a character specifying the number of minutes that the HLAPI spools the activity logs **HLAPILOG** and **APIPRINT** when messages are printed. If the HLAPI is spooling to a data set and this time interval has passed, the activity logs are recycled and new log information is written starting at the top of the data set, writing over any existing information. If you omit this parameter, the HLAPI does not log messages and the settings in **APIMSG_OPTION** and **HLIMSG_OPTION** are ignored. This parameter is optional and is used only if **PRESTARTSESSIONS=YES**.

TABLE_COUNT={0|n}
Contains a character that indicates the maximum number of alias tables and PIDTs and anchored PIPTs that the HLAPI can maintain in storage during the life of a Tivoli Information Management for z/OS session. Static PIDTs and PIDTs generated from data view records are treated the same for caching purposes. It can take a significant amount of time to generate a PIDT from data view records. The length of time depends on the number of data attribute records (and validation records they reference) contained in the data view record. Therefore, it can be especially important to direct the HLAPI to maintain PIDTs in storage if you are using data models. If you specify this value as zero or omit it, the Tivoli Information Management for z/OS session will not process **ALIAS_TABLE** parameters or cache PIDTs. Alias table and PIDT processing can increase transaction run time due to the increased I/O time of loading and unloading tables. By balancing the table count to alias table and PIDT usage, you can reduce to zero the additional I/O overhead for long-running applications. This parameter is optional and is used only if **PRESTARTSESSIONS=YES**.

TIMEOUT_INTERVAL={300|n}
Contains a character specifying the number of seconds that a transaction can run before a timer interrupt occurs. If you specify a value between 0 and 45 seconds, the HLAPI uses a value of 45 seconds. If you specify a value of 0 or omit this parameter, the HLAPI uses a default value of 300 seconds (five minutes). This parameter is optional and is used only if **PRESTARTSESSIONS=YES**.

Sample MRES Parameters

A sample of the MRES parameters is provided in **BLMMRESP** in the **SBLMSAMP** data set. If you rename this member or create a new one, be sure to change the member name in the **PRM=** parameters in the JCL for your MRES with TCP/IP procedure or be sure to specify **PRM=** on the **START** command when you start the MRES. Refer to “Defining a Procedure for an MRES with TCP/IP” on page 56 for an example of the JCL for your MRES with TCP/IP procedure. The following are sample parameters for an MRES with TCP/IP.

```
/*****************************************************************************/
/*                                                                           */
/*            MRES START UP PARAMETERS                                      */
/*                                                                           */
/*****************************************************************************/
BLMYPRM                          /* SPECIFY MRES PARAMETERS                  */
/*****************************************************************************/
/*                                                                           */
/*             PARAMETERS TO CONTROL THE GENERAL MRES SESSION                */
/*                                                                           */
/*****************************************************************************/

  APMINSTRUMENT=OFF,                     /* APM INSTRUMENTATION (ON OR OFF)    */
  MAXCONNECT=10,                         /* MAXIMUM NUMBER OF CONNECTIONS      */
  SHUTDOWNTFY=000200,                    /* SHUTDOWN NOTIFY TIME (HHMMSS)      */
  SHUTDOWNWT=000500,                     /* SHUTDOWN WAIT TIME (HHMMSS)        */
  WRITEOPER=1,                           /* WRITE-TO-OPERATOR ROUTING CODE     */

/*****************************************************************************/
/*                                                                           */
/*             PARAMETERS TO CONTROL THE COMMUNICATIONS SESSION              */
/*                                                                           */
/*****************************************************************************/

  COMMTYPE=TCPIP,                       /* COMMUNICATIONS (APPC OR TCPIP)      */
  IPADDRESS=000.000.000.000,            /* INTERNET PROTOCOL ADDRESS           */
  PORT=1451,                            /* TCP/IP UNIQUE PORT NUMBER           */

/*****************************************************************************/
/*                                                                           */
/*             PARAMETERS TO CONTROL THE PRE-STARTING OF API SESSIONS        */
/*                                                                           */
/*****************************************************************************/

  PRESTARTSESSIONS=NO,                  /* PRE-START (YES OR NO)               */
  APPLICATION_ID=XXXXXXXX,              /* APPLICATION ID                      */
  PRIVILEGE_CLASS=XXXXXXXX,             /* PRIVILEGE CLASS NAME                */
  SESSION_MEMBER=BLGSESXX,              /* SESSION MEMBER NAME                 */
  APIMSG_OPTION=C,                      /* LLAPI MESSAGES (C, P, OR B)         */
  BYPASS_PANEL_PROCESSING=NO,           /* BYPASS PANELS (YES OR NO)           */
  CLASS_COUNT=0                         /* MAXIMUM NUMBER TO BE CACHED         */
  DATABASE_ID=5,                        /* DATABASE ID                         */
  DEFAULT_DATA_STORAGE_SIZE=1024,       /* DEFAULT SIZE IN BYTES               */
  DATE_FORMAT=DATABASE,                 /* FORMAT OF DATE DATA                 */
  DEFAULT_OPTION=NONE,                  /* (ALL, REQUIRED, OR NONE)            */
  HLIMSG_OPTION=C,                      /* HLAPI MESSAGES (C, P, OR B)         */
  MULTIPLE_RESPONSE_FORMAT=SEPARATOR,         /* (PHRASE OR SEPARATOR)         */
  PDB_TRACE=NO,                               /* PDB DEBUG TRACE (YES OR NO)   */
  SPOOL_INTERVAL=0,                     /* MAXIMUM TIME IN MINUTES             */
  TABLE_COUNT=0,                        /* MAXIMUM TABLE ENTRIES               */
  TIMEOUT_INTERVAL=300;                 /* MAXIMUM TIME IN SECONDS             */
```

Adding the Data Sets to the APF List

If you neither specified the data sets that contain the MRES load modules in the link list nor have them in the link pack area, you must add the data sets to the APF library.

To define the data sets that contain the load modules as APF libraries, make an entry for each data set in the appropriate **PROG***xx* members of **SYS1.PARMLIB**. Each entry in a **PROG***xx* member includes the data set name (*dsn*) and the volume serial number (*volser*) of the library.

The changes will be activated the next time you IPL the system, or you can use the MVS operator command **SET PROG=***xx* to dynamically activate the changes.

Starting and Stopping an MRES with TCP/IP

TCP/IP/MVS and the BLX-SP server must be running before you start an MRES with TCP/IP.

To start an MRES with TCP/IP, issue the MVS system operator **START** command for the cataloged procedure. To stop an MRES with TCP/IP, issue the MVS system operator **STOP** command. For full descriptions of the MVS **START** and **STOP** operator commands, refer to *OS/390 Operations: System Commands*. The **START** command accepts the parameters listed in the following section.

START Command Syntax

The syntax of the MVS system operator **START** command for an MRES with TCP/IP is as follows:

```
S blmmres[,JOBNAME=job_name][,PRM=member_name]
```

S The MVS system operator **START** command.

blmmres
The name of the cataloged procedure for this MRES with TCP/IP.

Note: The name of the cataloged procedure becomes the name of the job unless it is overridden by the **JOBNAME=** parameter.

JOBNAME=*job_name*
The value assigned to this keyword becomes the name of the newly started MRES. If this parameter is not specified, the name of the cataloged procedure becomes the name of the job.

Note: For the MRES with TCP/IP, multiple MRESs can be started with the same or different job names, but only with different parameter data. This is because the **PORT=** parameter must be unique for each started MRES.

PRM=*MRES_parameters_member_name*
In the sample procedure **BLMMRES** (see “Defining a Procedure for an MRES with TCP/IP” on page 56), **PRM=** specifies the name of the data set member that contains the MRES parameters. If you specify **PRM=** on the **START** command, the MRES parameters in that member are used. They override the values contained in the MRES parameters data set member referenced by **PRM=** in the catalogued procedure for this MRES with TCP/IP. If **PRM=** is not specified on the **START** command, the MRES parameters contained in the data set member referenced by the cataloged procedure are used. Refer to “Defining a Procedure for an MRES with TCP/IP” on page 56 for a

sample of the cataloged procedure. The sample uses the MRES parameters in **SBLMSAMP** data set member **BLMMRESP**

STOP Command Syntax

The syntax of the MVS system operator **STOP** command for an MRES with TCP/IP is as follows:

```
P job_name[,A=n]
```

Note: Note that *job_name* is the name of the cataloged procedure that was started, unless the parameter **JOBNAME** was specified on the **START** command.

P The MVS system operator **STOP** command.

job_name
The name of the cataloged procedure for this MRES with TCP/IP.

A=*n*
An optional parameter that specifies the address space number consisting of from 1 to 4 hexadecimal digits (0–F). You can obtain this number in several ways:

- Use the Display Jobs operator command (*D J,BLMMRES* displays information about all jobs with the name **BLMMRES**).
- Look for **ASID=***n* in message **BLM03170I**. This message is written to the console when the MRES is successfully started.
- Use other display commands. (See "Using MVS Operator Commands" on page 67 for information about the **DISPLAY** command.)

If you do not use this parameter, all jobs with the name **job_name** are stopped. If you do use this parameter, only the job you specify is stopped.

Determining Values Clients Need

When you set up the communication links on the clients, you will need the following values:

- Port number

  This is the unique port number for a particular MRES with TCP/IP. For the HLAPI/UNIX client system, this value must be set in the **/etc/services** file. For the HLAPI/USS client system, this value must be set in either the **/etc/services** file or the **hlq.ETC.SERVICES** data set. For HLAPI/2 and HLAPI/NT client systems, this value must be set in the **SERVICES** file in the **ETC** subdirectory.

- Internet address

  This is the address of the MVS host with which you are communicating. If you use a host name in your database profile to identify the MVS host, this value must be set in the file on the HLAPI/UNIX client systems running the requester. If you use a host name on the HLAPI/2 or HLAPI/NT or HLAPI/USS client systems, the host name must be resolvable.

6

Using MVS Operator Commands

MVS supports a set of system commands that enables the operator to monitor APPC operation, and, thus, a RES or an MRES with APPC address space. These commands are relevant to the server address space. For specific details on using these commands, refer to *OS/390 MVS Planning: APPC/MVS Management* and *OS/390 MVS: System Commands*.

DISPLAY
Views server address space status

CANCEL
Removes a server address space from the system

FORCE
Destructively removes a server address space from the system

To use these commands, you must know the job and TP name for the server address space. You can use the following **DISPLAY** command to see these names:

```
DISPLAY APPC,TP,ALL
```

Because TCP/IP is not integrated into MVS, it does not have special operator commands (such as the **DISPLAY APPC** command). However, because the MRES with TCP/IP runs in an MVS address space, you can use job-related operator commands (for example, **DISPLAY JOB**).

Displaying Server Address Space Status

You can display detailed information about the APPC server address space running in a system by using the **DISPLAY APPC** and **DISPLAY ASCH** commands. The **DISPLAY APPC** command shows information about the status of the server TP (and other APPC TPs and LUs). The **DISPLAY ASCH** command displays information about the status of the APPC scheduler and the work it manages. The following sample commands can be used to find the status of one or more server address spaces:

To display information about the server TP, enter the statement:

```
D APPC,TP,LIST,LTPN=tp_name
```

To display the status of a server in a particular address space, enter the statement:

```
D APPC,LIST,ASID=asid
```

You can display detailed information about either the APPC or the TCP/IP server address space running in a system by using the Display Jobs operator command. For example, to display information about all jobs with the name **BLMMRES**, enter the statement:

```
D J,blmmres
```

Cancelling a Server Address Space

Use the **CANCEL** command to stop a single instance of a server immediately. The **CANCEL** command requires the job name for the particular server and the server's address space identifier (**ASID**). To get this information, use an example shown in section "Displaying Server Address Space Status" on page 67. When you have the **ASID**, use the command:

```
CANCEL job_name,A=server_asid
```

If the **CANCEL** request does not succeed in cancelling the server address space, you can use the **FORCE** command.

Forcing a Server Address Space

The **FORCE** command can be used when the **CANCEL** command fails to remove a server address space from the system. **FORCE** deletes the address space from the system without allowing any cleanup or recovery to occur. This can result in some loss of resources until the system is re-IPLed. **CANCEL** must be issued before you can use **FORCE**. The **FORCE** command requires the job name for the particular server and the server address space identifier (**ASID**). Use the following command:

```
FORCE job_name,A=server_asid
```

7

Introduction to the HLAPI/2

Tivoli Information Management for z/OS supports remote access from a workstation that runs in an OS/2 environment. It does this through the High-Level Application Program Interface (HLAPI) and the Tivoli Information Management for z/OS HLAPI Client for OS/2 (HLAPI/2). The HLAPI/2 provides remote access to Tivoli Information Management for z/OS data and data manipulation services. It consists of three parts:

- A Tivoli Information Management for z/OS server, an MVS-based transaction program that resides on the MVS host system. It provides the link between Tivoli Information Management for z/OS and the OS/2 system.
- The Tivoli Information Management for z/OS HLAPI/2 Requester (requester), an OS/2-based transaction program that provides workstation access to the HLAPI through a Tivoli Information Management for z/OS server.
- C and REXX language bindings and support Dynamic Link Libraries (DLLs) for the C and REXX languages.

Like the HLAPI, the HLAPI/2 is a transaction-based application programming interface. User application programs interact with Tivoli Information Management for z/OS from the remote environment in basically the same way as they do from MVS using the HLAPI. These remote environment user application programs can be thought of as the *clients* to Tivoli Information Management for z/OS's *server*. The remote environment offers a subset of HLAPI transactions, which are listed in Table 1 on page 3, described in "HLAPI/2 Transactions" on page 109, and described in the *Tivoli Information Management for z/OS Application Program Interface Guide* .

The HLAPI/2 enables application programmers to write applications for use in their specific work environment. The task described in "A Typical Scenario" is typical of the system problems that can be solved by using Tivoli Information Management for z/OS database services.

A Typical Scenario

Suppose an application programming group in an enterprise has written two workstation-based help desk applications that interact with the HLAPI through the HLAPI/2. One is a problem management database application, and the other is a configuration management database application. The application programming group has already provided the help desk with the information necessary to install and start these applications successfully.

For efficiency, the help desk operator maintains two separate user IDs on the MVS system: one with basic privilege class authority for queries sent through the configuration management application, and one with a higher privilege class authority for creating records through the problem management application.

1. A help desk operator starts an OS/2 workstation. The workstation's **STARTUP.CMD** command file starts up the HLAPI/2 requester, the problem management application, and the configuration management application.
2. When a problem call arrives, a help desk operator uses the problem management application to collect preliminary information and open a problem record through the HLAPI/2.
3. In another OS/2 window, the same operator uses the configuration application to query Tivoli Information Management for z/OS through the HLAPI/2 for information about the caller's configuration.
4. Meanwhile, Tivoli Information Management for z/OS has returned a problem number back through the HLAPI/2, and the operator gives the caller his problem number and promises follow-up on the problem.
5. By this time, Tivoli Information Management for z/OS has returned results of the configuration query through the HLAPI/2. The operator can then research the problem and update the problem record as necessary.

The same Tivoli Information Management for z/OS functions that once required direct host access are now performed on a desktop workstation. The remaining sections of this chapter help you understand the interactions of the HLAPI/2 and Tivoli Information Management for z/OS.

Server Overview

A Tivoli Information Management for z/OS server is an MVS/ESA transaction program that handles communication between an HLAPI/2 requester and any Tivoli Information Management for z/OS databases that reside on the MVS system where the server is installed. An OS/2 client application program must use a requester for access to the server.

A server must be installed and available on every MVS/ESA machine with a Tivoli Information Management for z/OS database that an application using the HLAPI/2 needs to access.

The HLAPI/2 can use any of the Tivoli Information Management for z/OS servers. See "Configuring and Running a Remote Environment Server (RES)" on page 25, "Configuring and Running a Multiclient Remote Environment Server (MRES) with APPC" on page 35, and "Configuring and Running a Multiclient Remote Environment Server (MRES) with TCP/IP" on page 53 for information about the servers.

When deciding which server to use, consider the communication protocol that each one supports. Also consider the security requirements of your application and how you will implement these requirements.

Requester Overview

The requester is a transaction program that runs on the workstation. It must be up and running before any HLAPI/2 activity can occur. At the request of a client application program, the requester initiates a conversation with a server. Then the requester transfers information from the client application program to the appropriate server, and from the server back to the client application program.

The requester appears on your workstation as a Presentation Manager® (PM) window. To start the requester, select **OK** in the window. You can also start the requester from the workstation's STARTUP.CMD file. See "The HLAPI/2 Requester" on page 105 for more information about starting the requester.

The HLAPI/2 requester is implemented as an OS/2 system service. Therefore, HLAPI/2 requester services are available to all processes and related threads running in the OS/2 environment. One copy of the HLAPI/2 requester service can support many user applications.

HLAPI/2 C Language Binding

A client application program communicates with the Tivoli Information Management for z/OS system by creating a high-level application communication area (HICA) and its related parameter data blocks (PDBs). The client application program then submits the HICA transaction by making HLAPI/2 program service calls. The HLAPI/2 program service routines exist on the user's workstation as a dynamic link library (DLL). When the calls are made by the client application program, the supporting HLAPI/2 routines are loaded from the DLL and started.

To utilize HLAPI/2 program service calls, the HLAPI/2 provides two standard header files and one import library. These are located in the H subdirectory of the directory in which you install HLAPI/2. These files are only used in the creation of an HLAPI/2 application. After the application is written and installed, these files are not required to be on a user's workstation.

IDBH.H is a required C programming language header file for all HLAPI/2 application programs. It defines data types and function calls used by HLAPI/2 to communicate with your application program.

IDBHLAPI.LIB is the import library that contains the function calls provided by HLAPI/2. You must specify this library when you link your compiled program.

IDBECH.H is an optional C programming language header file that defines constant declarations for return and reason codes used by the HLAPI/2.

After you install the HLAPI/2, refer to "HLAPI/2 C Language Application Program" on page 121 to learn more about the header files and the import library.

Basic Transaction Flow

A *transaction sequence* is a series of HLAPI/2 transactions that begins with an initialize Tivoli Information Management for z/OS (HL01) transaction, followed by other supported transactions in any order, and ends with a terminate Tivoli Information Management for z/OS (HL02) transaction. Client application programs submit transactions in a transaction sequence, which is referred to as a *logical session*.

Each HLAPI/2 transaction request travels from a client application program on OS/2 to Tivoli Information Management for z/OS on MVS along the route shown in Figure 9. This example illustrates OS/2 using a RES. The path would be similar using an MRES with APPC or an MRES with TCP/IP. With an MRES with TCP/IP, the communication protocol would be TCP/IP instead of APPC.

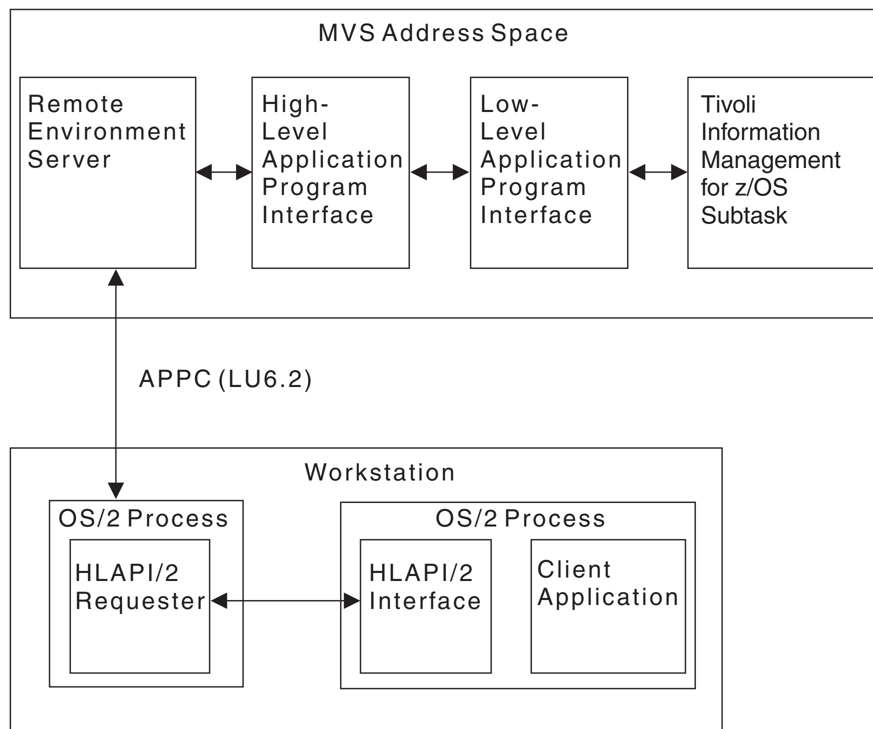


Figure 9. HLAPI/2 Overview

The transaction reply travels from Tivoli Information Management for z/OS to the client application along the same route, but reversed.

On the Workstation

The client application program, through the HLAPI/2 program service calls, uses the requester to request Tivoli Information Management for z/OS work to be performed. The user application creates a HICA and PDBs to represent a Tivoli Information Management for z/OS HLAPI transaction. Then it submits the HICA to HLAPI/2 for processing. HLAPI/2 takes the HICA and its related control and input PDBs and translates them for the server character set. It then packages and transfers them to the server associated with the specified HICA. (Refer to transaction HL01 in the *Tivoli Information Management for z/OS Application Program Interface Guide* for a description of how to associate a HICA with a particular Tivoli Information Management for z/OS database.)

Communication Link

The HLAPI/2 requester communicates with a server using the APPC LU 6.2 protocol or using TCP/IP. The client application chooses the communication protocol and that protocol is used for the entire transaction sequence submitted by that client application.

The requester can communicate with multiple servers on multiple MVS hosts. The **IDBSYMDESTNAME** database profile keyword indicates that the requester is to establish an APPC conversation on behalf of the client. The **IDBSERVERHOST** database profile keyword indicates that the client wants a TCP/IP connection. These keywords and the database profile are described in "HLAPI/2 Profiles, Environment Variables, and Data Logging" on page 95.

To enable the communication link between an OS/2 workstation and a RES or an MRES with APPC, APPC/MVS and the Communications Manager/2 must be configured for HLAPI/2 use. For information on configuring a RES or MRES with APPC, refer to "Configuring and Running a Remote Environment Server (RES)" on page 25 and "Configuring and Running a Multiclient Remote Environment Server (MRES) with APPC" on page 35.

To enable the communication link between an OS/2 workstation and an MRES with TCP/IP, TCP/IP must be set up for HLAPI/2 use. For information on configuring an MRES with TCP/IP, see "Configuring and Running a Multiclient Remote Environment Server (MRES) with TCP/IP" on page 53.

On the Host

Upon arrival in the server, the HICA and PDBs are processed and submitted to the Tivoli Information Management for z/OS HLAPI. After the requested HLAPI transaction finishes, the server transmits the HICA, the output, error, and message PDBs, and the PDBCODE field of the input PDB to the requester.

Back to the Workstation

The transmission buffers are received by the requester. The buffers are parsed in sequence and their contents (the PDBs) are translated from the server character set to the character set being used by the workstation. Memory is allocated for the newly received PDBs. These PDBs are chained onto their corresponding type list on the owning (and original) HICA. The HICA contains the original control and input PDB chains and the new output, error, and message PDB chains. The HICA contains other fields set by the HLAPI, such as **HICARETC** and **HICAREAS**. The **PDBPROC** value set by the HLAPI in the input PDB field is also returned. The transaction request is complete, and control of the HICA is returned to the client application program.

The REXX HLAPI/2 Interface

This interface enables you to access HLAPI/2 transactions from OS/2 REXX programs. The REXX HLAPI/2 is similar to the HLAPI/REXX interface on MVS. Using the REXX HLAPI/2, the client programmer can write REXX programs to:

- Set variables with control and input information
- Submit transactions to the HLAPI/2 through the REXX HLAPI/2 interface
- Retrieve information from REXX variables set by the REXX HLAPI/2 interface

All of the HLAPI/2 transactions are available to the REXX HLAPI/2.

The REXX HLAPI/2 is installed as a component of the HLAPI/2. It consists of a single functional part, the REXX HLAPI/2 DLL, which is the interface between the application REXX program and the HLAPI/2:

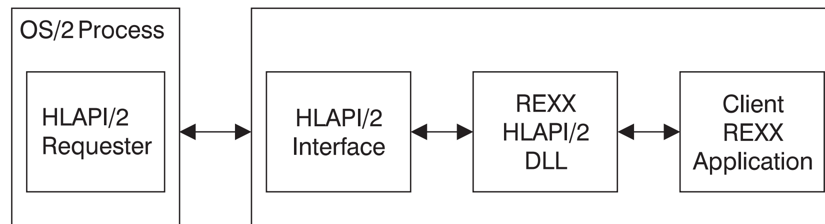


Figure 10. The REXX HLAPI/2 Interface

The *Tivoli Information Management for z/OS Application Program Interface Guide* gives useful information for HLAPI/REXX which also applies to writing REXX HLAPI/2 applications.

Client Workstation Requirements for the HLAPI/2

The HLAPI/2 requires certain software and hardware to function.

Software

Each HLAPI/2 client workstation requires the following software:

- OS/2 WARP® Version 3 or OS/2 WARP Version 4
- VisualAge® C++ for OS/2, or any C language compiler and linker that supports the 32-bit system linkage convention.

  Note: A compiler is required only if you are developing client applications. A client workstation does not need a compiler to run a client application.

To use the optional HLAPI for Java provided with the client, you must have Java Development Kit (JDK) Version 1.1 or higher, and OS/2 WARP Version 4.

To use a RES or MRES with APPC:

- Communications Manager/2 Version 1.1 or a later release (required for the APPC protocol).

To use an MRES with TCP/IP:

- Because OS/2 WARP Version 3 and OS/2 WARP Version 4 contain support for TCP/IP, no separate installation of TCP/IP is required.

Hardware

- An IBM personal computer or compatible system unit capable of running OS/2 WARP Version 3 or OS/2 WARP Version 4, and Communications Manager/2 1.1 (or higher) if you are using APPC.
- One or more fixed disk drives with sufficient capacity to contain your version of OS/2, and possibly, Communications Manager/2; note also the disk storage requirements specified in "Disk Storage" on page 75
- Token-Ring Adapter Card and network or a communication option capable of supporting LU 6.2 or TCP/IP communication to one or more MVS systems running a Tivoli Information Management for z/OS server.

Disk Storage

The amount of fixed disk space needed by HLAPI/2 depends on which parts of the product you install, and how you install them. You can install HLAPI/2 only on an HPFS drive. When you install HLAPI/2, the disk space needed for each component is:

- Installation and Maintenance utility, 1.40 MB
- Run time parts, 610 KB
- Toolkit parts, 260 KB

When you install the HLAPI/2 from a local area network (LAN), the disk space listed above is accurate, but the Installation and Maintenance utility is only temporarily copied to your workstation; it is then deleted.

8

Installing and Configuring HLAPI/2

To use HLAPI/2 on your workstation, you must do the following tasks:

1. Configure a communication link to a Tivoli Information Management for z/OS server. "Configuring a Communication Link to a Server" discusses this task.
2. Install the HLAPI/2 files from the source system to your workstation. "Installing HLAPI/2 on the Workstation" on page 82 discusses this task.
3. Customize the HLAPI/2 files. "Customizing the HLAPI/2 CONFIG.SYS File" on page 90 discusses this task.

Configuring a Communication Link to a Server

In order to use the functions of the HLAPI/2, you must configure a communication link for the workstation to each server you want to use. If the server you are using is a RES or MRES with APPC, you must configure an APPC/APPN communication link. If you are using an MRES with TCP/IP, you must configure a TCP/IP communication link. The following sections describe how to configure your communications software and update various files. "Configuring HLAPI/2 for TCP/IP" on page 81 provides you with the steps needed for a TCP/IP configuration. The following section, "Configuring HLAPI/2 for APPC", provides you with the steps needed for an APPC configuration.

Configuring HLAPI/2 for APPC

To use a RES or MRES with APPC, you must configure an APPC/APPN communication link for the workstation to each server you want to use. On an OS/2 workstation, you use Communications Manager/2 to configure the link. You can either use a text editor to modify a CM/2 **Node Definition File** (**NDF**) or step through the panels CM/2 provides. This section shows you the statements to add to an **NDF** file for the HLAPI/2. It does not include defining a link for 3270 sessions.

Your **NDF** file is located in the **\CMLIB** directory. It has the same file name as your Communications Manager configuration file, which has a file extension of **CFG**. The **NDF** file has the file extension **NDF**.

There are also example **CFG** and **NDF** files. These files are provided with the *CM/2 Configuration Guide* files. If you want to configure a machine from scratch, copy the **BASE2.NDF** and **BASE2.CFG** files into the **\CMLIB** directory. If you want to modify an existing configuration, do not copy the **BASE2** files. You can, however, cut and paste the verbs needed from the file **BASE2.NDF** to your own **NDF** file.

For more information about configuring CM/2, refer to the following guides:

- *CM/2 Installation, Configuration and Administration*
- *CM/2 Configuration Guide*

- *CM/2 APPC Programming Guide and Reference*

The tasks you need to do to configure APPC/APPN are:

1. Determine the values to use in the configuration.
2. Define a local LU.
3. Define a logical link to the MVS host where the server resides.
4. Define a partner LU.
5. Define CPI-C side information.
6. Verify the configuration.

The **NDF** file configuration samples in this chapter show what commands and parameters should be in an **NDF** file to define a direct link to a Tivoli Information Management for z/OS server. If the workstation is an end node on a token-ring local area network that communicates with MVS by way of a network node, define the partner LU on the network node and define a link from the end node to the network node.

Determining Configuration Values

Configuring Communications Manager/2 for the HLAPI/2 is independent of configuring a server. For example, whether the server is a RES or an MRES is transparent to the HLAPI/2. However, you need to know some of the values that were declared when the server was configured. See "Determining Values Clients Need" on page 33 if the server is a RES. See "Determining Values Clients Need" on page 50 if the server is an MRES with APPC.

Defining a Local LU for HLAPI/2

Each workstation intending to use HLAPI/2 must have a local LU definition in its environment. The LU name must match a partner LU name defined in VTAM for the MVS server system. Use the **DEFINE_LOCAL_CP** command as shown here.

```
DEFINE_LOCAL_CP
   FQ_CP_NAME(network_name.luname)
   CP_ALIAS(lu_alias)
   NODE_TYPE(node_type)
   HOST_FP_SUPPORT(YES)
   HOST_FP_LINK_NAME(link_name);
```

FQ_CP_NAME
The fully qualified name of this LU: **network_name** is the ID for the entire network and **luname** is the name of the local LU on this workstation.

CP_ALIAS
lu_alias is an alias for this LU. Alias names are case sensitive. You can define it as (**lu_alias,LU_ALIAS**).

NODE_TYPE
node_type is **EN** to define an end node.

HOST_FP_SUPPORT
The value of **YES** indicates that this end node is to communicate with the host over the link identified in the next parameter.

HOST_FP_LINK_NAME
link_name is the name of the logical link this LU uses for communication with the host.

Define a default local LU to use for communication with the host. This definition can be made only by adding a statement to the **NDF** file. Use the **DEFINE_DEFAULTS** statement shown here.

```
DEFINE_DEFAULTS  DEFAULT_LOCAL_LU_ALIAS(lu_alias)
```

lu_alias must match the value you declared on the **CP_ALIAS** statement.

Defining a Link to the MVS System

To define a link directly to the MVS system where the server is installed, add a link definition as shown here, to your **NDF** file.

```
DEFINE_LOGICAL_LINK
   LINK_NAME(linkname)
   FQ_ADJACENT_CP_NAME(partnernet.partnerlu)
   ADJACENT_NODE_TYPE(LEARN)
   DLC_NAME(IBMTRNET)
   ADAPTER_NUMBER(0)
   DESTINATION_ADDRESS(lanaddress)
   CP_CP_SESSION_SUPPORT(NO)
   ACTIVATE_AT_STARTUP(NO)
   SOLICIT_SSCP_SESSION(NO);
```

LINK_NAME
: **linkname** specifies the name of this link.

FQ_ADJACENT_CP_NAME
: *partnernet* is the ID of the network the MVS system is on. **partnerlu** is the name of the LU defined for the Tivoli Information Management for z/OS server on the MVS system.

DESTINATION_ADDRESS
: **lanaddress** is the address by which the MVS system is known to the local area network.

Defining a Partner LU

In order for your workstation to be able to locate a Tivoli Information Management for z/OS server when a client application wants to initiate a conversation, the workstation's **NDF** file must contain a partner LU definition for the server. This partner LU must specify the fully qualified name of the MVS server system as shown in this example.

```
DEFINE_PARTNER_LU  FQ_PARTNER_LU_NAME(network_name.imserver_lu)
                   PARTNER_LU_ALIAS(imserver_lu)
                   PARTNER_LU_UNINTERPRETED_NAME(imserver_lu)
                   MAX_MC_LL_SEND_SIZE(32767)
                   CONV_SECURITY_VERIFICATION(YES)
                   PARALLEL_SESSION_SUPPORT(YES);
DEFINE_PARTNER_LU_LOCATION  FQ_PARTNER_LU_NAME(network_name.imserver_lu)
                            WILDCARD_ENTRY(NO)
                            FQ_OWNING_CP_NAME(network_name.vtam_sscpname)
                            LOCAL_NODE_NN_SERVER(NO);
```

network.name
: The ID for the network the server's LU is on.

imserver_lu
: The name of the server's logical unit. This value on the **PARTNER_LU_ALIAS** parameter can be the same or different from the value on the other parameters.

vtam_sscpname
: The System Services Control Point Name for VTAM. You use this value

instead of a Control Point (CP) name because VTAM does not have a CP name for APPN to use. This value is set in the **ATCSTR***xx* member of the **VTAMLST** data set.

Defining CPI-C Side Information

The HLAPI/2 uses a symbolic destination name to locate the target for a conversation. You define symbolic destination names in your **NDF** file on a **DEFINE_CPIC_SIDE_INFO** statement. Then you specify that same symbolic destination name on the **IDBSYMDESTNAME** keyword in the database profile on the workstation. Refer to “HLAPI/2 Profiles, Environment Variables, and Data Logging” on page 95 for more information about database profile keywords.

This example illustrates the command for defining a symbolic destination name in an NDF file.

```
DEFINE_CPIC_SIDE_INFO  SYMBOLIC_DESTINATION_NAME(symname )
                       FQ_PARTNER_LU_NAME(network_name.imserver_lu)
                       MODE_NAME(mode)
                       TP_NAME(tpname);
```

symname
Declares the symbolic destination name to use on the workstation to locate a server. Use this same symbolic destination name when you define your database profile on the workstation.

network_name.imserver_lu
Specifies a partner LU that is defined in this same NDF file on a **DEFINE_PARTNER_LU** statement. If the symbolic destination name is for a RES, the partner LU must be one that is defined on the MVS system as scheduled. If the symbolic destination name is for an MRES, the partner LU must be one that is defined on the MVS system as nonscheduled.

network_name
Is the ID for the network the server’s LU is on.

imserver_lu
Is the name of the server’s logical unit.

mode
Specifies the name of a compiled log-on mode to use for the conversation. The mode name must match the mode specified in the TP profile for a RES or in the side information entry for an MRES. The log-on mode is an entry in **SYS1.VTAMLIB**. See “Defining the Log-on Mode” on page 30 for information on defining a mode.

tpname
If the symbolic destination name represents a RES, this value must match the name used for the TP profile on the MVS system where the RES resides.

If the symbolic destination name represents an MRES, this value must match the value on the **TPNAME** parameter of the side information entry that the *symname* maps to.

Verifying the Configuration

To verify the changes you have made to your **NDF** file, issue the following command from the OS/2 command prompt:

```
CMVERIFY ndf_file_name.NDF
```

Substitute the name of your **NDF** file for **ndf_file_name**.

To activate the changes you have made to your **NDF** file, stop and restart Communications Manager/2.

Configuring HLAPI/2 for TCP/IP

To use an MRES with TCP/IP, you must configure TCP/IP so that HLAPI/2 can connect to each MRES with TCP/IP server you want to use. Each MRES with TCP/IP is uniquely identified by the IP address of its MVS host and its port number. Refer to “Configuring and Running a Multiclient Remote Environment Server (MRES) with TCP/IP” on page 53 for more information about the MRES with TCP/IP.

To identify the port number of each MRES with TCP/IP, you must update the **SERVICES** file in your **ETC** subdirectory where you installed TCP/IP to associate a service name or alias with the port number of the MRES with TCP/IP. You must specify a service name and port number for each server the HLAPI/2 needs to be able to connect to. The port numbers must match those used by the Tivoli Information Management for z/OS MRES with TCP/IP servers. The service name is of your choosing. Service names are case-sensitive.

A default service name `infoman` and port number `1451` have been reserved for Tivoli Information Management for z/OS use. The general format of an entry in the **SERVICES** file is:

```
<service>     <port>/tcp   <alias_list>       #<comment>
```

<service>
: The service name of the Tivoli Information Management for z/OS MRES with TCP/IP.

<port>
: The port number of the Tivoli Information Management for z/OS MRES with TCP/IP.

<alias_list>
: Alias definitions for the service

<comment>
: Comment text that describes the service.

For example, to associate the default service name `infoman` with the default port number 1451, you must place the following line in the **SERVICES** file before you run HLAPI/2:

```
infoman  1451/tcp      #default MRES server
```

The default service name and default port number are reserved for Tivoli Information Management for z/OS so you can use them to designate your MRES with TCP/IP. If the client application does not specify a service name in the database profile specified on the HL01 transaction, `infoman` will be assumed. Therefore, be sure to include it in the **SERVICES** file.

Be sure that your client application programs use the service names that you define. If your client application program needs to access an MRES with TCP/IP that uses a port number other than the default **infoman/1451**, you must specify its service name in the **IDBSERVERSERVICE** keyword in the client application’s database profile.

Additional information regarding the **IDBSERVERSERVICE** keyword and the database profile can be found in “HLAPI/2 Profiles, Environment Variables, and Data Logging” on page 95.

To identify the MVS host running the MRES with TCP/IP, your client application must specify the host in the **IDBSERVERHOST** keyword in the database profile. The host can be identified by an IP address in dotted-decimal format or by a host name. If you use a host name, the host name must be resolvable. Refer to the OS/2 online help for information on how TCP/IP should be configured to resolve a host name.

Additional information regarding the **IDBSERVERHOST** keyword and the database profile can be found in “HLAPI/2 Profiles, Environment Variables, and Data Logging” on page 95.

Preparing to Install HLAPI/2

Before you begin installing HLAPI/2, you should familiarize yourself with the important information in the **READ.ME** file shipped with HLAPI/2. If you are installing HLAPI/2 from the HLAPI/2 installation CD-ROM, the **READ.ME** file is on the CD-ROM; if you are installing HLAPI/2 from a LAN server, the **READ.ME** file is in the LAN subdirectory.

HLAPI/2 is delivered on a CD-ROM. You can install HLAPI/2 directly from the CD-ROM to a workstation or install on a LAN server and then to workstations from the LAN Server.

Note: Upgrades or patches that can be downloaded from a Tivoli Web site may be available for HLAPI/2. Visit the Tivoli Information Management for z/OS Web site **http://www.tivoli.com/infoman** for more information.

Installing HLAPI/2 on the Workstation

The source files for installation can reside in two places. You either install the files directly from the HLAPI/2 installation CD-ROM, or install them from the HLAPI/2 installation CD-ROM to a local area network (LAN) server, then install HLAPI/2 from the LAN server. And you can install the files in either attended mode or unattended mode.

If you already have an existing version of HLAPI/2 installed, delete this version before installing HLAPI/2. See “Deleting HLAPI/2 from Your Workstation” on page 92.

The default requester (**IDBREQ.EXE**) supports both APPC/APPN and TCP/IP communication with MVS and requires Communications Manager/2 and TCP/IP. You can choose a requester that only supports one of these. See “Choosing the Appropriate Requester” on page 90.

When installing HLAPI/2 on your workstation, you install the HLAPI/2 Installation and Maintenance Utility and the HLAPI/2 components.

You use the HLAPI/2 Installation and Maintenance Utility to perform the following tasks:

- Install HLAPI/2. See “Installing HLAPI/2 in Attended Mode from CD-ROM” on page 83 and “Installing HLAPI/2 on a Workstation in Unattended Mode” on page 86.
- Maintain HLAPI/2. See “Applying HLAPI/2 Maintenance” on page 91.
- Restore the previous service level. See “Restoring HLAPI/2 to the Previous Service Level” on page 91.
- Delete HLAPI/2. See “Deleting HLAPI/2 from Your Workstation” on page 92.

Installing HLAPI/2 in Attended Mode from CD-ROM

Install HLAPI/2 by following these steps:

Note: You must install HLAPI/2 on an HPFS drive.

1. Switch to, or start, an OS/2 window or an OS/2 full screen session.
2. If you already have HLAPI/2 installed, delete it. See “Deleting HLAPI/2 from Your Workstation” on page 92.
3. Insert the HLAPI/2 installation CD-ROM into a CD-ROM drive.
4. Type the following command at the OS/2 command prompt, then press Enter:

   ```
   e:\hlapi\os2\pc\install
   ```

   where:

   e Is the drive letter of the CD-ROM drive that contains the HLAPI/2 installation CD-ROM.
5. Read the information in the instructions window, then select **Continue**.
6. In the Install window, if you want the HLAPI/2 Installation and Maintenance Utility to update your **CONFIG.SYS** file, select **OK** and go on to step 7. If you do not select **OK**, changes are put in a file called **CONFIG.ADD**.

   If you do not want the HLAPI/2 Installation and Maintenance Utility to update your **CONFIG.SYS** file, do the following:

   a. De-select **Update CONFIG.SYS** before you select **OK**.

   b. Modify the **CONFIG.SYS** file manually before you shut down and restart your workstation or start HLAPI/2. Modify the **CONFIG.SYS** file using the information in the **CONFIG.ADD** file. It is in the same directory as your **CONFIG.SYS** file. This file is not created until the HLAPI/2 Installation and Maintenance Utility has completed its part of the installation. See “Customizing the HLAPI/2 CONFIG.SYS File” on page 90 for information about updating your **CONFIG.SYS** file.
7. In the Install - Directories window:

   a. Select the components of the HLAPI/2 you want to install. (See “Components of HLAPI/2” on page 339 for a complete list of HLAPI/2 files.)

   b. Type the target paths in which to install the HLAPI/2 files. You can accept the default values or change them. If the paths do not exist, they will be created. The default directory is C:\INFOAPI.

      Note: You can select **Disk space...** to determine the amount of available space on the fixed disk drives in your workstation.

   c. Select **Install...**

      The HLAPI/2 files are transferred from the HLAPI/2 installation CD-ROM to your workstation. The **Install - Progress** window indicates progress.
8. When the transfer is complete, a message appears to indicate that HLAPI/2 is installed. Select **OK**.
9. Optionally, choose the requester to be used. See “Choosing the Appropriate Requester” on page 90.

Note: If you update the **CONFIG.SYS** file during the installation, you must shut down your workstation and start it again before starting HLAPI/2.

The HLAPI/2 installation is complete. After you start your workstation again, verify the installation.

Installing HLAPI/2 on a LAN Server

To install HLAPI/2 on a workstation from a LAN server, you install the HLAPI/2 Installation and Maintenance Utility and the HLAPI/2 package files on the LAN server.

Note: If you choose to install the HLAPI/2 Installation and Maintenance Utility on a LAN, you must do so in attended mode. You cannot perform these steps in unattended mode.

To put the HLAPI/2 Installation and Maintenance Utility and HLAPI/2 package files on a LAN server, follow these steps:

1. Switch to, or start, an OS/2 window or an OS/2 full screen session.
2. If you already have HLAPI/2 installed, delete it. See “Deleting HLAPI/2 from Your Workstation” on page 92.
3. Insert the HLAPI/2 installation CD-ROM into a CD-ROM drive.
4. Type the following command at the OS/2 command prompt, then press Enter:

   ```
   e:\hlapi\os2\lan\install /p:"HLAPI/2 on LAN SERVER
   ```

   Note: For the DBCS version of OS/2, type a space followed by a left bracket symbol ([) at the end of the command.

   e Is the drive letter of the CD-ROM drive that contains the HLAPI/2 installation CD-ROM.
5. Read the information in the instructions window, then select **Continue**.
6. In the Install window, de-select the **CONFIG.SYS** option so that the **CONFIG.SYS** file is not updated.

   Note: Verify that the install window says **HLAPI/2 on LAN SERVER**. If not, then choose **CANCEL** and return to 4 and reenter the command exactly as shown.
7. In the Install - Directories window:
   a. Type the target path in which to install the HLAPI/2 files. You can accept the default value or change it. If the path does not exist, it will be created. The default directory is **C:\INFOAPIS**. This directory should be on a LAN Server drive.

      Note: You can select **Disk space...** to determine the amount of available space on the fixed disk drives in your workstation.
   b. Select **Install...**

      The HLAPI/2 files are transferred from the HLAPI/2 installation CD-ROM to your workstation. The Install - Progress window indicates progress.
8. When the transfer is complete, a message appears to indicate that HLAPI/2 has been installed. Select **OK**.

The HLAPI/2 installation to the LAN server is complete. Give all authorized users access to the LAN drive on which you installed HLAPI/2.

Installing HLAPI/2 in Attended Mode from a LAN Server

If you have access to a LAN server that holds HLAPI/2, you can access the alias drive/directory and install HLAPI/2 on your workstation by following these steps:

Note: You can only install HLAPI/2 on an HPFS drive.

1. Switch to, or start, an OS/2 window or an OS/2 full screen session.
2. Use the **CHDIR** (CD) command to change to the LAN server directory that contains HLAPI/2.
3. Type **INSTALL** on the command line
4. The Initial Installation dialog and an Information window appear. Read the information in the window, then select **Continue**.
5. In the Install window, if you want the HLAPI/2 Installation and Maintenance Utility to update your **CONFIG.SYS** file, select **OK** and go on to step 6. This is recommended.

   If you do not want the HLAPI/2 Installation and Maintenance Utility to update your **CONFIG.SYS** file, do the following:

   a. De-select **Update CONFIG.SYS** before you select **OK**.

   b. Modify the **CONFIG.SYS** file manually before you shut down and restart your workstation, or start HLAPI/2. Modify the **CONFIG.SYS** file using the information in the CONFIG.ADD file. It is in the same directory as your **CONFIG.SYS** file. This file is not created until the HLAPI/2 Installation and Maintenance Utility has completed its part of the installation. See “Customizing the HLAPI/2 CONFIG.SYS File” on page 90 for information about updating your CONFIG.SYS file.
6. In the Install - Directories window:

   a. Select the components of the HLAPI/2 you want to install. (See “Components of HLAPI/2” on page 339 for a complete list of HLAPI/2 files.)

   b. Type the target paths in which to install the HLAPI/2 files. You can accept the default values or change them. These directories will be created if they do not already exist.

   Note: Select **Disk space...** to determine the amount of available space on the fixed disk drives in your workstation.

   c. Select **Install...**

   The HLAPI/2 files are transferred from the LAN server to your workstation. The **Install - Progress** window indicates progress.
7. When the transfer is complete, a message appears to indicate that HLAPI/2 has been installed. Select **OK**.
8. Optionally, choose the requester to be used. See “Choosing the Appropriate Requester” on page 90.

Note: If you update the **CONFIG.SYS** file during the installation, you must shut down your workstation and start it again before starting HLAPI/2.

The HLAPI/2 installation is complete. After you start your workstation again, you can verify the installation.

Installing HLAPI/2 on a Workstation in Unattended Mode

You can automate the installation of HLAPI/2 on your workstation by using a response file and command line parameters. The response file contains the responses you would normally provide interactively. The command line parameters enable the installation to proceed without user intervention.

A sample response file is shipped with HLAPI/2. You can use this file as is or you can modify it to meet your needs. The sample response file is named **BLMIRFW.DAT** and is contained on the HLAPI/2 installation CD-ROM, or, if HLAPI/2 is installed on a LAN server, it is contained in the LAN server subdirectory that contains the HLAPI/2 code.

You can use the sample response file to install HLAPI/2 from either the HLAPI/2 installation CD-ROM or a LAN server to a workstation. Details about the response file keywords are included in “HLAPI/2 Response File Keywords” on page 87. Details about the command line parameters are included in “Command Line Parameters” on page 88.

Installing HLAPI/2 from CD-ROM in Unattended Mode

To install HLAPI/2 from the HLAPI/2 installation CD-ROM in unattended mode, follow these steps:

Note: You can only install HLAPI/2 on an HPFS drive.

1. Switch to, or start, an OS/2 window or an OS/2 full screen session.
2. Insert the HLAPI/2 installation CD-ROM into a CD-ROM drive.
3. If a previous version of HLAPI/2 is not already installed, type the following at the OS/2 command prompt:

   ```
   e:\hlapi\os2\pc\install /X /A:I /R:<path>BLMIRFW.DAT /L1:LOG
   ```

   Otherwise, type:

   ```
   e:\hlapi\os2\pc\install /X /A:U /R:<path>BLMIRFW.DAT /L1:LOG
   ```

   where:

   e Is the drive letter of the CD-ROM drive that contains the HLAPI/2 installation CD-ROM.

   For details about the command line parameters, see “Command Line Parameters” on page 88.
4. Optionally, choose the requester to be used. See “Choosing the Appropriate Requester” on page 90.

Installing HLAPI/2 from a LAN Server in Unattended Mode

To install HLAPI/2 from a LAN server in unattended mode, follow these steps:

Note: You can only install HLAPI/2 on an HPFS drive.

1. Switch to, or start, an OS/2 window or an OS/2 full screen session.
2. Use the **CHDIR (CD)** command to change to the LAN server directory that contains HLAPI/2.
3. If a previous version of HLAPI/2 is not already installed, type the following at the OS/2 command prompt:

```
INSTALL /X /A:I /R:<path>BLMIRFW.DAT /L1:LOG
```

Otherwise, type:

```
INSTALL /X /A:U /R:<path>BLMIRFW.DAT /L1:LOG
```

For details about the command line parameters, see “Command Line Parameters” on page 88.

4. Optionally, choose the requester to be used. See “Choosing the Appropriate Requester” on page 90.

HLAPI/2 Response File Keywords

The sample response file shipped with HLAPI/2 includes these keywords:

CFGUPDATE *(required)*
: Specifies whether the **CONFIG.SYS** file is automatically updated. Valid values for this keyword are:
  AUTO
  : Automatically updates **CONFIG.SYS**
  MANUAL
  : Does not update **CONFIG.SYS**

COMP
: Specifies the unique name of a component of the product to which the information passed applies. You can specify a maximum of 100 components. The **COMP** value must match the **NAME** keyword of the **COMPONENT** entry in the package file.

  Note: Do not use quotes around the component name, even when the name is more than one word with blanks between words.

DELETEBACKUP *(required)*
: Specifies whether to delete only the backup versions of the product or to delete the entire product. Valid values for this keyword are **YES** and **NO**.

  If you attempt to perform an unattended delete and the **DELETEBACKUP** is not present in the response file, the deletion fails with an EPFIE212 error.

  This keyword is required because an existing dialog requests this information in the attended mode.

FILE *(conditionally required)*
: Provides the new default path for the file directory. Use this keyword only for installation processing.

  The **FILE** value is used in place of the **FILE** keyword of the **PATH** entry in the package file. This keyword is required if you specify a **FILE** keyword in the **PATH** entry.

OVERWRITE *(required)*
: Specifies whether to automatically overwrite files during installation. Valid values for this keyword are **YES** and **NO**. This keyword is required for unattended processing.

SAVEBACKUP *(required)*
: Specifies whether to save a backup version of the product when it is updated. Valid values for this keyword are **YES** and **NO**. This keyword is required for unattended processing because an existing dialog requests this information in the attended mode.

WORK *(conditionally required)*
Provides the new default path for the data directory. The **WORK** value is used in place of the **WORK** keyword of the **PATH** entry in the package file. This keyword is required if you specify a **WORK** keyword in the **PATH** entry.

Command Line Parameters

The command line parameters you use to install HLAPI/2 in unattended mode are:

/A:<action>
Specifies the action of the **EPFINSTS.EXE** or **INSTALL.EXE**.

If you use this parameter, the main window of the installation is not displayed. If you do not use this parameter, the installation starts normally with all windows displayed.

Valid values for this parameter are:
D Delete
I Install
R Restore
U Update.

This example uses the /A parameter to specify an install:

```
/A:I
```

/G:<include path>
Specifies the drive and directory of the general response files that are included by the specific response file. An example of how this can be specified is:

```
/G:C:\infoapi
```

/L1:<error log>
Specifies the drive, path, and file name of the error log file.

All lines logged to the error file are prefixed with a time stamp. The time stamp has this format:

```
YYYY-MM-DD HH:mm:SS:ss
```

where:
YYYY is the year
MM is the month
DD is the day
HH is the hour
mm is the minute
SS is the second
ss in the hundredth of a second.

The date and time separators are the current user-defined settings in the Country object of the System Settings folder. The default separators are the dash sign (-) and the colon (:), respectively.

Following is an example of using the **/L1** parameter to create an **ERROR.LOG** file in the **C:\ABC** directory:

```
/L1:C:\ABC\ERROR.LOG
```

/L2:<history log>
Specifies the drive, path, and file name of the history log file.

All lines added to the history file are prefixed with a time stamp in this format:

```
YYYY-MM-DD HH:mm:SS:ss
```

where:
YYYY is the year
MM is the month
DD is the day
HH is the hour
mm is the minute
SS is the second
ss in the hundredth of a second.

The date and time separators are the current user-defined settings in the Country object of the System Settings folder. The default separators are the dash sign (-) and the colon (:), respectively.

If you do not specify this parameter, no history log is maintained.

Following is an example of using the **/L2** parameter to create a **HISTORY.LOG** file in the **C:\ABC** directory:

```
/L2:C:\ABC\HISTORY.LOG
```


/L3: /L4: /L5:<log files>
Each of these parameters can contain a drive, path, and file name of a log file.

/P:<product name>
Provides the name of the product for the specified action.

If you do not specify this parameter, the installation comes up normally with all windows displayed.

If your product name string includes any spaces, you must use double quotes around the word string. For example:

```
/p:"product name with spaces"
```

/R:<response file>
Specifies the drive, path, and file name of the specific response file.

The following search order is used to find the response files:

1. The fully qualified file specification
2. The current directory
3. The file name together with the **/G:** invocation parameter
4. Each directory in the **PATH** environment variable
5. Each directory in the **DPATH** environment variable.

The drive and path are optional for this parameter.

This is an example of using this parameter:

```
/R:L:\XYZ\RESPONSE.DAT
```

/S:<source_location>
Specifies the drive and path that contains the source files to be installed.

/T:<install target directory>
Specifies the drive and path into which the product files are installed. An example of using this parameter follows:

```
/T:C:\IBB
```

/TU:<update target CONFIG.SYS directory>
Specifies the drive and path of the target **CONFIG.SYS** to be updated.

If you do not specify this parameter, the **CONFIG.SYS** files are updated as specified in the product's package file.

This is an example of using this parameter:

```
/TU:C:\
```

/X Specifies that the action is unattended.

If you do not specify all of the information needed for the action to complete, an error occurs. When you specify the **/X** option, no progress indication is shown and all error messages are logged in the log file. You can specify the location of this error log file by using the **/L1** parameter.

If you do not specify this option, the user is prompted for any information that is needed to complete the action. In this attended mode of action, progress indication is shown and error messages are displayed to the user in secondary windows.

Choosing the Appropriate Requester

The default requester (**IDBREQ.EXE**) installed for HLAPI/2 requires Communications Manager/2 and TCP/IP to be installed. If you do not have both of these installed, you can use a requester that supports only one of the protocols.

To do so:

1. Switch to, or start, an OS/2 window or an OS/2 full screen session.
2. Change to the directory where HLAPI/2 is installed.
3. Type BLMREQI at the OS/2 command prompt.
4. Follow the directions that appear on the screen. The chosen requester is copied to **IDBREQ.EXE**.

Customizing the HLAPI/2 CONFIG.SYS File

If you chose not to have the HLAPI/2 Installation and Maintenance Utility update the **CONFIG.SYS** file for you, you must update certain entries yourself. The example statements in this section use **C:\INFOAPI** as the default product directory (to show where HLAPI/2 is installed). If you specified a different drive and directory during your installation, use that information instead.

If you installed the HLAPI/2 **RUNTIME** component, update the **LIBPATH** and **SET** statements. Add the name of the HLAPI/2 directory where the product's dynamic link library file (**DLL**) is installed to the **LIBPATH** statement. Add the name of the HLAPI/2 directory to the **SET** statement. The updated statements should look like these:

```
LIBPATH=C:\OS2\DLL;C:\MUGLIB\DLL;...;C:\INFOAPI\DLL;
SET DPATH=C:\OS2;C:\MUGLIB;...;C:\INFOAPI;
```

Note: The ellipses in these statements indicate information already in the statements that does not need to be changed.

Applying HLAPI/2 Maintenance

Perform the following steps:

1. Ensure that the CD-ROM is accessible in its drive.
2. Stop any application that uses HLAPI/2.
3. Stop the HLAPI/2 requester.
4. Change to the directory that contains the HLAPI/2 Installation and Maintenance Utility. Then enter the command **EPFINSTS** to start the HLAPI/2 Installation and Maintenance Utility.
5. In the Installation and Maintenance window:
   a. Select **HLAPI/2** or **HLAPI/2 on LAN SERVER**.
   b. Select **Update** from the **Action** pull-down.
6. In the Update window, select **Save a Backup Version?** to store the backup of the current level of the product. If you want to automatically update the CONFIG.SYS file, select **Update CONFIG.SYS**.
7. Select **Update**. The update operation begins. When the update finishes, the Update window closes and a message indicating that the update was successful appears.
8. Select **OK**. The Installation and Maintenance window appears.
9. Close the HLAPI/2 Installation and Maintenance Utility.

Restoring HLAPI/2 to the Previous Service Level

You can restore the previous service level of the HLAPI/2 if:

- HLAPI/2 was installed (and not deleted).
- HLAPI/2 was updated at least once.
- The HLAPI/2 files were stored in a backup directory during the update.

Restoring HLAPI/2 If It Was Installed from the HLAPI/2 CD-ROM

If you want to restore the previous service level and HLAPI/2 was installed directly from the HLAPI/2 installation CD-ROM, follow these steps:

1. Stop any application that uses HLAPI/2.
2. Stop the HLAPI/2 requester.
3. Start the HLAPI/2 Installation and Maintenance Utility. Use the **CHDIR** (**CD**) command to change to the directory that contains the HLAPI/2 Installation and Maintenance Utility. Then enter the command **EPFINSTS**.
4. In the Installation and Maintenance window:
   a. Select **HLAPI/2** or **HLAPI/2 on LAN SERVER**.
   b. Select **Restore** from the **Action** pull-down.
5. In the Restore window, select **Restore** to restore the most recent backup of the previous level of this product. The restore operation begins. When the restore finishes, the Restore window closes and a message indicating that the restore operation was successful appears.

6. Select **OK**. The Installation and Maintenance window appears.
7. Close the HLAPI/2 Installation and Maintenance Utility.

Restoring HLAPI/2 If It Was Installed from a LAN Server

If you want to restore the previous service level and HLAPI/2 was installed from a LAN server, follow these steps:

1. Stop any application that uses HLAPI/2.
2. Stop the HLAPI/2 requester.
3. Access the alias drive/directory on the LAN where HLAPI/2 is located and type `EPFINSTS`.
4. In the Installation and Maintenance window:
   a. Select **HLAPI/2**.
   b. Select **Restore** from the **Action** pull-down.
5. In the Restore window, select **Restore** to restore the backup of the previous level of this product. The restore operation begins. When the restore finishes, the Restore window closes, and a message indicating that the restore operation was successful appears.
6. Select **OK**. The Installation and Maintenance window appears.
7. Close the HLAPI/2 Installation and Maintenance Utility.

Deleting HLAPI/2 from Your Workstation

Follow the steps below to remove HLAPI/2 from your system.

1. Stop any application that uses HLAPI/2.
2. Stop the HLAPI/2 requester.
3. Start the HLAPI/2 Installation and Maintenance Utility. Use the **CHDIR** (**CD**) command to change to the directory that contains the HLAPI/2 Installation and Maintenance Utility. Then enter the command **BLMINSTS** (for a pre-Information/Management Version 6.3 version of HLAPI/2) or **EPFINSTS** from the OS/2 window.
4. In the Installation and Maintenance window:
   a. Select **HLAPI/2** or **HLAPI/2 on LAN SERVER**
   b. Select **Delete** from the **Action** pull-down
5. In the Delete window, review the information and select the components you want to delete. Then select **Delete** to begin the delete operation. The delete operation begins. The Status changes to **Deleting Files**. All the HLAPI/2 product files associated with the selected component are deleted from your workstation. Any backup service level files associated with the selected component are also deleted. A message appears indicating that the Delete operation is finished.

   Note: Installation and maintenance files are *not* deleted.
6. Select **OK**. The Installation and Maintenance window appears.
7. Close the HLAPI/2 Installation and Maintenance Utility by selecting **File** on the pull-down, then select **Exit**.

Diagnostic Assistance

A valuable tool to help you trace the network traffic between the HLAPI/2 client and the server is provided by OS/2 Communications Manager. The tool is called **CMTRACE**. It can only be used if you choose to use APPC communication to MVS via OS/2 Communications Manager. Follow these steps to use and format the CMTRACE:

1. Select the **CMTRACE** option in Communications Manager.
2. Select the items you want to trace.
3. Run your application.
4. Return to Communications Manager and stop **CMTRACE**.
5. Specify the file name for your trace. The default directory is **CMLIB**.
6. Access the **CMLIB** directory, and use the following command to format the trace output:
   ```
   FMTTRACE xxxx.xxx +d
   ```
   where **xxxx.xxx** is the file name you specified in step 5.
7. Your formatted output is stored as **xxxx.DET**. Use a browse utility to look at the file.

9

HLAPI/2 Profiles, Environment Variables, and Data Logging

Certain aspects of the HLAPI/2 interface can be tuned to the needs of your application. You do this by specifying profile keywords and values in two OS/2 text files–the system profile and the database profile.

The system profile is optional. It is specified in the command that starts the HLAPI/2 requester. See "The HLAPI/2 Requester" on page 105 for more information about this command. The system profile defines the sizes of the data buffers that are passed between the OS/2 workstation and the MVS host.

The database profile is required and must be specified in a control PDB that is passed to HLAPI/2 as part of an HL01 transaction. The database profile defines which MVS server is the destination of Tivoli Information Management for z/OS HLAPI transactions that are submitted by the user application. It controls all aspects of logging these transactions on the workstation, and it defines which ASCII and EBCDIC code pages to use for data conversion.

Profile Syntax

A profile can be created and manipulated with common text editors. The profile syntax is keyword driven. Keyword processing is not case-sensitive. The keywords can be entered with any mix of uppercase and lowercase characters. Each keyword requires a data parameter.

For each keyword, the equals character (=) separates the keyword from its data value. Optionally, one or more spaces may precede or follow the equals character or the keyword. The data value consists of all nonblank characters to the right of the equals character, up to the end of line. This is an example:

```
IDBServCharCodepage = 37
```

You can specify a comment; but it must be specified on a text line of its own. The comment must be preceded by the characters **REM** as in the following example:

```
REM Code page 37 is the EBCDIC U.S. English code page.
```

When you enter numbers for profile keyword data, *do not* use commas in the numbers; enter the numbers without spaces or any punctuation whatsoever:

```
IDBLogFileSize = 262144
```

System Profile Keywords

When the requester is started, it can be given a system profile file name. The system profile enables the application programmer to tune certain aspects of the client HLAPI/2 system.

The system profile file name can be fully qualified with its path and drive, or you can specify the file name only. If only a file name is specified, the search path name is obtained from the current value held in the **IDBSMPATH** OS/2 environment variable, described in “IDBSMPATH” on page 102.

If a profile is not specified, default values are used for each of the keywords. Information on **IDBINBOUNDBUFSIZE**, **IDBOUTBOUNDBUFSIZE**, and **IDBSHARECMS** can be found on this page and pages following.

An example of a System Profile is contained in “System Profile Example” on page 97.

IDBINBOUNDBUFSIZE

This value is the number of bytes to be allocated for each buffer that the workstation receives from the server. The number is rounded up to the next highest multiple of 4096 in all cases except at the uppermost range (greater than 28672). A buffer size greater than this, but less than 32767, is rounded up to 32767.

Note: Do not use commas when entering numbers in profile keywords.

An example of when to increase the **IDBINBOUNDBUFSIZE** is when you have an application that uses retrieve transactions containing a large amount of data.

Valid entries: any number between 1 and 32767.

Default value: 4096.

IDBOUTBOUNDBUFSIZE

This value is the number of bytes to be allocated for each buffer sent from the workstation to the server. The number is rounded up to the next highest multiple of 4096 in all cases except at the uppermost range (greater than 28672). A buffer size greater than this, but less than 32767, is rounded up to 32767.

Note: Do not use commas when entering numbers in profile keywords.

An example of when to increase the **IDBOUTBOUNDBUFSIZE** is when you have an application that uses create transactions that contain a large amount of data.

Valid entries: any number between 1 and 32767.

Default value: 4096.

IDBSHARECMS

This keyword determines whether the requester should enable or disable conversation sharing. When conversation sharing is enabled, the requester assigns new client applications to an existing conversation if criteria such as same server and same security ID are met. When conversation sharing is disabled (the default), each client application is assigned its own dedicated conversation. A conversation is terminated when the last client assigned to it submits an HL02.

Note: If you choose to use conversation sharing, you must be aware that there is a potential for a delay because transactions are handled synchronously. Thus, if Client A and Client B share a conversation, and Client A submits a long search and Client B submits an update, Client B will wait for Client A's search to complete before its transaction can be processed.

Note: If you are using pre-started API sessions (described in "MRES with Pre-started API Sessions Considerations" on page 18), you must disable conversation sharing.

Valid entries: 0 (conversation sharing disabled) or 1 (conversation sharing enabled).

Default value: 0 (conversation sharing disabled).

System Profile Example

```
REM*****************************************************************
REM
REM
REM
IDBINBOUNDBUFSIZE  = 4096
REM
IDBOUTBOUNDBUFSIZE = 4096
REM
IDBSHARECMS = 0
REM
```

Database Profile Keywords

Database profiles are created by the user. The database profile file name is passed to HLAPI/2 through a PDB named **DATABASE_PROFILE** on the control PDB list. This occurs when your application connects to the database using the HL01 transaction. The database profile is used only with the HL01 transaction. If you specify it for other transactions, it is ignored.

The database profile file name can be fully qualified with its path and drive, or you can specify the file name only. If only a file name is specified, the search path name is obtained from the current value held in the OS/2 environment variable **IDBDBPATH** described in "IDBDBPATH" on page 101.

During profile resolution, the contents of database profile text files and profile overrides are compiled together into a final collection of profile settings. This final collection is then used by the HLAPI/2. For more information about profile overrides, see "Profile Override" on page 101.

Within a profile, a keyword cannot be duplicated. If a keyword is duplicated, an error is reported, and processing ends.

Information about individual keywords can be found in the following sections:

- "IDBDataLogLevel" on page 98
- "IDBLogFileSize" on page 98
- "IDBLogFileNameActive" on page 98
- "IDBLogFileNameOld" on page 98
- "IDBCharCodePage" on page 98
- "IDBServCharCodePage" on page 99
- "IDBSymDestName" on page 99

- "IDBServerHost" on page 99
- "IDBServerService" on page 99

An example of a Database Profile can be found in "Database Profile Example" on page 100.

IDBDataLogLevel

The level at which client data logging is performed. This profile setting can be overridden with the OS/2 environment variable also called **IDBDataLogLevel**.

Valid entries: 0 (logging disabled) and 1 (logging enabled).

Default value: 0 (logging disabled).

IDBLogFileSize

The approximate maximum size in bytes of a log file. The log file is phased after it grows larger than this size. To *phase* a log file is to close the current log file, rename and archive it, and open a new empty log file.

Valid entries: Any positive integer between 4096 and 10485760 (do not include commas when entering numbers). If a value between 1 and 4095 is specified, 4096 is substituted. Specifying zero causes the log file to grow indefinitely.

Default value: 262144.

IDBLogFileNameActive

The primary name given to the active log file name for the client.

Valid entries: Any valid file name.

Default value: **IDB_LOG.ACT**. If you are running in a LAN environment, it is suggested that this file be written to a file unique to each LAN workstation to avoid errors due to file contention.

IDBLogFileNameOld

The name given to log files about to be archived. After the active log file as specified by **IDBLogFileNameActive** is phased, the file is renamed to this value.

Valid entries: Any valid file name.

Default value: **IDB_LOG.OLD**. If you are running in a LAN environment, it is suggested that this file be written to a file unique to each LAN workstation to avoid file contention errors.

IDBCharCodePage

This keyword indicates the code page to use in the client. Character data bound for the client is translated to this code page. Character data bound for the server is translated from this code page.

Valid entries: any code page listed in "HLAPI/2 Code Pages" on page 114.

Default value: the current code page. If you do not enter a value for this variable, HLAPI/2 interrogates the operating system for the current code page.

IDBServCharCodePage

This keyword indicates the code page that the server uses. Character data bound for the server is translated to this code page. Character data bound for the client is translated from this code page.

Valid entries: any code page listed in "HLAPI/2 Code Pages" on page 114.

Default value: 37 (U.S. English).

IDBSymDestName

Symbolic destination name. This keyword specifies the name of a Common Programming Interface for Communications (CPI-C) side information entry, which provides information required for HLAPI/2 to establish a conversation with a RES or an MRES with APPC. For more information on CPI-C side information entries, refer to the *OS/2 Communications Manager/2 Configuration Guide*. You can also refer to "Configuring and Running a Remote Environment Server (RES)" on page 25, "Configuring and Running a Multiclient Remote Environment Server (MRES) with APPC" on page 35, and "Configuring HLAPI/2 for APPC" on page 77 for additional information.

If you use this keyword, you must not use the **IDBServerHost** keyword.

Valid entries: Any CPI-C symbolic destination name (a 1- to 8-byte character string) defined in the Communications Manager/2 CPI-C side information entry.

Default value: none. This value is required if **IDBServerHost** is not specified.

IDBServerHost

This keyword identifies the MVS host that is running the MRES with TCP/IP server you want the requester to establish a conversation with. For more information about the MRES with TCP/IP and setting up the HLAPI/2 to communicate with an MRES with TCP/IP , refer to "Configuring and Running a Multiclient Remote Environment Server (MRES) with TCP/IP" on page 53 and "Configuring HLAPI/2 for TCP/IP" on page 81.

If you use this keyword, you must not use the **IDBSymDestName** keyword.

Valid entries: Any valid IP address in dotted-decimal format, or any valid host name, such as mvshost. If you specify a host name, the host name must be resolvable. Refer to the OS/2 online help for information on host name resolution.

Default value: none. This value is required if **IDBSymDestName** is not specified.

IDBServerService

This keyword identifies the service name of the MRES with TCP/IP server you want the requester to establish a conversation with. The service name must be listed in the **SERVICES** file in the **ETC** subdirectory on your workstation. For more information about the MRES with TCP/IP and setting up the HLAPI/2 to communicate with an MRES with TCP/IP, refer to "Configuring and Running a Multiclient Remote Environment Server (MRES) with TCP/IP" on page 53 and "Configuring HLAPI/2 for TCP/IP" on page 81.

Valid entries: Any valid service name or alias. Service names and aliases are *case-sensitive*.

Default value: *infoman*. **IDBServerService** is an optional keyword. If you do not specify it, when you specify **IDBServerHost**, the default is assumed. If you specify **IDBSymDestName**, **IDBServerService** is ignored.

Database Profile Example

```
REM*********************************************************************
REM
REM                        Example Database Profile
REM
REM
REM*********************************************************************
REM
REM -----------------------------------
REM Symbolic Destination Name (Required if using APPC)
REM -----------------------------------
IDBSymDestName      = LUNAME
REM
REM -----------------------------------
REM Host Name or TCP/IP Address (Required if using TCP/IP)
REM -----------------------------------
REM IDBServerHost       = yourhost
REM
REM -----------------------------------
REM MRES Service Name (Optional)
REM -----------------------------------
REM IDBServerService    = infoman
REM
REM *****  Specify Client and Server Code Pages *****
REM -----------------------------------
REM Client Character Code Page (Optional)
REM -----------------------------------
IDBCharCodepage     = 437
REM
REM -----------------------------------
REM Server Character Code Page (Optional)
REM -----------------------------------
IDBServCharCodepage = 37
REM
REM
REM *****  Specify Log File Parameters  *************
REM -----------------------------------
REM Client Data Log Level (Optional)
REM -----------------------------------
IDBDataLogLevel     = 1
REM
REM -----------------------------------
REM Log File Size (Optional)
REM -----------------------------------
IDBLogFilesize      = 262144
REM
REM -----------------------------------
REM The Active Log File Name (Optional)
REM -----------------------------------
IDBLogFileNameActive = IDBLOG.ACT
REM
REM -----------------------------------
REM The Old Log File Name (Optional)
REM -----------------------------------
IDBLogFileNameOld    = IDBLOG.OLD
```

HLAPI/2 OS/2 Environment Variables

HLAPI/2 uses OS/2 environment variables in two different ways. One environment variable can be set and used as a profile override. Other variables can be used to fully qualify the names of database and system profiles when a user does not do so. The means of doing a profile override is described in “Profile Override” and an explanation of Profile Search Path is described in “Profile Search Path”.

Profile Override

Profile override specifications enable certain profile values to be specified through OS/2 environment variables. Not all profile parameters that can be specified in a profile can be overridden by environment variables. **IDBDataLogLevel** is currently the only variable that can be overridden this way. You can use the OS/2 environment variable **IDBDataLogLevel** to override the database profile parameter **IDBDataLogLevel**. The value of the OS/2 environment variable always takes precedence.

By setting profile overrides in the **CONFIG.SYS** file, you can cause the profile override to effect all OS/2 sessions on the workstation.

By using the **SET** command on the command line, the values you specify are only in effect for a single OS/2 session (the current one). Values specified in this way override any values previously given in **CONFIG.SYS**. An example follows.

Assume this is part of your current database profile:

```
REM ** Connection to Information Management for z/OS BLX0
IDBSymDestName = OS2BLX0
REM ** Turn workstation logging on
IDBDataLogLevel = 1
REM ** Allow the workstation file to grow indefinitely
IDBLogFileSize = 0
REM ** Write the log file to BLX0TRAN.LOG
IDBLogFileNameActive = C:\INFOAPI\BLX0TRAN.LOG
```

This profile logs the data from all HLAPI transactions to the file **C:\INFOAPI\BLX0TRAN.LOG**. However, if you submit the command before you start the application that uses this profile, then no logging occurs on the workstation for transactions issued by the application. You have overridden the setting for **IDBDataLogLevel** for the current session. If you put the same command into **CONFIG.SYS**, you override the setting for all OS/2 sessions on this workstation.

Profile Search Path

Two environment variables are used to fully qualify the name of a database profile or system profile when the user does not do this. More information on **IDBDBPATH** can be found in “IDBDBPATH” and more information on **IDBSMPATH** can be found in “IDBSMPATH” on page 102.

IDBDBPATH

This is the search path to be used when a HLAPI/2 database profile name is specified without full qualification. If a user specifies a database profile name without a drive and path, HLAPI/2 first checks the current directory for the file with that name. If the database profile is not found there, HLAPI/2 searches the directories specified by the **IDBDBPATH** value. A sample default path is **C:\PROBLEM\HLAPI2**.

You can use this variable to specify multiple paths to search. For example

```
SET IDBDBPATH=C:\;C:\PROBLEM\HLAPI2\;
```

causes the C:\ directory to be searched first, followed by **C:\PROBLEM\HLAPI2**.

Valid entries: Any valid file path qualifier. The last backslash (\) is optional.

IDBSMPATH

This is the search path to be used when a HLAPI/2 system profile name is specified without full qualification. If a user specifies a system profile name without a drive and path, HLAPI/2 first checks the current directory for the file with that name. If the system profile is not found there, HLAPI/2 searches the directories specified by the **IDBSMPATH** value. A sample default path is **C:\HLAPI2\REQ**.

You can use this variable to specify multiple paths to search. For example:

```
SET IDBSMPATH=C:\;C:\HLAPI2\REQ\;
```

This causes the **C:** directory to be searched first, followed by **C:\HLAPI2\REQ**.

Valid entries: Any valid file path qualifier. The last backslash (\) is optional.

Server Logging

The content of the server log produced by HLAPI/2 is similar to that of one produced by the HLAPI. Each HLAPI/2 logical session that has logging enabled has its various transaction data, results, and messages logged as each transaction is completed on the host. For more information, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*.

Transaction Logging

Each started HLAPI/2 Tivoli Information Management for z/OS logical session has one log file. The database profile parameter, described in “IDBDataLogLevel” on page 98, (or the OS/2 environment variable used as an override) specifies whether data logging is enabled. If the parameter is not specified, then logging does not occur.

A logical session’s log entry on the host is identified by the **HLAPILOG_ID** PDB (refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*) passed on an HL01 transaction. This identifier is repeated for each transaction recorded in the server log.

The client writes the transaction to the log file specified by the database profile parameter, described in “IDBLogFileNameActive” on page 98, until it reaches the size (in bytes) specified by the database profile parameter, described in “IDBLogFileSize” on page 98. If you do not specify a log file name, the default name of **IDB_LOG.ACT** is used. The current log file is renamed to that specified by the parameter **IDBLogFileNameOld**, described in “IDBLogFileNameOld” on page 98. The default name for this parameter is **IDB_LOG.OLD**.

If an old log already exists it is deleted before the current log file is renamed. A new log file is created.

If two sessions are started specifying the same log file, then the first session that opens the log file has access to it, and the other session receives an error. When two sessions contend with each other for write access to the same log file, the following rules are followed to decide who can write to it.

- If the log file does not exist, then it is created and opened by the first session to ask for it.
- If the log file already exists, then it is opened and new log entries are appended to it by the first session.
- If the log file is already open, then this is not the first session to request it, and logging is not performed. The transaction continues to try to open the log until it reaches the internal retry limit, or is successful in opening the log. If it reaches the retry limit, then a return code and reason code are passed back in the HICA, indicating that logging was tried until the internal retry limit was reached.

When both HLAPI/2 and HLAPI logging are turned on, you may see differences in the PDBs that each one logs. The HLAPI/2 log shows any PDBs with a data length of zero. However, because HLAPI/2 does not send zero length PDBs to the server, the HLAPI log does not show any zero length PDBs. The HLAPI log also does not show the **SECURITY_ID**, **PASSWORD**, and **DATABASE_PROFILE** PDBs, because the HLAPI/2 does not send them to the server.

HLAPI/2 Error Logging

When HLAPI/2 encounters an unexpected logic or system error, it automatically creates or updates an error log file on the workstation's startup (IPL) drive. This log file is always written to the same place on your workstation, regardless of where the user application started. Called the **IDBPROBE.LOG**, it is found in the **\INFOAPI** subdirectory (**\INFOAPI\IDBPROBE.LOG**). If the **\INFOAPI** subdirectory does not already exist, HLAPI/2 creates it before writing the file. It provides more information about errors than can be returned in the HICA return and reason codes.

You can delete or rename the log file any time after it is created. If you do, HLAPI/2 creates a new error log file if new error information must be recorded.

10

The HLAPI/2 Requester

The HLAPI/2 requester is a program that must be running on the workstation before any HLAPI/2 activity can occur.

Starting the Requester

The requester program is typically started from within the **STARTUP.CMD** file on the OS/2 system. An optional system profile name is passed on the command line as a parameter when starting the HLAPI/2 requester program. Any OS/2 file name can be specified for the system profile name.

A sample system profile is copied to your workstation when the HLAPI/2 is installed. Look for **SYSTEM.PRO** in the **INFOAPI\SAMPLE** directory.

To start the HLAPI/2 requester, put the following line in the OS/2 **STARTUP.CMD** file:

```
IDBREQ [/P profile_file_name]
```

The command line parameters **/P** and the **profile_file_name** are optional. If you do not specify a file, default values are taken as specified in the system profile keyword list. The **profile_file_name** is preceded by a slash (**/**) followed by an uppercase or lowercase letter **P**. Separate the **/P** and the file name by at least one space.

You can also start the requester by placing a shadow of the HLAPI/2 Requester object that is in the Tivoli Information Management for z/OS folder into the OS/2 Startup folder or you can open the Tivoli Information Management for z/OS folder and double-click on the HLAPI/2 Requester object to start the requester. To specify a system profile name with either of these methods, you will need to modify the SETTINGS for the object and specify **/P** profile_file_name as a parameter.

The requester starts with a Presentation Manager (PM) window that shows standard Tivoli copyright information. Select **OK**, and another PM window briefly appears to indicate that the requester is running. This PM window is the *requester run time window*. It minimizes to an icon if there are no errors starting the requester.

Stopping the Requester

You end the requester program by pressing the **EXIT** button on the requester run time window. A confirmation panel appears where you select **OK** to indicate that you want to exit the program. This action starts a shutdown of the requester environment, ending all conversations and freeing resources. If you select **CANCEL** from the confirmation panel, your request to exit the program is dropped, and the requester continues to run.

The requester program can also be closed by bringing the OS/2 Task list window into focus and closing the process running the requester. However, the preferred method is to use the requester run time window.

Client user applications receive a *requester not started* return code (Return code=12, Reason code=109) for all transaction requests that occur after the requester is closed or before it is started.

Diagnosis of Some Common HLAPI/2 Problems

When attempting to diagnose unexpected results from your use of the HLAPI/2, the *Tivoli Information Management for z/OS Diagnosis Guide* can help you analyze Tivoli Information Management for z/OS. Some common problems that can occur with the HLAPI/2 specifically are discussed in this section.

Changing the Profile and Its Effect on Program Operation

Perhaps you made a change to your database profile variables and it seems that nothing has changed in the way the program runs. For example, you change the setting for **IDBDataLogLevel** in your profile. The expected result does not occur when you next use the program. Check the following:

- Check your user application to determine which profile it uses to perform its task. HLAPI/2 looks at the database profile and the system profile.
- Verify that no environment variable (as a profile override) takes precedence over the setting in your database profile. In the current example, check the setting for this variable in the **CONFIG.SYS** file. The variable setting there overrides the profile.

  You can use the OS/2 command **SET** to check current settings. If an override is in effect, check the **CONFIG.SYS** file or any **.CMD** file that you run to find the profile override.
- Check the user application that you are running to see if it is setting the value of the environment variable (and thus the override) directly.
- HLAPI/2 might be reading a different profile with the same name, but which is located in a different directory from the profile you have changed. Use the OS/2 command **SET** to check each directory listed in the **IDBDBPath** environment variable for a file with the same name as the file that you are changing.

Establishing a Conversation with the Host

If the server you are using is a RES or an MRES with APPC, APPC/MVS and APPC on your workstation can get out of synchronization if the host APPC/MVS is restarted while Communications Manager/2 is still active on your workstation. If they are not in synch, you might not be able to start a conversation. Stop and restart the Communications Manager/2 program on your workstation to synchronize it with the host.

Another source of failed connections can be the APPC Transaction Scheduler (ASCH) on the MVS host system. If you are not able to establish as many total sessions on your workstations as you expect to, check the **CLASSADD** command used to define the **APPC** class in the Transaction Scheduler that you are using for HLAPI/2. To increase the number of sessions, increase the **MAX** value of the command, or define a separate class for your transaction program to run under.

If the server you are using is an MRES with TCP/IP or an MRES with APPC, the server must be started before HLAPI/2 can access it.

Establishing Too Many APPC Conversations

If you attempt to establish too many concurrent conversations using either a RES or an MRES with APPC, you may reach the system limits for APPC. Conversations you try to establish after the limit is reached are suspended until an earlier one shuts down. To expand the limit, update the APPC settings to the desired number.

11

HLAPI/2 Transactions

The work done by the HLAPI/2 takes place through the use of HLAPI transactions. For a list of all the transactions that are available to the HLAPI/2, see "Transaction List" on page 135. Also, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* for an explanation of each transaction. Because the MVS and OS/2 environments are different, slight differences appear in the way the HLAPI/2 and the HLAPI use the same transaction. This section explains some differences to consider.

Transaction Operating Modes

A user application can select from two forms of transaction processing: synchronous or asynchronous. Users familiar with the Tivoli Information Management for z/OS Low Level Application Programming Interface (LLAPI) know that it selects either *all* synchronous or *all* asynchronous processing for Tivoli Information Management for z/OS database transactions within a session. With the HLAPI/2 you can select synchronous or asynchronous processing for a transaction at any time. This is also different from the HLAPI, which does not support asynchronous processing.

Synchronous Processing

Synchronous processing forces your user application's current thread to wait for a Tivoli Information Management for z/OS transaction to finish before it can perform any other work. The thread that submits the synchronous transaction does not receive control from the **IDBTransactionSubmit** HLAPI/2 service call until the transaction ends. You choose this mode of operation by coding a transaction type of **IDB_SYNC** (synchronous) on the **IDBTransactionSubmit** HLAPI/2 service call. Transaction completion includes both successful and unsuccessful outcomes.

You can implement the user application using the multitasking capabilities of OS/2 and use the synchronous mode of transaction processing. By using multiple threads within your application, one application thread can be dedicated to Tivoli Information Management for z/OS transaction processing while others perform other application duties. In this case, only the dedicated HLAPI/2 thread is blocked while the synchronous Tivoli Information Management for z/OS transaction finishes. Meanwhile, the other application threads continue to perform work.

CAUTION:

During synchronous processing, do not modify a HICA or PDB that you have submitted until after the IDBTransactionSubmit service call returns to the calling thread. Any changes to the HICA or its associated PDBs during transaction processing may cause unpredictable results.

Asynchronous Processing

Asynchronous processing enables your user application to submit a Tivoli Information Management for z/OS transaction and then continue performing application-related work while the submitted transaction finishes. After your application submits a transaction, control is immediately returned from the **IDBTransactionSubmit** service call to the application. Application processing and transaction processing occur concurrently. You choose this mode of operation by coding a transaction type of **IDB_ASYNC** (asynchronous) on the **IDBTransactionSubmit** HLAPI/2 service call.

After a transaction is submitted for asynchronous processing, your application must determine when the transaction finishes. Use the **IDBTransactionStatus** HLAPI/2 service call to do this. For every asynchronous transaction that returns an **IDBRC_NOERR** code from **IDBTransactionSubmit**, you must call the **IDBTransactionStatus** function to determine when the transaction ends or to change the transaction to a synchronous type.

Your user application can check for transaction completion either by polling the transaction or by converting the asynchronous transaction to a synchronous transaction. Polling is achieved by coding a transaction query type of **IDB_CHECKFORCOMPLETION** on the **IDBTransactionStatus** service call. If the transaction is finished, an **IDBTransactionStatus** of **IDB_TCOMPLETE** indicates completion. The data returned from an asynchronous transaction is not stored in the HICA until the status value **IDB_TCOMPLETE** is returned from an **IDBTransactionStatus** call. If the transaction is not finished, a status of **IDB_TINPROGRESS** indicates the transaction has not finished processing.

However, when HLAPI/2 no longer needs the conversation between the requester and the server to process a given transaction, the conversation is immediately free to be used by another transaction. This way, processing can start on another transaction with a different HICA using the same conversation, even before the **IDBTransactionStatus** call is performed for the current HICA.

If you want your application to convert the asynchronous transaction into a synchronous transaction, you can code a query type of **IDB_WAITFORCOMPLETION** on the **IDBTransactionStatus** service call. With this option, control is not returned from the **IDBTransactionStatus** service call until the transaction finishes. Transaction completion includes both successful and unsuccessful completions.

CAUTION:

During asynchronous processing, do not modify a HICA or PDB that you have submitted until after an IDBTransactionStatus service call returns the value of IDB_TCOMPLETE to the calling process. Any changes to the HICA or its associated PDBs during transaction processing may cause unpredictable results.

Data Conversion Characteristics

OS/2 uses the ASCII character set. MVS uses the EBCDIC character set. Thus, the HLAPI/2 requester and server each use a different character set, and character data exchanged between the host system and workstation requires conversion for the data to be useful in both environments.

HLAPI/2 converts the data at the workstation using the OS/2 system-provided functions **WinCpTranslateString** and **TrnsDt**. The function **WinCpTranslateString** converts

single-byte character set (SBCS) characters. The function **TrnsDt** converts mixed DBCS characters. A list of all source and target code pages supported by HLAPI/2 can be found in "HLAPI/2 Code Pages" on page 114.

Database Profile Parameters

The **IDBCharCodePage** parameter specifies the code page the client application is using. If this parameter is not specified, then the operating system is polled for the current code page.

The **IDBServCharCodePage** parameter specifies the code page the server is using. This code page is used to retrieve and store data in the Tivoli Information Management for z/OS database. If this parameter is not specified, then the default code page 37 (EBCDIC U.S. English) is used. See "Database Profile Keywords" on page 97 for more information about the database profile parameters.

Possible Truncation of Mixed SBCS/DBCS Data

Some HLAPI/2 data fields, such as the external record identifier (user defined) and privilege class, have a maximum defined size. The HLAPI/2 requester does not restrict mixed data from being entered (up to the maximum size) when such data is presented to HLAPI/2 for processing. During conversion from ASCII to EBCDIC, HLAPI/2 adds two bytes of data to each contiguous group of DBCS characters. If any data, after being converted between code pages by HLAPI/2, is larger than its maximum defined field size, then truncation occurs while maintaining proper DBCS truncation and padding.

Your application program must ensure that critical data is not lost because of DBCS truncation. Be sure that enough spaces appear at the end of each line of freeform text, or at the end of each data field, so that only spaces are truncated during data conversion.

Differences between HLAPI/2 and HLAPI Transactions

The HLAPI/2 requester enables a user application in the OS/2 environment to use many of the transactions available on the Tivoli Information Management for z/OS HLAPI. Some differences do exist between transactions originating on the host and those originating on the workstation.

One global consideration is that the HLAPI/2 does not support text data sets.

The HLAPI/2 also requires several PDBs not used by the HLAPI on the initialization transaction. They are:

SECURITY_ID --
your MVS userid (maximum length 8 characters)

PASSWORD --
your MVS password (maximum length 8 characters)

DATABASE_PROFILE --
Database Profile, described in "Database Profile Keywords" on page 97

Initialize Tivoli Information Management for z/OS (HL01)

The Tivoli Information Management for z/OS HLAPI transaction HL01 requests a connection to a database on a specific Tivoli Information Management for z/OS server. The steps that occur when a user application requests an HL01 transaction are outlined in this section.

1. The application author creates a database profile for the Tivoli Information Management for z/OS database connection. Use a text editor to create the database profile. See “Database Profile Keywords” on page 97 for a list of valid profile keywords. You should, but are not required to, create a unique database profile for each database connection that the application uses.

   Information obtained from the database profile includes the particular server you want to establish a conversation with. If you are using a RES or an MRES with APPC, then the database profile must include the symbolic destination name to use when establishing the APPC conversation to the server. If you are using an MRES with TCP/IP, then the database profile must include the host name or IP address of the MVS host where the server resides and optionally, you can specify a service name associated with the server.

2. The user application initializes a HICA structure for a given logical database. It inserts a **DATABASE_PROFILE** PDB onto the Control PDB chain to specify the name of the database profile that is to be used for the database connection.

   The database profile file name can be fully qualified (drive and subdirectory path), or just the name can be specified. If only the name is specified, the current directory is searched for the specified database profile. If it is not found in the current directory, the drive and path are obtained from the **IDBDBPath** environment variable.

3. Create a **SECURITY_ID** PDB and a P**ASSWORD** PDB, and any additional PDBs for the HL01 transaction, and place them on the Control PDB chain.

4. The user application submits the transaction for processing by using the HLAPI/2 service call **IDBTransactionSubmit**.

5. HLAPI/2 looks for the **DATABASE_PROFILE** PDB in the Control PDB chain in the HICA. The database profile is read, and the specifications are recorded for use throughout the specified logical Tivoli Information Management for z/OS session.

6. The server information is obtained from the database profile. If a symbolic destination name is specified, and a conversation associated with the symbolic destination name is not already established, it is established now. If a server host is specified, and optionally, a server service name, and a conversation associated with the server host and service is not already established, it is established now. If the HLAPI/2 requester was started with a system profile in which IDBSHARECMS was set to 1, an established conversation is used for the session when the same server is specified, and the same **SECURITY_ID** and **PASSWORD** are specified. Otherwise, a new conversation is established.

7. The **TIMEOUT_INTERVAL** PDB applies to the HLAPI that is running on Tivoli Information Management for z/OS. (Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* for more information.) If you specify a timeout interval, it determines the interval of the HLAPI running on the server. HLAPI/2 processing time and communications time between the workstation and the host are not considered for this timeout interval. Therefore, a transaction submitted from the workstation may take longer to time out than this value indicates.

8. Code pages for workstation and host processing are established when an HL01 transaction is submitted. They remain in effect until an HL02 (terminate) transaction is submitted for this particular HICA.

9. For information about logging, see “Server Logging” on page 102 and “Transaction Logging” on page 102.

10. The rest of the processing for the HL01 transaction is normal HLAPI/2 processing. See “Basic Transaction Flow” on page 72 for this description.

After you establish the main database connection with your first HL01 transaction, your application can issue multiple HL01 transactions in the same conversation to the server that is started. Each of these transactions creates a new database connection, and each of them performs tasks without affecting the others. Each of these connections can also be stopped without affecting any of the others. A graphic representation of this transaction nesting effect looks like this:

```
HL01 (initialize main session, session 1)
    HL01  (initialize session 2)
    HL01  (initialize session 3)
    HL02  (terminate session 2)
    HL02  (terminate session 3)
HL02  (terminate session 1)
```

Sessions 2 and 3 run independently of each other, and ending either of them does not affect session 1. However, multiple logical sessions on the same conversation can affect each other, because the requester waits on APPC or TCP/IP to send and receive a request before processing the next request.

Logical Session and Process Ownership

When an HL01 transaction is performed from within a client application, all further transactions associated with this HL01 transaction must be performed by the same process. The HICA and PDB chains can be shared across threads within the same process. For example, one thread in a client application can submit a transaction, and a different thread in the same application can process the results.

Terminate Tivoli Information Management for z/OS (HL02)

The Tivoli Information Management for z/OS HLAPI transaction HL02 closes a database connection on a specific Tivoli Information Management for z/OS server. The steps that occur when your user application requests an HL02 transaction follow.

1. The user application creates the normal Control PDBs requesting a disconnect from the logical database associated with the HICA and submits the transaction using the IDBTransactionSubmit service call.
2. Normal HLAPI/2 transaction processing requests a database disconnect from the corresponding Tivoli Information Management for z/OS server. See “Basic Transaction Flow” on page 72 for a description of this processing.
3. The HLAPI/2 examines the resulting return code to determine that the disconnect finished successfully. If this is the last active HL01 connection on this conversation, then the conversation is closed. If the conversation is with a RES, then the associated MVS address space is also closed. If this is not the last active HL01 connection, then the conversation continues.
4. The appropriate return codes are set and control returns to the caller.

Retrieve Record (HL06)

The difference between this transaction on the HLAPI/2 and the HLAPI is:

- The optional PDB called **TEXT_MEDIUM** supports only one storage media type for HLAPI/2. The only type supported is type **B**. If you omit this value or specify any character other than B, the HLAPI/2 assumes the value of **B**.

- If you want to retrieve freeform text as a continuous stream of data with carriage return / line feed characters (ASCII X'0D0A') after each text line, set the control PDB **TEXT_STREAM** to **YES**. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information about the **TEXT_STREAM** PDB.

Create Record (HL08)

The difference between this transaction on the HLAPI/2 and the HLAPI is:

- Text data sets are not supported. The **PDB_DATW** field in the input PDBs should always be specified with a nonzero value for text data.
- If you are creating a record that contains freeform text, and the input text contains either line feed characters (ASCII X'0A') or carriage return / line feed characters (ASCII X'0D0A'), set the control PDB **TEXT_STREAM** to **YES**. This will ensure that text formatting information is stored in the record. When the text is retrieved, it will be formatted exactly as it was entered. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information about the **TEXT_STREAM** PDB.

Update Record (HL09)

The difference between this transaction on the HLAPI/2 and the HLAPI is:

- Text data sets are not supported in HLAPI/2. The **PDB_DATW** field in the input PDBs should always be specified with a nonzero value for text data.
- If you are updating a record that contains freeform text, and the input text contains either line feed characters (ASCII X'0A') or carriage return / line feed characters (ASCII X'0D0A'), set the control PDB **TEXT_STREAM** to **YES**. This will ensure that text formatting information is stored in the record. When the text is retrieved, it will be formatted exactly as it was entered. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information about the **TEXT_STREAM** PDB.

HLAPI/2 Code Pages

Following is a list of all code pages supported by the HLAPI/2. Some of these code pages might *not* be supported by the particular OS/2 system that you work with. Some of the pages are supported only on a DBCS OS/2 system, while others are only supported on a SBCS OS/2 system. Check the OS/2 system you use to be sure which code pages it supports.

ASCII SBCS Code Pages

437 USA
850 Multilingual
852 Czechoslovakia/Hungary/Poland/Yugoslavia
857 Turkey
860 Portugal
861 Iceland
862 Hebrew-speaking
863 Canada (French-speaking)
864 Arabic-speaking
865 Norway
891 Korean
897 Japanese

903 S-Chinese
904 T-Chinese
1004 Desktop publish
1040 Korean Extended
1041 Japanese Extended
1042 S-Chinese Extended
1043 T-Chinese Extended

ASCII DBCS Code Pages

301 Japanese pure DBCS
926 Korean pure DBCS
927 T-Chinese pure DBCS
928 S-Chinese pure DBCS
932 Japanese
934 Korean
936 S-Chinese
938 T-Chinese
942 Japanese Extended
944 Korean Extended
946 S-Chinese Extended
948 T-Chinese Extended

EBCDIC SBCS Code Pages

37 USA/Canada (French)/Netherlands/Portugal
273 Austria/Germany
274 Belgian
277 Denmark/Norway
278 Finland/Sweden
280 Italy
282 Portugal
284 Latin America/Spain
285 United Kingdom
290 Japanese (Katakana) Extended
297 France
500 Multilingual
833 Korean Extended
836 S-Chinese Extended
870 Czechoslovakia/Hungary/Poland/Yugoslavia
871 Iceland
1026 Turkey
1027 Japanese (Latin) Extended

EBCDIC MIX Code Pages

930 Japanese (Katakana) Extended
931 English & Japanese Extended
933 Korean Extended
935 S-Chinese Extended
937 T-Chinese Extended
939 Japanese (Latin) Extended

EBCDIC PURE DBCS Code Pages

300 Japanese
834 Korean

835 T-Chinese
837 S-Chinese

12

Tips for Writing a HLAPI/2 Application

This chapter describes the steps typically involved in creating an application that uses the Tivoli Information Management for z/OS HLAPI/2. Every programmer has a certain technique or style for designing applications, so think of this chapter as more of a set of guidelines than as a set of rules. Refer to ″Tips for Writing An API Application″ and ″Tailoring the APIs″ in the *Tivoli Information Management for z/OS Application Program Interface Guide* for more information on this subject. Refer to “Choosing a Server” on page 13 for more information on the Tivoli Information Management for z/OS servers.

Determine What You Want Your Application to Do

The first step in creating an application is to determine exactly what you want it to do. After you decide that, consider:

- Which Tivoli Information Management for z/OS functions (for example, create or update) do you use?
- Which record types (for example, problem or change) do you use?
- Which fields (for example, status or assignee name) do you use?
- On which MVS system is your Tivoli Information Management for z/OS database located?
- Do you need to connect to more than one Tivoli Information Management for z/OS database?
- Are the databases on the same or different MVS systems?
- Which server do you use to access the databases?
- Do you want different OS/2 processes (or threads) to manage the different database connections, or do you want to use just one?
- How much storage do you need in the host address space to handle requests from the workstation?
- Do you want data logging enabled on the host? On the workstation?
- What data validation (if any) do you want to perform?

Converting C Programs

If you want to convert an existing C program that uses the HLAPI to a C program that uses HLAPI/2, here are some general instructions on how to do it.

- Make any general modifications to modify the program to run on OS/2, including fixing {} pairs that might have been mistranslated when the files were transferred from one

environment to another, and modifying the parameters passed to the main procedure. This step is required if the download program you use to copy the code from MVS to OS/2 translates the source improperly.

- Be sure you:
  - Include **IDBH.H** header file. *Do not* include **BLGUHLC** header file.
  - Include **IDBECH.H** if you want constant declarations of HLAPI/2 return codes.
  - *Do not* include **spc.h**.
- Delete #pragmas used for the MVS program.
- Convert any data set method freeform text processing for HL06, HL08, and HL09 to use buffer method freeform text processing.
- Convert HLAPI call syntax to HLAPI/2 format: use **IDBTransactionSubmit** and **IDBTransactionStatus**. Do not define variables to point to the **BLGYHLPI** module. Do not call the **BLGYHLPI** module.
- Add processing to create and initialize the three HLAPI/2-specific control PDBs: **SECURITY_ID**, **PASSWORD**, and **DATABASE_PROFILE**. Add them to the control PDB linked list.
- Build a database profile for use with the HLAPI/2 sessions you will be starting (or multiple database profiles if necessary).
- Review the error handling you use after the call to the HLAPI to see if changes are required to handle the error codes specific to the HLAPI/2.
- If you are using an MRES with APPC or a RES:
  - Set up the CPI-C side information entries for Communication Manager/2 to communicate with the particular MVS host. Specify these names in the database profile.
  - Set up APPC/MVS on the host to accept conversations from the workstations that your application runs on.
  - Set up the MRES with APPC or the RES on the host.
- If you are using an MRES with TCP/IP:
  - Set up TCP/IP to communicate with the particular MVS host, and the particular MRES you want to use. Specify the information in the database profile.
  - Set up the MRES with TCP/IP on the particular MVS host you want to use.

Installation and Setup Summary for HLAPI/2 Sample Applications

1. If you are using a RES or an MRES with APPC, install and configure APPC/MVS and Communication Manager/2 to connect your OS/2 workstation to your MVS host system. If you are using an MRES with TCP/IP, configure TCP/IP to connect your OS/2 workstation to your MVS host system.
2. If you are using a RES, create a TP profile on the MVS host that brings up Tivoli Information Management for z/OS and the Remote Environment. This must include the appropriate BLX-SP load module. Refer to the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* and *Tivoli Information Management for z/OS Operation and Maintenance Reference* manuals, or “Configuring and Running a Remote Environment Server (RES)” on page 25 in this manual, for more information.

3. If you are using an MRES with APPC or an MRES with TCP/IP, create a cataloged procedure with the JCL to start the MRES. If you are using an MRES with APPC, define the MRES to APPC and VTAM. The MRES must be started before you attempt to establish a conversation with it. See “Configuring and Running a Multiclient Remote Environment Server (MRES) with APPC” on page 35 and “Configuring and Running a Multiclient Remote Environment Server (MRES) with TCP/IP” on page 53 for additional information.
4. Install Tivoli Information Management for z/OS including building a session parameters member. You can also build the session parameter **BLGSES00** at this time, which enables you to skip the modification to the sample application for the session member.
5. Bring up Tivoli Information Management for z/OS as an interactive user using the session parameters member that you want the sample application to use. This ensures that Tivoli Information Management for z/OS can be brought up using that session member.
6. Install HLAPI/2. Refer to “Installing and Configuring HLAPI/2” on page 77 for information about how to do this.
7. Set up a database profile for use by the sample program. A sample database profile is copied to your OS/2 workstation during the installation of HLAPI/2. This profile is in the **SAMPLE** subdirectory in whatever directory you chose for HLAPI/2. Look for **DATABASE.PRO**. Be sure to modify the symbolic destination name if you are using APPC or modify the server host and server service names if you are using TCP/IP to fit your particular system.
8. To complete the setup and run the sample program, you will need to perform additional steps listed in the corresponding section, “Steps Required to Run the HLAPI/2 C Sample Program” on page 127 or “Steps Required to Run the REXX HLAPI/2 Sample Program” on page 134.

13

HLAPI/2 C Language Application Program

The HLAPI/2 interface enables you to access the HLAPI transactions from an OS/2 C language application. A C language client application program communicates with the Tivoli Information Management for z/OS system by creating a high-level application communication area (HICA) and its related parameter data blocks (PDBs). The client application program then submits the HICA transaction by making HLAPI/2 program service calls. The HLAPI/2 program service routines exist on the user's workstation as a dynamic link library (DLL). When the calls are made by the client application program, the supporting HLAPI/2 routines are loaded from the DLL and started.

Allocating HICAs and PDBs

To utilize HLAPI/2 program service calls, your user application must include a C language-based header file (named IDBH.H) in its source file and specify **IDBHLAPI.LIB** as one of the link libraries. Linking to **IDBHLAPI.LIB** gives you access to the HLAPI/2 service calls. To use the service calls, your user application must:

- Allocate HICA and PDB structures using the types provided in the header file
- Include the code for the HLAPI/2 program service calls.

The HICA and PDB data structures that you submit to HLAPI/2 must exist for the entire time that a transaction using them is being processed. Make sure that your program does not accidentally deallocate them, such as by using local variables in a subroutine, then exiting the subroutine while HLAPI/2 is processing your request.

The required header file **IDBH.H** provides the type definitions for both the HICA and PDB data structures. See "Allocating and Initializing a HICA" on page 122 and "Allocating and Initializing a PDB" on page 122 for more information.

Including the Header File in Your Program

Before using HLAPI/2, you must first include the HLAPI/2 header file into your program source. This file identifies the HLAPI/2 call prototypes and the various type definitions and constants your program uses to communicate information to the HLAPI/2-connected system. To include the HLAPI/2 header file into your source file, put the following line into your source file before you make any HLAPI/2 service calls and before you define any HICAs or PDBs.

```
#include  "IDBH.H"   /* Include the HLAPI/2 header file. */
```

The text of the header file appears in "HLAPI/2 Header Code" on page 127.

Allocating and Initializing a HICA

Your application must allocate and initialize at least one HICA data structure to communicate through the HLAPI/2. The values you put in the control and input PDB chains depend on the specific HLAPI/2 transaction you want to use. PDB allocation is discussed in "Allocating and Initializing a PDB". The way in which the user application initializes the control and input PDB chains is covered in the *Tivoli Information Management for z/OS Application Program Interface Guide* under the individual transaction discussions. This example shows part of an application that illustrates the declaration and partial initialization of a HICA structure for HLAPI/2.

```
     /* Allocate a HICA structure.  */
     static HICA     MyHICA;

     /* Initialize HICA eyecatcher to "HICA".  */
     memcpy(MyHICA.ACRO, HICAACRO_TEXT,
            HICAACRO_MAX_SIZE);

     /* Initialize the HICA fields to NULL  */
     MyHICA.ENVP = NULL;     /* pointer to environment block  */

     MyHICA.OUTP = NULL;     /* pointer to output pdb chain   */

     MyHICA.CTLP = NULL;     /* pointer to control pdb chain  */

     MyHICA.INPP = NULL;     /* pointer to input pdb chain    */

     MyHICA.MSGP = NULL;     /*  pointer to message pdb chain */

     MyHICA.ERRP = NULL;     /*  pointer to error pdb chain   */

     MyHICA.LENG = sizeof(HICA);   /*  HICA data block length */
#include  "IDBH.H"   /* Include the HLAPI/2 header file. */
```

An alternate method of allocating the HICA structure using typical C language calls is:

```
pMyHICA = malloc(sizeof(HICA));
```

Addressing the elements of the HICA then follows the **'pMyHICA->'** (pointer) notation rather than the **'MyHICA.'** (dot) notation.

A full description of the contents of a HICA appears in the *Tivoli Information Management for z/OS Application Program Interface Guide* in the section about HLAPI structures.

Allocating and Initializing a PDB

Parameter information is communicated between the user application and the HLAPI/2-connected system through PDBs. PDBs can be several types: control, input, output, error, or message. All PDBs of the same type are organized as a linked list. Each linked list type is anchored to a specific field within a HICA. For a description of the HICA fields, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*.

Your user application must create any required control and input type PDBs before it submits a transaction. Your user application should examine the output, error, and message PDB chains that are returned when a transaction completes processing.

Part of an application follows. It illustrates the declaration and initialization of an input PDB chain containing one PDB

Note: The PDB data structure has a variable length whose value can be very large. To conserve storage, the example below shows allocation of the PDB with only as much storage as it needs. The total storage in this case is the sum of the length of the data fields all of the PDBs have, plus the length of the variable field at the end of the PDB. The file **IDBH.H** contains the variable **PDBFIX_SIZE** to make this programming more convenient.

```
/* Temporary PDB pointer variable.  */
PPDB     pTempPDB;

/* Allocate the PDB memory.  */
pTempPDB = malloc(PDBFIX_SIZE + strlen("John Doe"));

/* Because this is the first PDB, init the NEXT PDB  */
/* pointer to null. */
pTempPDB->Next = NULL;

/* Init the previous PDB pointer to null.   */
pTempPDB->Prev = NULL;

/* set the PDB eyecatcher to "PDB "  */
memcpy(pTempPDB->Acro, PDBACRO_TEXT,
       PDBACRO_MAX_SIZE);

/* Set the parameter data item name to REPORTER_NAME */
memset(pTempPDB->Name, ' ', PDBNAME_MAX_SIZE);
memcpy(pTempPDB->Name, "REPORTER_NAME",
       strlen("REPORTER_NAME"));

/* fill the parameter with the name John Doe  */
memcpy(pTempPDB->Data, "John Doe",
             strlen("John Doe"));

/* Perform data response validation.   */
pTempPDB->Proc = 'V';

/* Initialize the PDB error code to a blank.   */
pTempPDB->Code = ' ';

 ... and the rest of the PDB is initialized in similar ways ...


 /* Place the new PDB on the INPUT PDB chain */
 MyHICA.INPP = pTempPDB;
```

Binding Prototypes

The two entry points into HLAPI/2 are **IDBTransactionSubmit**, described in “IDBTransactionSubmit” on page 124 and **IDBTransactionStatus**, described in “IDBTransactionStatus” on page 124.

Your application must allocate data structures conforming to the data types defined in the **IDBH.H** header file. HLAPI/2 program options must be specified by using the constants also supplied by the **IDBH.H** header file. Descriptions of the bindings and related data types follow.

Note: Fields are specified differently for the HLAPI/2 HICA and PDB structures than for those in the HLAPI. In the OS/2 client environment, field names are similar, but they might have been extended to conform with standard C language naming conventions. For more information about the HICA and PDBs, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*.

IDBTransactionSubmit

The **IDBTransactionSubmit** program call is used to submit a transaction to the HLAPI/2 system. In a C language program, the call looks like the following:

```
rc = IDBTransactionSubmit(pHICA, TranType);
```

The two variables that your application must provide are:

pHICA
: A pointer to a structure of the type HICA that contains the HICA that you want to submit to HLAPI/2.

TranType
: Your selection of the type of transaction to perform. This variable has a type definition of **TRANTYPE_TYPE**, and can be one of these two values:
  IDB_SYNC --- Synchronous transaction processing
  IDB_ASYNC -- Asynchronous transaction processing

HLAPI/2 returns a value from this function call that you should examine before looking at the HICA return and reason codes. This return code (rc) is a variable of type **IDBRC_TYPE**. The values that can be returned for it are described in “HLAPI Service Call Return Codes” on page 367. Here is an example of how this routine might be used:

```
#include "idbh.h"
#include <OS2.H>
#include <stdio.h>
#include "MYCODE.H"

main()
{
HICA MyHICA;
IDBRC_TYPE rc;

Initialize_HICA(&MyHICA);         /* routine that sets up the data */
                                      /* for an HL01 transaction.     */

rc = IDBTransactionSubmit(&MyHICA,IDB_SYNC);
switch (rc)
  {
  case (IDBRC_NOERR) : Main_Loop(&MyHICA);  /* The rest of your    */
                                              /* application program.*/
                     break;
  case (IDBRC_XERR)  : Process_Error(&MyHICA);  /* A non-zero HICA */
                                      /* return and reason code was    */
                                      /* detected.                     */
                     break;
  case (IDBRC_BADHICA): printf("The HICA data structure is corrupt\n");
                      return(-1);
  case (IDBRC_BADPARM): printf("Incorrect value passed on call\n");
                      return(-2);
  case (IDBRC_SYSERR) : printf("HLAPI/2 encountered a System Error\n");
                      printf("Check the file IDBPROBE.LOG.\n");
                      return(-3);
  }
}
```

IDBTransactionStatus

The **IDBTransactionStatus** program call is used to request the status of an asynchronous transaction. It can also be used to convert an asynchronous transaction to a synchronous transaction. In a C language program, the call looks like this:

```
rc = IDBTransactionStatus(pHICA, QueryType, pTStatus);
```

The three variables that your application must provide are:

- **pHICA**

  A pointer to a structure of type HICA that contains the HICA whose status you want to check.

- **QueryType**

  This input value is of the type **QUERYTYPE_TYPE**, and has one of the following two values:

  - **IDB_CHECKFORCOMPLETION**

    This value causes HLAPI/2 to check the current status of a transaction submitted on the HICA provided in pHICA. This program call returns to your application immediately, no matter what state the transaction is in.

  - **IDB_WAITFORCOMPLETION**

    This value causes HLAPI/2 to change the asynchronous transaction to a synchronous transaction. This program call does not return to your application until the transaction processing has finished.

- **pTStatus**

  This is a pointer to a variable of type **TRANSTATUS_TYPE**. It can have one of the following two values:

  - **IDB_TCOMPLETE**

    The transaction has completed processing. Any Output, Error, and Message PDBs returned from the host are attached to the HICA and available for your program's use. The **RETC** and **REAS** fields of the HICA are set with values indicating the result of the transaction (see *Tivoli Information Management for z/OS Application Program Interface Guide* for possible values).

  - **IDB_TINPROGRESS**

    The transaction is still in progress. No data has been returned on the HICA, and the HICA is unavailable for use by your application.

HLAPI/2 returns a value from this function call which you should examine before looking at the HICA return and reason codes. This return code (rc) is defined as a variable of type **IDBRC_TYPE**. The values that can be returned appear in "HLAPI Service Call Return Codes" on page 367. Here is an example of how this routine might be used:

```
#include "idbh.h"
#include <OS2.H>
#include <stdio.h>
#include "MYCODE.H"

main()
{
HICA MyHICA;
TRANSTATUS_TYPE MyStatus;
IDBRC_TYPE rc;

Initialize_HICA(&MyHICA);   /* routine that sets up the data   */
                                /* for an HL01 transaction.       */

rc = IDBTransactionSubmit(&MyHICA,IDB_ASYNC);  /* Use Asynchronous */
                                /* processing so this code has the */
```

```
                                  /* ability to maintain a PM window */
   if (rc!=IDBRC_NOERR)           /* Detected a non-zero return code */
      Process_rc(rc,&MyHICA);
   else
     {
     MyStatus = IDB_TINPROGESS;    /* Start the while loop out right  */
     while(MyStatus == IDB_TINPROGRESS)
       {
       rc = IDBTransactionStatus(&MyHICA,
                                 IDB_CHECKFORCOMPLETION,
                                 &MyStatus); /* Keep checking this*/
                                  /* until MyStatus == IDB_TCOMPLETE  */
       if (rc != IDBRC_NOERR)
         Process_rc(rc,&MyHICA);
       Process_PM_Window();       /* Handle the PM window duties      */

 /* Note:  If you do not have time-critical duties to perform in this */
 /*        loop, you may want to use the DosSleep() routine, or some  */
 /*        similar routine to prevent 100% CPU usage.                 */

       }
     }
  .
  .
  .
  }
```

Linking Your Program

Before using HLAPI/2, you must link to **IDBHLAPI.LIB**. This is an import library that resolves the external references to the HLAPI/2 service routines you use to perform HLAPI/2 functions.

The **BLM2SAMI.CMD** file in the **SAMPLE\C** subdirectory of the directory in which you installed HLAPI/2 contains:

```
ICC BLM2SAM1.C IDBHLAPI.LIB
```

It illustrates one way to link **IDBHLAPI.LIB** with your application using VisualAge C++ for OS/2.

Sample HLAPI/2 C Program

A sample program **BLM2SAM1.C** is installed as part of the installation of HLAPI/2. Look for it in the directory **SAMPLE\C**, which is a subdirectory of the directory where HLAPI/2 is installed. The sample shows the setup and start of HLAPI/2. It includes the header file **IDBH.H**. The sample C code performs the following functions:

1. Initialize the HLAPI by performing an HL01 transaction.
2. Obtain a system-assigned record ID and save it to use for the create transaction. (This step is not mandatory because the HL08 transaction generates a record ID if one is not specified).
3. Create a record using the previously obtained record ID.
4. Update several fields in the record just created.
5. Retrieve the record just created and updated, and print the fields just retrieved.
6. Search for all records created today by this program and print the search results.
7. End the HLAPI with an HL02 transaction.

8. Perform cleanup.

Steps Required to Run the HLAPI/2 C Sample Program

1. Perform the steps described in "Installation and Setup Summary for HLAPI/2 Sample Applications" on page 118.
2. Modify the sample program **BLM2SAM1.C**. You may want to back up **BLM2SAM1.C** before making any changes to it. This program is in the **SAMPLE\C** subdirectory in the directory in which you installed HLAPI/2. The default installation directory is **C:\INFOAPI**.
   - Find the **SESSMBR** #define near the beginning of **BLM2SAM1.C**. Change the value of this #define to the name of the session parameters member you want to use. If you want to use **BLGSES00**, no changes are required.
   - Find the **PRIVCLAS** #define. Change the value of this #define to the name of the privilege class to be used on the HL01 (initialize) transaction. This privilege class must be in the database defined in your chosen session parameters member. The privilege class must have authority to display, create, and update problem records.

     If you want to use **MASTER** privilege class, no changes are required.
   - Find the **APPLID** #define. Change the value of this #define to the name of the application ID to be used on the HL01 (initialize) transaction. The value you choose must be defined as an eligible user in the privilege class you use.
   - Find the **SECID** and **PASSWORD** #defines. Put in the appropriate values for the security ID and password for the ID you want to use on the MVS system.
   - Find the **DBPROF** #define. Use the name of the database profile or the sample profile shipped with HLAPI/2 (**DATABASE.PRO**).

     The database profile must be in the subdirectory where the sample program resides or in the path defined on your system in the **IDBDBPATH** variable. See "IDBDBPATH" on page 101 for more information about this variable.
3. To compile and link **BLM2SAM1.C** using VisualAge C++ for OS/2, run **BLM2SAMI.CMD**. First, verify that the OS/2 environment variable **LIB** contains the directory that holds **IDBHLAPI.LIB** and that the OS/2 environment variable **INCLUDE** contains the directory that holds **IDBH.H**.
4. Start the HLAPI/2 requester. See "The HLAPI/2 Requester" on page 105 for more information.
5. If you are running a RES or an MRES with APPC, be sure that Communications Manager/2 is running. If you are running an MRES with TCP/IP, be sure that TCP/IP for OS/2 is running.
6. Run the program **BLM2SAM1**.

HLAPI/2 Header Code

This is the text of the HLAPI/2 header. The file name is **IDBH.H** and it can be found in the **INFOAPI\H** sudirectory in the directory where HLAPI/2 is installed on your workstation.

```
/*                                                                */
/*                                                                */
/*  Source file: IDBH.H                                           */
/*                                                                */
/*  FUNCTION: This header file contains all public function       */
/*            prototypes  and structure definitions for the       */
```

```
/*           language bindings.                                    */
/*                                                                 */
/*   USAGE NOTE: All allocations should be made using a "packed"   */
/*         attribute. That is, no alignment should be used         */
/*         that would insert empty bytes between fields.           */
/*                                                                 */
/*         If the compiler you are using has an optimization       */
/*         function which you intend to use, you must allocate the */
/*         HICA and PDB structures as "volatile" objects. This     */
/*         ensures that all program statements that write data to  */
/*         these structures in your program are complete when      */
/*         your program submits the HICA data structure to HLAPI/2. */
/*                                                                 */
/*         Below is a list of atomic types and their storage sizes. */
/*                                                                 */
/*         "unsigned long"      4 byte unsigned integer            */
/*         "unsigned char"      1 byte unsigned integer            */
/*         "void *       "      4 byte pointer                     */
/*                                                                 */
/*******************************************************************/

#ifndef __idbh_h
#define __idbh_h

#ifdef __cplusplus
extern "C"
  {
#endif/**********************************************************************/
/*-----------------------------------------------------------------*/
/* PDB Type and General definitions.                               */
/*-----------------------------------------------------------------*/
#define   PDBACRO_MAX_SIZE       4
#define   PDBNAME_MAX_SIZE       32
#define   PDBACRO_TEXT           ("PDB ")
#define   PDB_SIZE               (sizeof(PDB))
#define   PDBFIX_SIZE            (sizeof(PDB)-sizeof(PDBDATA_TYPE))

#pragma pack(1)
typedef   struct _PDB   *PPDB;    /* Forward reference.            */

typedef   PPDB                PDBNEXT_TYPE;
typedef   PPDB                PDBPREV_TYPE;
typedef   unsigned char       PDBACRO_TYPE[PDBACRO_MAX_SIZE];
typedef   unsigned char       PDBNAME_TYPE[PDBNAME_MAX_SIZE];
typedef   unsigned char       PDBTYPE_TYPE;
typedef   unsigned char       PDBPROC_TYPE;
typedef   unsigned char       PDBCODE_TYPE;
typedef   unsigned char       PDBRSV1_TYPE[5];
typedef   unsigned long       PDBDATW_TYPE;
typedef   unsigned long       PDBAPPL_TYPE;
typedef   unsigned long       PDBDATL_TYPE;
typedef   unsigned char       PDBDATA_TYPE[1];
#pragma pack()


/*-----------------------------------------------------------------*/
/* PDB   -  structure definition.                                  */
/*-----------------------------------------------------------------*/
#pragma pack(1)
typedef struct  _PDB
  {
    PDBNEXT_TYPE      Next;   /* Address of next PDB in chain.       */
    PDBPREV_TYPE      Prev;   /* Address of previous PDB in chain.   */
    PDBACRO_TYPE      Acro;   /* The acronym "PDB "                  */
    PDBNAME_TYPE      Name;   /* Parameter data symbolic name        */
    PDBTYPE_TYPE      Type;   /* Parameter data type                 */
    PDBPROC_TYPE      Proc;   /* Parameter data processing flag      */
```

```
    PDBCODE_TYPE      Code;   /* Parameter data error code           */
    PDBRSV1_TYPE      RSV1;   /* Reserved                            */
    PDBDATW_TYPE      Datw;   /* Parameter data unit width           */
    PDBAPPL_TYPE      Appl;   /* Application use field               */
    PDBDATL_TYPE      Datl;   /* Length of parameter data            */
    PDBDATA_TYPE      Data;   /* Parameter data                      */
  }
         PDB;                 /* Type - Structure.                   */
#pragma pack()
/*-------------------------------------------------------------------*/
/* HICA Type and General definitions.                                */
/*-------------------------------------------------------------------*/
#define   HICAACRO_MAX_SIZE       4
#define   HICAACRO_TEXT          ("HICA")

typedef   unsigned char       HICAACRO_TYPE[HICAACRO_MAX_SIZE];
typedef   unsigned long       HICALENG_TYPE;
typedef   unsigned long       HICARETC_TYPE;
typedef   unsigned long       HICAREAS_TYPE;
typedef   void *              HICAENVP_TYPE;
typedef   PPDB                HICACTLP_TYPE;
typedef   PPDB                HICAINPP_TYPE;
typedef   PPDB                HICAOUTP_TYPE;
typedef   PPDB                HICAMSGP_TYPE;
typedef   PPDB                HICAERRP_TYPE;
typedef   unsigned long       HICASTPA_TYPE;
typedef   unsigned long       HICACRRC_TYPE;
typedef   unsigned char       HICARESV_TYPE[32];


/*-------------------------------------------------------------------*/
/* HICA  -  structure definition.                                    */
/*-------------------------------------------------------------------*/
#pragma pack(1)
typedef struct  _HICA
  {
    HICAACRO_TYPE     ACRO;   /* The acronym "HICA"                  */
    HICALENG_TYPE     LENG;   /* Length of this structure            */
    HICARETC_TYPE     RETC;   /* Transaction return code             */
    HICAREAS_TYPE     REAS;   /* Transaction reason code             */
    HICAENVP_TYPE     ENVP;   /* Transaction environment anchor      */
    HICACTLP_TYPE     CTLP;   /* Control   PDB anchor                */
    HICAINPP_TYPE     INPP;   /* Input     PDB anchor                */
    HICAOUTP_TYPE     OUTP;   /* Output    PDB anchor                */
    HICAMSGP_TYPE     MSGP;   /* Message   PDB anchor                */
    HICAERRP_TYPE     ERRP;   /* Error     PDB anchor                */
    HICASTPA_TYPE     STPA;   /* Reserved                            */
    HICACRRC_TYPE     CRRC;   /* Reserved                            */
    HICARESV_TYPE     RESV;   /* Reserved                            */
  }
         HICA,                /* Type - Structure.                   */
       *PHICA;                /* Type - Pointer to Structure.        */
#pragma pack()
/*-------------------------------------------------------------------*/
/* Language Binding type definitions.                                */
/*-------------------------------------------------------------------*/
typedef   unsigned long       QUERYTYPE_TYPE;
typedef   unsigned long       TRANSTATUS_TYPE;
typedef   unsigned long       TRANTYPE_TYPE;
typedef   unsigned long       IDBRC_TYPE;


/*-------------------------------------------------------------------*/
/* Language Binding return code constants.                           */
/*-------------------------------------------------------------------*/

#define   IDBRC_NOERR                 0
```

```
#define   IDBRC_XERR                1
#define   IDBRC_BADHICA             2
#define   IDBRC_BADPARM             3
#define   IDBRC_SYSERROR            4


/*-------------------------------------------------------------------*/
/* Prototype and constants for "IDBTransactionStatus".               */
/*-------------------------------------------------------------------*/

/* Status-type-check constants.                                      */
#define   IDB_CHECKFORCOMPLETION    0
#define   IDB_WAITFORCOMPLETION     1

/* Transaction status constants.                                     */
#define   IDB_TCOMPLETE             0
#define   IDB_TINPROGRESS           1


IDBRC_TYPE  _System IDBTransactionStatus( PHICA,
                                          QUERYTYPE_TYPE,
                                          TRANSTATUS_TYPE * );


/*-------------------------------------------------------------------*/
/* Prototype and constants for "IDBTransactionSubmit".               */
/*-------------------------------------------------------------------*/

/* Transaction type constants.                                       */
#define   IDB_SYNC           1       /* Synchronous  transaction.    */
#define   IDB_ASYNC          2       /* Asynchronous transaction.    */

IDBRC_TYPE  _System IDBTransactionSubmit( PHICA,
                                          TRANTYPE_TYPE );
#ifdef __cplusplus
}
#endif
#endif  /* #ifndef __idbh_h */
```

14

REXX HLAPI/2 Interface

The REXX HLAPI/2 interface enables you to access HLAPI/2 transactions from OS/2 REXX programs. It supports the same set of functional transactions as the HLAPI/2 interface, which is a subset of the transactions available via the HLAPI interface. For a list of the transactions, refer to "Transaction List" on page 135. The REXX HLAPI/2 interface is similar to HLAPI/REXX on MVS, so you should be familiar with HLAPI/REXX described in the *Tivoli Information Management for z/OS Application Program Interface Guide* before you use REXX HLAPI/2.

To utilize the HLAPI/2 transactions in an OS/2 REXX program, the REXX HLAPI/2 provides **BLMYRXM**, the REXX HLAPI/2 DLL. The **BLMYRXM DLL** is the interface between the REXX application program and the HLAPI/2. Users write OS/2 REXX applications which call the REXX HLAPI/2 DLL to submit their REXX HLAPI/2 transactions. **BLMYRXM** in turn invokes the HLAPI/2 client API to process the requested transactions on MVS.

The use of shared REXX variables for specifying control and input data to Tivoli Information Management for z/OS and returning output data from Tivoli Information Management for z/OS is equivalent to the HLAPI/REXX interface on MVS. Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* for information on:

- Using the HLAPI/REXX Interface
- REXX Reserved Variables
- Examples of REXX programs

The rest of this section is devoted to information unique to the REXX HLAPI/2.

REXX HLAPI/2 Installation and Setup

The REXX HLAPI/2 is installed as a component of the HLAPI/2. See "Installing and Configuring HLAPI/2" on page 77 for additional information on HLAPI/2 installation.

Once the installation procedure is completed, the following REXX HLAPI/2 parts are resident on your OS/2 system in the default installation subdirectory **\INFOAPI** or in the directory where you installed HLAPI/2:

- **BLMYRXM.DLL** - the REXX HLAPI/2 DLL in the **\DLL** subdirectory
- **BLMYRXSA.CMD** - sample REXX code in your **\SAMPLE\REXX** subdirectory

Prerequisite Setup

The REXX HLAPI/2 requires OS/2 Procedures Language 2/REXX (commonly referred to as REXX) which is included with OS/2 V2.11 or later.

Before the REXX HLAPI/2 is executed, the Tivoli Information Management for z/OS HLAPI/2 requester must be running. Refer to the section "The HLAPI/2 Requester" on page 105 for this information.

Registering the REXX HLAPI/2 DLL

To call the REXX HLAPI/2, the REXX HLAPI/2 DLL must first be registered with the function **RxFuncAdd**. Registering **BLMYRXM** makes its location known to REXX programs on your OS/2 system.

The syntax of RxFuncAdd is:

```
CALL RxFuncAdd 'blmyrxm','blmyrxm','blmyrxm'
```

blmyrxm
The first parameter is the name your REXX program will use to call the function.

blmyrxm
The second parameter is the name of the file containing the function.

blmyrxm
The third parameter is the name of the routine in the file that contains the function.

The RxFuncAdd call may be placed in individual REXX programs or as a separate REXX procedure which is called once by **STARTUP.CMD**. The latter technique saves the programmer the task of adding registration code within their REXX applications.

REXX HLAPI/2 Interface Calls

After registering the REXX HLAPI/2 interface, it is now callable from within an OS/2 REXX program. The parameter syntax for making interface calls is identical to the HLAPI/REXX on MVS, that is the transaction name, control, input and output parameters are the same.

Syntax of REXX HLAPI/2 call:

```
CALL BLMYRXM transaction-name,{control},{input},{output}
```

transaction-name
Specifies the transaction to perform. For a list of the transaction names supported by REXX HLAPI/2, see "Transaction List" on page 135.

control
An optional item. It is the stem of a compound variable that identifies control data items that this transaction uses.

input An optional item. It is the stem of a compound variable that identifies input data to use when processing the transaction.

output
An optional item. It is the stem of a compound variable for the REXX HLAPI/2 interface to use to return output data.

Deregistering the REXX HLAPI/2 DLL

Deregistering the REXX HLAPI/2 DLL causes REXX programs to lose access to the REXX HLAPI/2 functions. To drop the REXX HLAPI/2 DLL, the RxFuncDrop function is used. Although the sample REXX code shows how to deregister the REXX HLAPI/2 DLL, it is recommended that REXX applications not drop the REXX HLAPI/2 DLL because other REXX programs may currently be using the REXX HLAPI/2 on the same OS/2 system. This is also a consideration if you are running your REXX program repeatedly because the REXX HLAPI/2 DLL must be registered again once it is dropped.

The syntax of **RxFuncDrop** is:

```
CALL RxFuncDrop 'blmyrxm'
```

blmyrxm
Required parameter specifying the name of the function to be dropped.

Differences between the REXX HLAPI/2 and the HLAPI/2

Although the REXX HLAPI/2 uses the HLAPI/2, there is a difference between the two in how they process transactions.

- Only synchronous processing is supported for REXX HLAPI/2 transactions

  All transactions submitted are processed synchronously by the REXX HLAPI/2. Synchronous processing forces the REXX application's current thread to wait for a Tivoli Information Management for z/OS transaction to finish before it can perform any other work.

  The asynchronous processing feature of the HLAPI/2 is not supported.

Differences between the REXX HLAPI/2 and the HLAPI/REXX

The differences between the HLAPI/2 and HLAPI transactions also apply to REXX HLAPI/2. For a full description, see "Differences between HLAPI/2 and HLAPI Transactions" on page 111. In addition, the following differences affect the REXX HLAPI/2 interface:

- The REXX HLAPI/2 does not support MVS text data sets. As a result,
  - For the **RETRIEVE** transaction, the REXX variable **TEXT_MEDIUM** supports only a value of type B.
  - For the **CREATE** and **UPDATE** transactions, the **TEXT-NAME.?WIDTH** field, if specified, should always have a nonzero value indicating that a text data set is not used.

REXX Reserved Variables

The *Tivoli Information Management for z/OS Application Program Interface Guide* contains a list of REXX reserved variables used by HLAPI/REXX. In addition, the REXX HLAPI/2 uses the following REXX reserved variables:

BLG_HLAPI2_RC	Return code passed back from the HLAPI/2 interface. Null if the HLAPI/2 is not called.
BLG_REXXVAR_POOL_RC	Set to return code from the REXX variable pool service on failures.

RESULT	Return code from the REXX HLAPI/2. Used instead of RC.

Note: The following REXX reserved variables are not used by the REXX HLAPI/2:
- BLG_R15
- BLG_IRXEXCOM_RC
- RC

Sample REXX HLAPI/2 Program

The **SAMPLE\REXX** subdirectory in the directory where HLAPI/2 is installed, contains a sample REXX program named **BLMYRXSA.CMD**. This program illustrates how to register and deregister the REXX HLAPI/2 DLL, setup REXX variables, make REXX HLAPI/2 transaction calls, and retrieve output data.

BLMYRXSA performs the following functions.

1. Registers the REXX HLAPI/2 DLL by calling the function **RxFuncAdd**.
2. Initializes Tivoli Information Management for z/OS by performing an **INIT** transaction.
3. Records REXX HLAPI/2 output to an OS/2 file named **BLMYRXSA.OUT**. This is done for each transaction.
4. Creates a record with id **SAMP1**.
5. Retrieves the record **SAMP1** and records the fields just retrieved.
6. Deletes the record **SAMP1**.
7. Terminates Tivoli Information Management for z/OS by performing a **TERM** transaction.
8. Deregisters the REXX HLAPI/2 DLL by calling the function **RxFuncDrop**.

Steps Required to Run the REXX HLAPI/2 Sample Program

1. Perform the steps described in "Installation and Setup Summary for HLAPI/2 Sample Applications" on page 118.
2. Modify the sample REXX program **BLMYRXSA.CMD**. You may want to back up **BLMYRXSA** before making any changes to it. Change the following REXX variables used on the **INIT** transaction.
   - Change the value of the **SESSION_MEMBER** to the name of the session parameters member you want to use. If you want to use **BLGSES00**, no changes are required.
   - Change the value of the **PRIVILEGE_CLASS** to the name of the privilege class to be used on the **INIT** transaction. This privilege class must be in the database defined in your chosen session parameters member. The privilege class must have authority to display, create, and delete problem records.

     If you want to use **MASTER** privilege class, no changes are required.
   - Change the value of the **APPLICATION_ID** to the name of an eligible user in the privilege class you use.
   - Change the values of the **SECURITY_ID** and **PASSWORD** for the ID you want to use on the MVS system.

- If you choose to use another database profile other then **DATABASE.PRO**, change the value of the **DATABASE_PROFILE** to that name.

3. Start the HLAPI/2 requester. See “The HLAPI/2 Requester” on page 105 for additional information.
4. If you are using a RES or an MRES with APPC, be sure that Communications Manager/2 is running. If you are using an MRES with TCP/IP, be sure that TCP/IP is running.
5. From a command prompt, enter **BLMYRXSA** to run the sample program.

Transaction List

HLAPI Function	Transaction Number	REXX HLAPI/2 Transaction Name
Initialize Tivoli Information Management for z/OS	HL01	INIT
Terminate Tivoli Information Management for z/OS	HL02	TERM
Obtain External Record ID	HL03	GETID
Check Out Record	HL04	CHECKOUT
Check In Record	HL05	CHECKIN
Retrieve Record	HL06	RETRIEVE
(Reserved)	HL07	(Reserved)
Create Record	HL08	CREATE
Update Record	HL09	UPDATE
Change Record Approval	HL10	CHANGE_APPROVAL
Record Inquiry	HL11	SEARCH
Add Record Relations	HL12	ADD_REL
Delete Record	HL13	DELETE
Start User TSP or TSX	HL14	USERTSP
Get Data Model	HL31	GETDATAMODEL

15

Introduction to the HLAPI/NT

Tivoli Information Management for z/OS supports remote access from a workstation that runs in a Windows NT environment. It does this through the High-Level Application Program Interface (HLAPI) and the Tivoli Information Management for z/OS HLAPI Client for Windows NT (HLAPI/NT). The HLAPI/NT provides remote access to Tivoli Information Management for z/OS data and data manipulation services. It consists of three parts:

- A Tivoli Information Management for z/OS server, an MVS-based transaction program that resides on the MVS host system. It provides the link between Tivoli Information Management for z/OS and the Windows NT system.
- The Tivoli Information Management for z/OS HLAPI/NT Requester (requester), a Windows NT-based transaction program that provides workstation access to the HLAPI through a Tivoli Information Management for z/OS server.
- Language bindings and a support Dynamic Link Library (DLL) for the C language.

Two versions of the HLAPI/NT were available for installation, and you must know which was installed at your location. The file **IDBREQ.EXE** was installed if you elected to install the version that supports the TCP/IP protocol only; the file **IDBREQB.EXE** was installed if you elected to install the version that supports both the TCP/IP and the APPC protocols.

Like the HLAPI, the HLAPI/NT is a transaction-based application programming interface. User application programs interact with Tivoli Information Management for z/OS from the remote environment in basically the same way as they do from MVS using the HLAPI. These remote environment user application programs can be thought of as the *clients* to Tivoli Information Management for z/OS's *server*. The remote environment offers a subset of HLAPI transactions, which are listed in Table 1 on page 3, and described in the *Tivoli Information Management for z/OS Application Program Interface Guide*.

The HLAPI/NT enables application programmers to write applications for use in their specific work environment. The task described in “A Typical Scenario” is typical of the system problems that can be solved by using Tivoli Information Management for z/OS database services.

A Typical Scenario

Suppose an application programming group in an enterprise has written two workstation-based help desk applications that interact with the HLAPI through the HLAPI/NT. One is a problem management database application, and the other is a configuration management database application. The application programming group has already provided the help desk with the information necessary to install and start these applications successfully.

For efficiency, the help desk operator maintains two separate user IDs on the MVS system: one with basic privilege class authority for queries sent through the configuration management application, and one with a higher privilege class authority for creating records through the problem management application.

1. A help desk operator starts a Windows NT workstation. The workstation's **STARTUP** group starts up the HLAPI/NT requester, the problem management application, and the configuration management application.
2. When a problem call arrives, a help desk operator uses the problem management application to collect preliminary information and open a problem record through the HLAPI/NT.
3. In another window, the same operator uses the configuration application to query Tivoli Information Management for z/OS through the HLAPI/NT for information about the caller's configuration.
4. Meanwhile, Tivoli Information Management for z/OS has returned a problem number back through the HLAPI/NT, and the operator gives the caller his problem number and promises follow-up on the problem.
5. By this time, Tivoli Information Management for z/OS has returned results of the configuration query through the HLAPI/NT. The operator can then research the problem and update the problem record as necessary.

The same Tivoli Information Management for z/OS functions that once required direct host access are now performed on a desktop workstation. The remaining sections of this chapter help you understand the interactions of the HLAPI/NT and Tivoli Information Management for z/OS.

Server Overview

A Tivoli Information Management for z/OS server is an MVS/ESA transaction program that handles communication between a HLAPI/NT requester and any Tivoli Information Management for z/OS databases that reside on the MVS system where the server is installed. A Windows NT client application program must use a requester for access to the server.

A server must be installed and available on every MVS/ESA machine with a Tivoli Information Management for z/OS database that an application using the HLAPI/NT needs to access.

The HLAPI/NT can use any of the Tivoli Information Management for z/OS servers. See "Configuring and Running a Remote Environment Server (RES)" on page 25, "Configuring and Running a Multiclient Remote Environment Server (MRES) with APPC" on page 35, and "Configuring and Running a Multiclient Remote Environment Server (MRES) with TCP/IP" on page 53 for information about the servers.

Requester Overview

The requester is a transaction program that runs on the workstation. It must be up and running before any HLAPI/NT activity can occur. At the request of a client application program, the requester initiates a conversation with a server. Then the requester transfers information from the client application program to the appropriate server, and from the server back to the client application program.

To start the requester, open the Tivoli Information Management for z/OS folder and click on the HLAPI FOR WINDOWS NT REQUESTER icon. You can also start the requester from the workstation's **STARTUP** group. See "The HLAPI/NT Requester" on page 163 for more information about starting the requester.

The HLAPI/NT requester is implemented as a Windows NT system service. Therefore, HLAPI/NT requester services are available to all processes and related threads running in the Windows NT environment. One copy of the HLAPI/NT requester service can support many user applications.

HLAPI/NT C Language Binding

A client application program communicates with the Tivoli Information Management for z/OS system by creating a high-level application communication area (HICA) and its related parameter data blocks (PDBs). The client application program then submits the HICA transaction by making HLAPI/NT program service calls. The HLAPI/NT program service routines exist on the user's workstation as a dynamic link library (DLL). When the calls are made by the client application program, the supporting HLAPI/NT routines are loaded from the DLL and started.

To utilize HLAPI/NT program service calls, the HLAPI/NT provides two standard header files and one import library. These are located in the H subdirectory of the directory in which you install HLAPI/NT. These files are only used in the creation of an HLAPI/NT application. After the application is written and installed, these files are not required to be on a user's workstation.

IDBH.H is a required C programming language header file for all HLAPI/NT application programs. It defines data types and function calls used by HLAPI/NT to communicate with your application program.

IDBHLAPI.LIB is the import library that contains the function calls provided by HLAPI/NT. You must specify this library when you link your compiled program.

IDBECH.H is an optional C programming language header file that defines constant declarations for return and reason codes used by the HLAPI/NT.

Basic Transaction Flow

A *transaction sequence* is a series of HLAPI/NT transactions that begins with an initialize Tivoli Information Management for z/OS (HL01) transaction, followed by other supported transactions in any order, and ends with a terminate Tivoli Information Management for z/OS (HL02) transaction. Client application programs submit transactions in a transaction sequence, which is referred to as a *logical session*.

Each HLAPI/NT transaction request travels from a client application program on Windows NT to Tivoli Information Management for z/OS on MVS along the route shown in Figure 11 on page 140. This example illustrates HLAPI/NT using an MRES with TCP/IP. The path would be similar using a RES or an MRES with APPC, but the communication protocol would be APPC instead of TCP/IP.

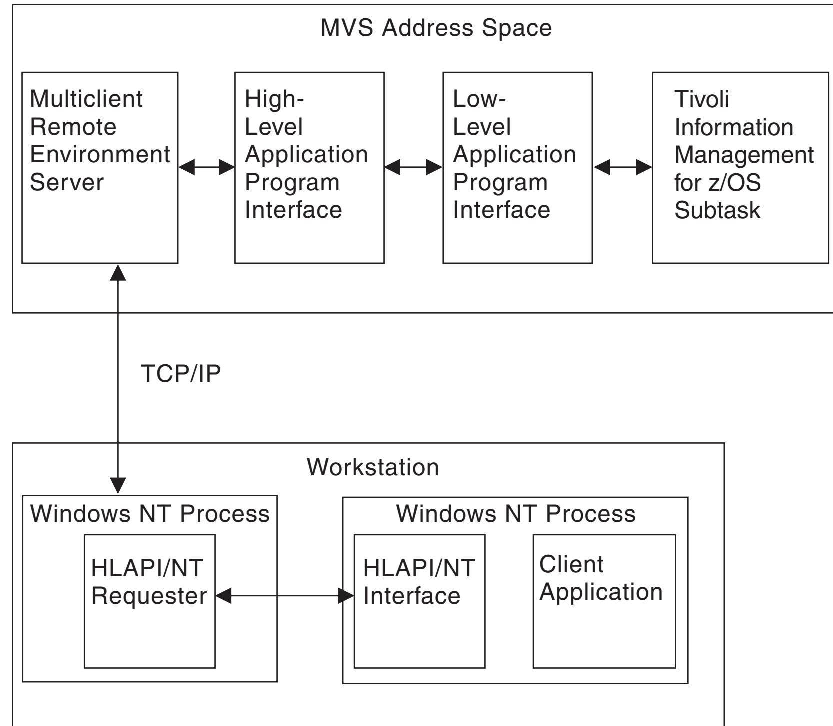


Figure 11. HLAPI/NT Overview

On the Workstation

The client application program, through the HLAPI/NT program service calls, uses the requester to request Tivoli Information Management for z/OS work to be performed. The client application creates a HICA and PDBs to represent a Tivoli Information Management for z/OS HLAPI transaction. Then it submits the HICA to HLAPI/NT for processing. HLAPI/NT takes the HICA and its related control and input PDBs and translates them for the server character set. It then packages and transfers them to the server associated with the specified HICA. (Refer to transaction HL01 in the *Tivoli Information Management for z/OS Application Program Interface Guide*, for a description of how to associate a HICA with a particular Tivoli Information Management for z/OS database.)

Communication Link

The HLAPI/NT requester communicates with a server using TCP/IP or the APPC LU 6.2 protocol. The client application chooses the communication protocol and that protocol is used for the entire transaction sequence submitted by that client application.

The requester can communicate with multiple servers on multiple MVS hosts. The **IDBSERVERHOST** database profile keyword indicates that the client wants a TCP/IP connection. The **IDBSYMDESTNAME** database profile keyword indicates that the requester is to establish an APPC conversation on behalf of the client. These keywords and the database profile are described in "HLAPI/NT Profiles, Environment Variables, and Data Logging" on page 153.

To enable the communication link between a Windows NT workstation and an MRES with TCP/IP, TCP/IP must be set up for HLAPI/NT use. For information on configuring an MRES with TCP/IP, see “Configuring and Running a Multiclient Remote Environment Server (MRES) with TCP/IP” on page 53.

To enable the communication link between a Windows NT workstation and a RES or an MRES with APPC, APPC client software, such as the client software provided with IBM Communications Server for Windows NT or Microsoft SNA Server, must be installed and configured on the workstation where the client application is running. For information on configuring a RES or MRES with APPC, refer to “Configuring and Running a Remote Environment Server (RES)” on page 25 and “Configuring and Running a Multiclient Remote Environment Server (MRES) with APPC” on page 35.

On the Host

Upon arrival in the server, the HICA and PDBs are processed and submitted to the Tivoli Information Management for z/OS HLAPI. After the requested HLAPI transaction finishes, the server transmits the HICA, the output, error, and message PDBs, and the **PDBCODE** field of the input PDB to the requester.

Back to the Workstation

The transmission buffers are received by the requester. The buffers are parsed in sequence and their contents (the PDBs) are translated from the server character set to the character set being used by the workstation. Memory is allocated for the newly received PDBs. These PDBs are chained onto their corresponding type list on the owning (and original) HICA. The HICA contains the original control and input PDB chains and the new output, error, and message PDB chains. The HICA contains other fields set by the HLAPI, such as **HICARETC** and **HICAREAS**. The value set by the HLAPI in the input PDB field is also returned. The transaction request is complete, and control of the HICA is returned to the client application program.

Client Workstation Requirements for the HLAPI/NT

The HLAPI/NT requires certain software and hardware in order to function.

Note: For the HLAPI/NT, the requester and client interface must be run on the same machine.

Software

- Microsoft Windows NT Version 4.0. This version includes support for TCP/IP.
- If you install the version of HLAPI/NT that supports both TCP/IP and APPC, you must install and configure the APPC client software, such as the client software provided with IBM Communications Server for Windows NT Version 5.0 or Microsoft SNA Server Version 2.11 or higher.
- IBM VisualAge for C++ for Windows or Microsoft Visual C++ or any C language compiler and linker that supports the 32-bit system linkage convention.

  Note: A compiler is required only if you are developing client applications. A client workstation does not need a compiler to run a client application.
- To use the optional HLAPI for Java provided with the client, you must have Java Development Kit (JDK) Version 1.1 or higher, and Microsoft Windows NT Workstation Version 4.

Hardware

- An IBM personal computer or compatible system unit capable of running Windows NT Workstation Version 4.0.
- One or more fixed disk drives with sufficient capacity to contain your version of Windows NT and the disk storage requirements as specified in "Disk Storage".
- Token-Ring Adapter Card and network or a communication option capable of supporting TCP/IP or APPC LU 6.2 communication to one or more MVS systems running a Tivoli Information Management for z/OS server.

Disk Storage

The disk space needed for each component is:

- Installation utility, 520 KB
- Requester, 1.5 MB
- Toolkit, 16.3 MB

1 MB equals 1 048 576 bytes; 1 KB equals 1024 bytes.

16

Installing and Configuring HLAPI/NT

To use HLAPI/NT on your workstation, you must do the following tasks:

1. Configure a communication link to a Tivoli Information Management for z/OS server. "Configuring a Communication Link to a Server" discusses this task.
2. Install the HLAPI/NT files on your workstation.

Configuring a Communication Link to a Server

In order to use the functions of the HLAPI/NT, you must configure a communication link for the workstation to each server you want to use. If you are using an MRES with TCP/IP, you must configure a TCP/IP communication link. If the server you are using is a RES or MRES with APPC, you must configure an APPC/APPN communication link. The following sections describe how to configure your communications software and update various files. "Configuring HLAPI/NT for TCP/IP" provides the steps needed for a TCP/IP configuration. "Configuring HLAPI/NT for APPC" on page 144 provides the steps needed for an APPC configuration.

Configuring HLAPI/NT for TCP/IP

To use an MRES with TCP/IP, you must configure TCP/IP so that HLAPI/NT can connect to each server you want to use. Each MRES with TCP/IP is uniquely identified by the IP address of its MVS host and its port number. Refer to "Configuring and Running a Multiclient Remote Environment Server (MRES) with TCP/IP" on page 53 for more information about the MRES with TCP/IP.

To identify the port number of each MRES with TCP/IP, you must update the **SERVICES** file in your **ETC** subdirectory to associate a service name or alias with the port number defined in the MRES with TCP/IP. You must specify a service name and port number for each server the HLAPI/NT needs to be able to connect to. The port numbers must match those used by the Tivoli Information Management for z/OS MRES with TCP/IP servers. The service name is of your choosing. Service names are case-sensitive.

A default service name `infoman` and port number `1451` have been reserved for Tivoli Information Management for z/OS use. The general format of an entry in the SERVICES file is:

```
<service>     <port>/tcp   <alias_list>       #<comment>
```

<service>
: The service name of the Tivoli Information Management for z/OS MRES with TCP/IP.

<port>
: The port number of the Tivoli Information Management for z/OS MRES with TCP/IP.

<alias_list>
: Alias definitions for the service

<comment>
: Comment text that describes the service.

For example, to associate the default service name (`infoman`) with the default port number (`1451`), you must place the following line in the **SERVICES** file before you run HLAPI/NT:

```
infoman  1451/tcp      #default MRES server
```

The default service name and default port number are reserved for Tivoli Information Management for z/OS so you can use them to designate your MRES with TCP/IP. If the client application does not specify a service name in the database profile specified on the HL01 transaction, `infoman` will be assumed. Therefore, be sure to include it in the **SERVICES** file.

Be sure that your client application programs use the service names that you define. If your client application program needs to access an MRES with TCP/IP that uses a port number other than the default `infoman/1451`, you must specify its service name in the **IDBSERVERSERVICE** keyword in the client application's database profile.

Additional information regarding the **IDBSERVERSERVICE** keyword and the database profile is located in "HLAPI/NT Profiles, Environment Variables, and Data Logging" on page 153.

To identify the MVS host running the MRES with TCP/IP, your client application must specify the host in the **IDBSERVERHOST** keyword in the database profile. The host can be identified by an IP address in dotted-decimal format or by a host name. If you use a host name, the host name must be resolvable. Refer to the Windows NT system documentation for information on how TCP/IP should be configured to resolve a host name.

Additional information regarding the **IDBSERVERHOST** keyword and the database profile is located in "HLAPI/NT Profiles, Environment Variables, and Data Logging" on page 153.

Configuring HLAPI/NT for APPC

To use a RES or MRES with APPC, you must configure an APPC/APPN communication link for the workstation to each server you want to use. The tasks you need to do to configure APPC/APPN are:

1. Determine the values to use in the configuration. See "Determining Values Clients Need" on page 50 for information on possible values.
2. Define a local LU.
3. Define a logical link to the MVS host where the server resides.
4. Define a partner LU.
5. Define the symbolic destination name for a CPI-C side information entry.

For detailed information about performing these tasks, consult the appropriate documentation for your APPC client software. For the Microsoft SNA Server, refer to the *Administration*

Guide for information about assigning LUs to users and configuring symbolic destination names. For the IBM Communications Server, refer to *Quick Beginnings*.

Preparing to Install HLAPI/NT

HLAPI/NT is delivered on a CD-ROM. You can install HLAPI/NT directly from the CD-ROM to a workstation or install onto a network drive and then to workstations from the network drive.

Before you begin installing HLAPI/NT, you should familiarize yourself with the important information in the **README.TXT** file shipped with it. If you are installing HLAPI/NT from the HLAPI/NT installation CD-ROM, the **README.TXT** file is on the CD-ROM; if you are installing HLAPI/NT from a network drive, the **README.TXT** file is in the network drive subdirectory.

If you already have a version of HLAPI/NT installed, delete it before installing the new version of HLAPI/NT. See “Deleting HLAPI/NT from a Workstation” on page 151.

When installing HLAPI/NT on your workstation, you install a folder containing icons for the HLAPI/NT Requester, the **README.TXT** file, and the **UNINSTALL** utility.

Note: Upgrades or patches that can be downloaded from a Tivoli Web site may be available for HLAPI/NT. Visit the Tivoli Information Management for z/OS Web site for more information:

http://www.tivoli.com/infoman

Installing HLAPI/NT in Attended Mode from CD-ROM

Install HLAPI/NT by following these steps (during the installation, you can press **F1** on any window to display Help information):

1. Switch to, or start, an MS DOS window.
2. If you already have HLAPI/NT installed, delete it. See “Deleting HLAPI/NT from a Workstation” on page 151.
3. Insert the HLAPI/NT installation CD-ROM into the CD-ROM drive.
4. At the DOS command prompt, type `e:\hlapi\nt\disk1\setup` and press Enter (`e` is the drive letter of the drive that contains the HLAPI/NT installation CD-ROM).
5. Select **Next >** from the Welcome window.
6. In the Setup Options window, select **Workstation Installation**.
7. In the Select Components window:
   - Select the components of HLAPI/NT you want to install. If you select the Requester component, you must choose either the requester component which supports only TCP/IP (**IDBREQ.EXE**) or the requester component which supports both TCP/IP and APPC (**IDBREQB.EXE**). “Components of HLAPI/NT” on page 342 contains a complete list of HLAPI/NT files.
   - Select **Browse...** to change the destination directory in which to install the HLAPI/NT files. You can accept the default value or change it.

Note: You can select **Disk space...** to determine the amount of available space on the fixed disk drives in your workstation.

- Select **Next >**.

8. In the Select Program Folder window:
   - Select the folder into which you want the HLAPI/NT icons placed. You can choose to have a new folder created or select an existing folder.
   - Select **Next >**.
9. In the Start Copying Files window, verify that the information is correct and then select **Next >**.
10. In the Setup Complete window:
    - Select whether you want your computer to be restarted now.
    - Select **Finish**.

The HLAPI/NT installation is finished. After you start your workstation again, verify the installation.

Installing HLAPI/NT onto a Network Drive

To install HLAPI/NT on a workstation from a network drive, you must first install HLAPI/NT onto the network drive. You do this by choosing the Network Drive installation option during the installation.

To install HLAPI/NT, follow these steps:

1. Switch to, or start, an MS DOS window.
2. If you already have HLAPI/NT installed, delete it. See “Deleting HLAPI/NT from a Workstation” on page 151.
3. Insert the HLAPI/NT installation CD-ROM into a drive.
4. At the DOS command prompt, type `e:\hlapi\nt\disk1\setup` and press Enter (`e` is the drive letter of the CD-ROM drive that contains the HLAPI/NT installation CD-ROM).
5. Select **Next >** from the Welcome window.
6. In the Setup Options window, select **Network Drive** installation.
7. In the Select Components window:
   - Select the components of HLAPI/NT you want to install. (See “Components of HLAPI/NT” on page 342 for a complete list of HLAPI/NT files.)
   - Select **Browse...** to change the destination directory in which to install the HLAPI/NT files. You can accept the default value or change it.

     Note: You can select **Disk space...** to determine the amount of available space on the fixed disk drives in your workstation.
   - Select **Next >**.
8. In the Select Program Folder window:

- Select the folder into which you want the HLAPI/NT icons placed. You can choose to have a new folder created or select an existing folder.
- Select **Next >**.

  Note: The HLAPI/NT Requester icon will not be created because you cannot run the product from this installation.

9. In the Start Copying Files window, verify that the information is correct and then select **Next >**.

   The HLAPI/NT files are transferred from the CD-ROM to the network drive.

10. In the Setup Complete window, select **Finish**.

The HLAPI/NT installation to the network drive is complete. Give all authorized users access to the network drive on which you installed HLAPI/NT.

Installing HLAPI/NT in Attended Mode from a Network Drive

If you have access to a network drive on which HLAPI/NT is installed, you can access the drive/directory and install HLAPI/NT on your workstation by following these steps:

1. Switch to, or start, an MS DOS window.
2. Use the CHDIR (CD) command to change to the DISK1 subdirectory of the network drive directory that contains HLAPI/NT.
3. If you already have HLAPI/NT installed, delete it. See "Deleting HLAPI/NT from a Workstation" on page 151.
4. Type **SETUP** on the command line.
5. Select **Next >** from the Welcome window.
6. In the Setup Options window, select **Workstation Installation**.
7. In the Select Components window:
   - Select the components of HLAPI/NT you want to install. If you select the Requester component, you must choose either the requester component which supports only TCP/IP (**IDBREQ.EXE**) or the requester component which supports both TCP/IP and APPC (**IDBREQB.EXE**). "Components of HLAPI/NT" on page 342 contains a complete list of HLAPI/NT files.
   - Select **Browse...** to change the destination directory in which to install the HLAPI/NT files. You can accept the default value or change it.

     Note: You can select **Disk space...** to determine the amount of available space on the fixed disk drives in your workstation.
   - Select **Next >**.
8. In the Select Program Folder window:
   - Select the folder into which you want the HLAPI/NT icons placed. You can choose to have a new folder created or select an existing folder.
   - Select **Next >**.
9. In the Start Copying Files window, verify that the information is correct and then select **Next >**.

The HLAPI/NT files are transferred from the network drive to your workstation.

10. In the Setup Complete window:
    - Select whether you want your computer to be restarted now.
    - Select **Finish**.

The HLAPI/NT installation is complete. After you start your workstation again, you can verify the installation.

Installing HLAPI/NT in Unattended Mode

A sample response file is shipped with HLAPI/NT. You can use this file as is, or you can modify it to meet your needs. The sample response file is named **SETUP.ISS** and it is contained on the HLAPI/NT installation CD-ROM or the network drive directory that contains the HLAPI/NT code.

You can use the sample response file to install HLAPI/NT from either the HLAPI/NT installation CD-ROM or from a network drive. Details about the response file keywords are contained in “HLAPI/NT Response File Keywords”.

To install HLAPI/NT in unattended mode, follow these steps:

1. Switch to, or start, an MS DOS window.
2. Use the CHDIR (CD) command to change to the appropriate subdirectory of the HLAPI/NT installation CD-ROM or the appropriate subdirectory of the network drive directory that contains HLAPI/NT.
3. If you already have HLAPI/NT installed, delete it. See “Deleting HLAPI/NT from a Workstation” on page 151.
4. Type **SETUP -s** [-*f1***path\ResponseFile**] [-*f2***path\LogFile**], where

`-s`	Runs InstallShield Silent to perform an unattended installation
[-*f1*path\ResponseFile]	Specifies the alternate location and name of the response file (**.ISS** file). The default path is the location of the **SETUP.EXE** file and the default name is **SETUP.ISS**.
[-*f2*path\LogFile]	Specifies the alternate location and name of the log file created by InstallShield Silent. The default path is the location of the **SETUP.EXE** file and the default name is **SETUP.LOG**.

HLAPI/NT Response File Keywords

The sample response file shipped with HLAPI/NT includes the following keywords. All keywords are required.

`[InstallShield Silent]`

Response File Silent Header.

Version

Indicates the version of the InstallShield response file. The current valid value for this keyword is:

v3.00.000

File Identifies the file as a legitimate InstallShield response file. The current valid value for this keyword is:

Response File

[Application]

Response File Application Header.

Name Indicates the Product Key from the Installation Information. The current valid value for this keyword is:

Service Desk for OS/390

Version
Indicates the Version Key from the Installation Information. The current valid value for this keyword is:

1.2

Company
Indicates the Company Key from the Installation Information. The current valid value for this keyword is:

Tivoli

[DlgOrder]

Response File Dialog Sequence Header. This section lists all dialogs used in the installation in the order in which they appear. In addition, the exact number of dialogs listed is specified in this section. This section is very installation dependent and should not be modified.

Dlg0=SdBitmap-0

Count=6

Dlg1=SdOptionsButtons-0

Dlg2=SdComponentDialogAdv-0

Dlg3=SdSelectFolder-0

Dlg4=SdStartCopy-0

Dlg5=SdFinishReboot-0

[SdBitmap-0]

Response File Dialog Data for the initial Welcome window.

Result Button that was clicked to exit or end the dialog. Valid values for this keyword are:

12 Back button
1 Next button

[SdOptionsButtons-0]

Response File Dialog Data for the Setup Options Dialog window.

Result Button that was clicked to exit or end the dialog. Valid values for this keyword are:

12 Back button
101 Workstation Installation
102 Network Drive Installation

`[SdComponentDialogAdv-0]`

Response File Dialog Data for the Select Components window.

szDir Destination directory path. Default path is *C:\INFOAPI*.

Component-type
The only value currently allowed is *String*.

Component-count
The total number of components selected to be installed. For a workstation installation, this number can be 1 or 2. For a network drive installation, this number can be 1 or 2 or 3. You must specify

- The COMPONENT-COUNT keyword which specifies the number of components which will be installed.
- A list of components to be installed (the number of components specified must equal the number specified in the **COMPONENT-COUNT** keyword). The component names must be specified as **COMPONENT-N=NAME** where **N** is a number, beginning with 0, that numbers each component to be installed, and **NAME** is the name of the component. Because numbering begins with 0, the greatest value for **N** will always be one less than the value of **COMPONENT-COUNT**. The valid values for **NAME** are:
  Toolkit
  Requester - TCP-IP only
  Requester - TCP-IP and APPC

  Note: For the purposes of specifying these names, note that **TCP-IP** is specified as **TCP-IP** and not **TCP/IP**.

For example, to install the Toolkit component and the Requester component that supports the TCP/IP protocol only, you would specify a component count of **2** and then list the 2 components:
Component-Count=2
Component-0=Toolkit
Component-1=Requester - TCP-IP only

Result Button that was clicked to exit or end the dialog. Valid values for this keyword are:

12 Back button
1 Next button

`[SdSelectFolder-0]`

Response File Dialog Data for the Select Program Folder window.

szFolder
The name of the program folder where you want the icons for the product placed. The default program folder name is *Tivoli Service Desk for OS390*.

Note: The folder name cannot contain a slash / character.

Result Button that was clicked to exit or end the dialog. Valid values for this keyword are:

12 Back button
1 Next button

[SdStartCopy-0]

Response File Dialog Data for the Start Copying Files window.

Result Button that was clicked to exit or end the dialog. Valid values for this keyword are:

12 Back button
1 Next button

[SdFinishReboot-0]

Response File Dialog Data for the Setup Complete window.

BootOption
Boot option used upon completion of the installation. Valid values for this keyword are:

0 Do not restart Windows NT or the machine.
2 Restart Windows NT
3 Reboot the machine

Result Button that was clicked to exit or end the dialog. The only valid value for this keyword is:

1 Finish

Applying HLAPI/NT Maintenance

To apply maintenance to HLAPI/NT, you must delete and reinstall the components that are affected by the update or maintenance. See "Deleting HLAPI/NT from a Workstation" and "Installing HLAPI/NT in Attended Mode from CD-ROM" on page 145 or "Installing HLAPI/NT onto a Network Drive" on page 146.

Deleting HLAPI/NT from a Workstation

To delete HLAPI/NT from a workstation, follow these steps:

1. Open the folder where you installed the HLAPI/NT components.
2. Run the Uninstall Components program.
3. From the Select Components window:
   a. Select the components you wish to uninstall.
   b. Select **Remove**.
4. A message appears when the uninstall is complete. Select **OK**.

17

HLAPI/NT Profiles, Environment Variables, and Data Logging

Certain aspects of the HLAPI/NT interface can be tuned to the needs of your application. You do this by specifying profile keywords and values in two Windows NT text files—the system profile and the database profile.

The system profile is optional. It is specified in the command that starts the HLAPI/NT requester. See "The HLAPI/NT Requester" on page 163 for more information about this command. The system profile defines the sizes of the data buffers that are passed between the Windows NT workstation and the MVS host.

The database profile is required and must be specified in a control PDB that is passed to HLAPI/NT as part of an HL01 transaction. The database profile defines which MVS server is the destination of Tivoli Information Management for z/OS HLAPI transactions that are submitted by the user application. It controls all aspects of logging these transactions on the workstation, and it defines which ASCII and EBCDIC code sets to use for data conversion.

Profile Syntax

A profile can be created and manipulated with common text editors. You must press the Enter key at the end of each line. The profile syntax is keyword driven. Keyword processing is not case-sensitive. The keywords can be entered with any mix of uppercase and lowercase characters. Each keyword requires a data parameter.

For each keyword, the equals character (=) separates the keyword from its data value. Optionally, one or more spaces may precede or follow the equals character or the keyword. The data value consists of all nonblank characters to the right of the equals character, up to the end of line. An example follows:

```
IDBServCharCodeSet = IBM-037
```

You can specify a comment; but it must be specified on a text line of its own. The comment must be preceded by the characters REM as in the following example:

```
REM Code set IBM-037 is the EBCDIC U.S. English code page.
```

When you enter numbers for profile keyword data, *do not* use commas in the numbers; for example

```
IDBLogFileSize = 262144
```

System Profile Keywords

When the requester is started, it can be given a system profile file name. The system profile enables the application programmer to tune certain aspects of the client HLAPI/NT system.

The system profile file name can be fully qualified with its path and drive, or you can specify the file name only. If only a file name is specified, the search path name is obtained from the current value held in the **IDBSMPATH** environment variable described in “IDBSMPATH” on page 160.

If a profile is not specified, default values are used for each of the keywords. Additional detail about the keywords is contained on this page and on pages following.

For an example of a System Profile, see “System Profile Example” on page 155.

IDBINBOUNDBUFSIZE

This value is the number of bytes to be allocated for each buffer that the workstation receives from the server. The number is rounded up to the next highest multiple of 4096 in all cases except at the uppermost range (greater than 28672). A buffer size greater than this, but less than 32767, is rounded up to 32767.

Note: Do not use commas when entering numbers in profile keywords.

An example of when to increase the **IDBINBOUNDBUFSIZE** is when you have an application that uses retrieve transactions containing a large amount of data.

Valid entries: any number between 1 and 32767.

Default value: 4096.

IDBOUTBOUNDBUFSIZE

This value is the number of bytes to be allocated for each buffer sent from the workstation to the server. The number is rounded up to the next highest multiple of 4096 in all cases except at the uppermost range (greater than 28672). A buffer size greater than this, but less than 32767, is rounded up to 32767.

Note: Do not use commas when entering numbers in profile keywords.

An example of when to increase the **IDBOUTBOUNDBUFSIZE** is when you have an application that uses create transactions that contain a large amount of data.

Valid entries: any number between 1 and 32767.

Default value: 4096.

IDBSHARECMS

This keyword determines whether the requester should enable or disable conversation sharing. When conversation sharing is enabled, the requester assigns new client applications to an existing conversation if criteria such as same server and same security ID are met. When conversation sharing is disabled (the default), each client application is assigned its own dedicated conversation. A conversation is terminated when the last client assigned to it submits an HL02.

Note: If you choose to use conversation sharing, you must be aware that there is a potential for a delay because transactions are handled synchronously. Thus, if Client A and Client B share a conversation, and Client A submits a long search and Client B submits an update, Client B will wait for Client A's search to complete before its transaction can be processed.

Note: If you are using pre-started API sessions (described in "MRES with Pre-started API Sessions Considerations" on page 18), you must disable conversation sharing.

Valid entries: 0 (conversation sharing disabled) or 1 (conversation sharing enabled).

Default value: 0 (conversation sharing disabled).

System Profile Example

```
REM**********************************************************************
REM
REM                        Example  System Profile
REM
REM
REM**********************************************************************
REM
REM
IDBINBOUNDBUFSIZE = 4096
REM
IDBOUTBOUNDBUFSIZE = 4096
REM
IDBSHARECMS = 0
REM
```

Database Profile Keywords

Database profiles are created by the user. The database profile file name is passed to HLAPI/NT through a PDB named **DATABASE_PROFILE** on the Control PDB list. This occurs when your application connects to the database using the HL01 transaction. The database profile is used only with the HL01 transaction. If you specify it for other transactions, it is ignored.

The database profile file name can be fully qualified with its path and drive, or you can specify the file name only. If only a file name is specified, the search path name is obtained from the current value held in the environment variable described in "IDBDBPATH" on page 159.

During profile resolution, the contents of database profile text files and profile overrides are compiled together into a final collection of profile settings. This final collection is then used by the HLAPI/NT. For more information about profile overrides, see "Profile Override" on page 159.

Within a profile, a keyword cannot be duplicated. If a keyword is duplicated, an error is reported, and processing ends.

To learn about individual keywords, go to one of the following:

"IDBDataLogLevel" on page 156
"IDBLogFileSize" on page 156
"IDBLogFileNameActive" on page 156
"IDBLogFileNameOld" on page 156

"IDBCharCodeSet"
"IDBServCharCodeSet" on page 157
"IDBServerHost" on page 157
"IDBServerService" on page 157
"IDBSymDestName" on page 157

For an example of a Database Profile, see "Database Profile Example" on page 158.

IDBDataLogLevel

The level at which client data logging is performed. This profile setting can be overridden with the Windows NT environment variable also called **IDBDataLogLevel**.

Valid entries: *0* (logging disabled) and *1* (logging enabled).

Default value: *0* (logging disabled).

IDBLogFileSize

The approximate maximum size in bytes of a log file. The log file is phased after it grows larger than this size. To *phase* a log file is to close the current log file, rename and archive it, and open a new empty log file.

Valid entries: Any positive integer between 4096 and 10 485 760 (do not include commas when entering numbers). If a value between 1 and 4095 is specified, 4096 is substituted. Specifying zero causes the log file to grow indefinitely.

Default value: 262 144.

IDBLogFileNameActive

The primary name given to the active log file name for the client.

Valid entries: Any valid file name.

Default value: **IDB_LOG.ACT**. If you are running in a LAN environment, it is suggested that this file be written to a file unique to each LAN workstation to avoid errors due to file contention.

IDBLogFileNameOld

The name given to log files about to be archived. After the active log file as specified by **IDBLogFileNameActive** is phased, the file is renamed to this value.

Valid entries: Any valid file name.

Default value**: IDB_LOG.OLD**. If you are running in a LAN environment, it is suggested that this file be written to a file unique to each LAN workstation to avoid file contention errors.

IDBCharCodeSet

This keyword indicates the code set to use in the client. Character data bound for the client is translated to this code set. Character data bound for the server is translated from this code set.

Default value: IBM-850 (U.S. English).

IDBServCharCodeSet

This keyword indicates the code set that the server uses. Character data bound for the server is translated to this code set. Character data bound for the client is translated from this code set.

Default value: IBM-037 (U.S. English).

IDBServerHost

This keyword identifies the MVS host that is running the MRES with TCP/IP server you want the requester to establish a conversation with. For more information about the MRES with TCP/IP and setting up the HLAPI/NT to communicate with an MRES with TCP/IP, refer to “Configuring and Running a Multiclient Remote Environment Server (MRES) with TCP/IP” on page 53 and “Configuring HLAPI/NT for TCP/IP” on page 143.

Valid entries: Any valid IP address in dotted-decimal format, or any valid host name, such as mvshost. If you specify a host name, the host name must be resolvable. If you use this keyword, you must not use the **IDBSymDestName** keyword. Refer to the Windows NT online documentation for information on host name resolution.

Default value: none. If the **IDBSymDestName** keyword is not specified, this value is required.

IDBServerService

This keyword identifies the service name of the MRES with TCP/IP server you want the requester to establish a conversation with. The service name must be listed in the **SERVICES** file in the **ETC** subdirectory on your workstation. For more information about the MRES with TCP/IP and setting up the HLAPI/NT to communicate with an MRES with TCP/IP, refer to the “Configuring HLAPI/NT for TCP/IP” on page 143 and “Configuring and Running a Multiclient Remote Environment Server (MRES) with TCP/IP” on page 53.

Valid entries: Any valid service name or alias. Service names and aliases are *case-sensitive*.

Default value: **infoman**. **IDBServerService** is an optional keyword. If you do not specify it when you specify **IDBServerHost**, the default is assumed. If you specify **IDBSymDestName**, then **IDBServerService** is ignored.

IDBSymDestName

Symbolic destination name. This keyword specifies the name of a Common Programming Interface for Communications (CPI-C) side information entry, which provides information required for HLAPI/NT to establish a conversation with a RES or an MRES with APPC. For more information on CPI-C side information entries, refer to the your APPC client software documentation. You can also refer to “Configuring and Running a Remote Environment Server (RES)” on page 25, “Configuring and Running a Multiclient Remote Environment Server (MRES) with APPC” on page 35, and “Configuring HLAPI/NT for APPC” on page 144 for additional information.

If you use this keyword, you must not use the **IDBServerHost** keyword.

Valid entries: Any CPI-C symbolic destination name (a 1- to 8-byte character string) that you have defined.

Default value: none. This value is required if **IDBServerHost** is not specified.

Database Profile Example

```
REM*********************************************************************
REM
REM                          SAMPLE Database Profile
REM
REM*********************************************************************
REM
REM ---------------------------------------------------------------------
REM Server Host Name or TCP/IP address (Required if using TCP/IP)
REM ---------------------------------------------------------------------
IDBServerHost       = yourhost
REM
REM ---------------------------------------------------------------------
REM MRES Service Name (Optional)
REM ---------------------------------------------------------------------
IDBServerService    = infoman
REM
REM ---------------------------------------------------------------------
REM Symbolic Destination Name (Required if using APPC)
REM ---------------------------------------------------------------------
REM IDBSymDestName      = LUNAME
REM
REM
REM *****  Specify Client and Server Code Sets *****
REM ---------------------------------------------------------------------
REM Client Character Code Set (Optional)
REM ---------------------------------------------------------------------
IDBCharCodeSet      = IBM-850
REM
REM ---------------------------------------------------------------------
REM Server Character Code Set (Optional)
REM ---------------------------------------------------------------------
IDBServCharCodeSet  = IBM-037
REM
REMREM *****  Specify Log File Parameters  *************
REM ---------------------------------------------------------------------
REM Client Data Log Level (Optional)
REM ---------------------------------------------------------------------
IDBDataLogLevel     = 1
REM
REM ---------------------------------------------------------------------
REM Log File Size (Optional)
REM ---------------------------------------------------------------------
IDBLogFilesize      = 262144
REM
REM ---------------------------------------------------------------------
REM The Active Log File Name (Optional)
REM ---------------------------------------------------------------------
IDBLogFileNameActive = IDBLOG.ACT
REM
REM ---------------------------------------------------------------------
REM The Old Log File Name (Optional)
REM ---------------------------------------------------------------------
IDBLogFileNameOld    = IDBLOG.OLD
```

Environment Variables

HLAPI/NT uses Windows NT environment variables in two different ways. One environment variable can be set and used as a profile override. Other variables can be used to fully qualify the names of database and system profiles when a user does not do so. To see an explanation of the use of these variables, see “Profile Override” on page 159 or “Profile Search Path” on page 159.

Profile Override

Profile override specifications enable certain profile values to be specified through Windows NT environment variables. Not all profile parameters that can be specified in a profile can be overridden by environment variables. **IDBDataLogLevel** is currently the only variable that can be overridden this way. You can use the Windows NT environment variable **IDBDataLogLevel** to override the database profile parameter **IDBDataLogLevel**. The value of the Windows NT environment variable always takes precedence.

By setting profile overrides in the System dialog box, you can cause the profile override to effect all Windows NT sessions on the workstation. The System dialog box appears when you choose the System icon in the Control Panel window.

By using the **SET** command on the command line, the values you specify are only in effect for a single Windows NT session (the current one). An example follows.

Assume this is part of your current database profile:

```
REM ** Turn workstation logging on
IDBDataLogLevel = 1
REM ** Allow the workstation file to grow indefinitely
IDBLogFileSize = 0
REM ** Write the log file to BLX0TRAN.LOG
IDBLogFileNameActive = C:\INFOAPI\BLX0TRAN.LOG
```

This profile logs the data from all HLAPI transactions to the file C:\INFOAPI\BLX0TRAN.LOG. However, if you submit the command

```
SET IDBDataLogLevel=0
```

before you start the application that uses this profile, then no logging occurs on the workstation for transactions issued by the application. You have overridden the setting for **IDBDataLogLevel** for the current session. If you set IDBDataLogLevel in the System dialog box, you override the setting for all Windows NT sessions on this workstation.

Profile Search Path

Two environment variables are used to fully qualify the name of a database profile or system profile when the user does not do this. For additional information, see “IDBDBPATH” or “IDBSMPATH” on page 160.

IDBDBPATH

The search path to be used when an HLAPI/NT database profile name is specified without full qualification. If a user specifies a database profile name without a drive and path, HLAPI/NT first checks the current directory for the file with that name. If the database profile is not found there, HLAPI/NT searches the directories specified by the **IDBDBPATH** value. A sample default path is **C:\PROBLEM\HLAPINT**.

You can use this variable to specify multiple paths to search. For example:

```
SET IDBDBPATH=C:\;C:\PROBLEM\HLAPINT\;
```

causes the **C:** directory to be searched first, followed by **C:\PROBLEM\HLAPINT**.

Valid entries: Any valid file path qualifier. The last backslash (\) is optional.

IDBSMPATH

The search path to be used when an HLAPI/NT system profile name is specified without full qualification. If a user specifies a system profile name without a drive and path, HLAPI/NT first checks the current directory for the file with that name. If the system profile is not found there, HLAPI/NT searches the directories specified by the **IDBSMPATH** value. A sample default path is **C:\HLAPINT\REQ**.

You can use this variable to specify multiple paths to search. For example:

```
SET IDBSMPATH=C:\;C:\HLAPINT\REQ\;
```

This causes the **C:** directory to be searched first, followed by **C:\HLAPINT\REQ**.

Valid entries: Any valid file path qualifier. The last backslash (\) is optional.

Server Logging

The content of the server log produced by HLAPI/NT is similar to that of one produced by the HLAPI. Each HLAPI/NT logical session that has logging enabled has its various transaction data, results, and messages logged as each transaction is completed on the host. For more information about server logging, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*.

Transaction Logging

Each started HLAPI/NT Tivoli Information Management for z/OS logical session has one log file. The database profile parameter “IDBDataLogLevel” on page 156 (or the Windows NT environment variable used as an override) specifies whether data logging is enabled. If the parameter is not specified, then logging does not occur.

A logical session’s log entry on the host is identified by the **HLAPILOG_ID** PDB (refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*) passed on an HL01 transaction. This identifier is repeated for each transaction recorded in the server log.

The client writes the transaction to the log file specified by the database profile parameter “IDBLogFileNameActive” on page 156 until it reaches the size (in bytes) specified by the database profile parameter “IDBLogFileSize” on page 156. If you do not specify a log file name, the default name of **IDB_LOG.ACT** is used. The current log file is renamed to that specified by the parameter “IDBLogFileNameOld” on page 156. The default name for this parameter is **IDB_LOG.OLD**.

If an old log already exists it is deleted before the current log file is renamed. A new log file is created.

If two sessions are started specifying the same log file, then the first session that opens the log file has access to it, and the other session receives an error. When two sessions contend with each other for write access to the same log file, the following rules are followed to decide who can write to it.

- If the log file does not exist, then it is created and opened by the first session to ask for it.
- If the log file already exists, then it is opened and new log entries are appended to it by the first session.

- If the log file is already open, then this is not the first session to request it, and logging is not performed. The transaction continues to try to open the log until it reaches the internal retry limit, or is successful in opening the log. If it reaches the retry limit, then a return code and reason code are passed back in the HICA, indicating that logging was tried until the internal retry limit was reached.

When both HLAPI/NT and HLAPI logging are turned on, you may see differences in the PDBs that each one logs. The HLAPI/NT log shows any PDBs with a data length of zero. However, because HLAPI/NT does not send zero length PDBs to the server, the HLAPI log does not show any zero length PDBs. The HLAPI log also does not show the **SECURITY_ID**, **PASSWORD** and **DATABASE_PROFILE** PDBs, because the HLAPI/NT does not send them to the server.

HLAPI/NT Error Logging

When HLAPI/NT encounters an unexpected logic or system error, it automatically creates or updates an error log file on the workstation's startup (IPL) drive. This log file is always written to the same place on your workstation, regardless of where the user application started. Called the **IDBPROBE.LOG**, it is found in the **\INFOAPI** subdirectory (**\INFOAPI\IDBPROBE.LOG**). If the **\INFOAPI** subdirectory does not already exist, HLAPI/NT creates it before writing the file. It provides more information about errors than can be returned in the HICA return and reason codes.

You can delete or rename the log file any time after it is created. If you do, HLAPI/NT creates a new error log file if new error information must be recorded.

18

The HLAPI/NT Requester

The HLAPI/NT requester is a program that must be running on the workstation before any HLAPI/NT activity can occur.

Starting the Requester

The requester program is typically started by including its icon in the **STARTUP** group on the Windows NT system. An optional system profile name is passed as a parameter when starting the HLAPI/NT requester program. Any valid file name can be specified for the system profile name.

A sample system profile is copied to your workstation when the HLAPI/NT is installed. Look for **SYSTEM.PRO** in the **SAMPLE** subdirectory in the directory where HLAPI/NT is installed.

To start the HLAPI/NT, copy the requester's icon to the **STARTUP** group. If you choose to pass a system profile to the requester, open the **PROPERTIES** window for the requester's icon and type

```
IDBREQ [/P profile_file_name]SET IDBSMPATH=C:\;C:\HLAPINT\REQ\;
```

in the **Command Line:** field.

The command line parameters **/P** and the **profile_file_name** are optional. If you do not specify a file, default values are taken as specified in the system profile keyword list. The **profile_file_name** is preceded by a slash (**/**) followed by an uppercase or lowercase letter **P**. Separate the **/P** and the file name by at least one space.

The requester starts with a window that shows standard Tivoli copyright information. Select **OK** (or wait 15 seconds) and another window briefly appears to indicate that the requester is running. This window is the *requester run time window*. It minimizes to an icon if there are no errors starting the requester.

Stopping the Requester

You end the requester program by pressing the **EXIT** button on the requester run time window. A confirmation panel appears where you select **OK** to indicate that you want to exit the program. This action starts a shutdown of the requester environment, ending all conversations and freeing resources. If you select **CANCEL** from the confirmation panel, your request to exit the program is dropped, and the requester continues to run.

The requester program can also be closed by bringing the Task List window into focus and closing the process running the requester. However, the preferred method is to use the requester run time window.

Client user applications receive a *requester not started* return code (Return code=12, Reason code=109) for all transaction requests that occur after the requester is closed or before it is started.

Diagnosis of Some Common HLAPI/NT Problems

When attempting to diagnose unexpected results from your use of the HLAPI/NT, the *Tivoli Information Management for z/OS Diagnosis Guide* can help you analyze Tivoli Information Management for z/OS. Some common problems that can occur with the HLAPI/NT specifically are discussed in this section.

Changing the Profile and Its Effect on Program Operation

Perhaps you made a change to your database profile variables and it seems that nothing has changed in the way the program runs. For example, you change the setting for **IDBDataLogLevel** in your profile. The expected result does not occur when you next use the program. Check the following:

- Check your user application to determine which profile it uses to perform its task. HLAPI/NT looks at the database profile and the system profile.
- Verify that no environment variable (as a profile override) takes precedence over the setting in your database profile. In the current example, check the setting for this variable in the System dialog box. The variable setting there overrides the profile.

  You can use the Windows NT command **SET** to check current settings. If an override is in effect, check the System dialog box or any **.BAT** file that you run to find the profile override.
- Check the user application that you are running to see if it is setting the value of the environment variable (and thus the override) directly.
- HLAPI/NT might be reading a different profile with the same name, but which is located in a different directory from the profile you have changed. Use the Windows NT command **SET** to check each directory listed in the IDBDBPath environment variable for a file with the same name as the file that you are changing.

Data Conversion Problems

If you are getting return codes indicating there are problems pertaining to code set translation, refer to “Data Conversion Characteristics” on page 168 to ensure the code set translation tables are installed properly and the **LOCPATH** environment variable is properly set.

Establishing a Conversation with the Host

If the server you are using is a RES or an MRES with APPC, the APPC Transaction Scheduler (ASCH) on the MVS host system may be a source of failed connections. If you are not able to establish as many total sessions on your workstations as you expect to, check the **CLASSADD** command used to define the APPC class in the Transaction Scheduler that you are using for HLAPI/NT. To increase the number of sessions, increase the **MAX** value of the command, or define a separate class for your transaction program to run under.

If the server you are using is an MRES with TCP/IP or an MRES with APPC, the server must be started before HLAPI/NT can access it.

Establishing Too Many APPC Conversations

If you attempt to establish too many concurrent conversations using either a RES or an MRES with APPC, you may reach the system limits for APPC. Conversations you try to establish after the limit is reached are suspended until an earlier one shuts down. To expand the limit, update the APPC settings to the desired number.

19

HLAPI/NT Transactions

The work done by the HLAPI/NT takes place through the use of HLAPI transactions. For a list of all the transactions that are available to the HLAPI/NT, see "Clients" on page 2. Also, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* for an explanation of each transaction. Because the MVS and Windows NT environments are different, slight differences appear in the way the HLAPI/NT and the HLAPI use the same transaction. This section explains some differences to consider.

Transaction Operating Modes

A user application can select from two forms of transaction processing: synchronous or asynchronous. Users familiar with the Tivoli Information Management for z/OS Low Level Application Programming Interface (LLAPI) know that it selects either *all* synchronous or *all* asynchronous processing for Tivoli Information Management for z/OS database transactions within a session. With the HLAPI/NT you can select synchronous or asynchronous processing for a transaction at any time. This is also different from the HLAPI, which does not support asynchronous processing.

Synchronous Processing

Synchronous processing forces your user application's current thread to wait for a Tivoli Information Management for z/OS transaction to finish before it can perform any other work. The thread that submits the synchronous transaction does not receive control from the **IDBTransactionSubmit** HLAPI/NT service call until the transaction ends. You choose this mode of operation by coding a transaction type of **IDB_SYNC** (synchronous) on the **IDBTransactionSubmit** HLAPI/NT service call. Transaction completion includes both successful and unsuccessful outcomes.

You can implement the user application using the multitasking capabilities of Windows NT and use the synchronous mode of transaction processing. By using multiple threads within your application, one application thread can be dedicated to Tivoli Information Management for z/OS transaction processing while others perform other application duties. In this case, only the dedicated HLAPI/NT thread is blocked while the synchronous Tivoli Information Management for z/OS transaction finishes. Meanwhile, the other application threads continue to perform work.

CAUTION:

During synchronous processing, do not modify a HICA or PDB that you have submitted until after the IDBTransactionSubmit service call returns to the calling thread. Any changes to the HICA or its associated PDBs during transaction processing may cause unpredictable results.

Asynchronous Processing

Asynchronous processing enables your user application to submit a Tivoli Information Management for z/OS transaction and then continue performing application-related work while the submitted transaction finishes. After your application submits a transaction, control is immediately returned from the **IDBTransactionSubmit** service call to the application. Application processing and transaction processing occur concurrently. You choose this mode of operation by coding a transaction type of **IDB_ASYNC** (asynchronous) on the **IDBTransactionSubmit** HLAPI/NT service call.

After a transaction is submitted for asynchronous processing, your application must determine when the transaction finishes. Use the **IDBTransactionStatus** HLAPI/NT service call to do this. For every asynchronous transaction that returns an **IDBRC_NOERR** code from **IDBTransactionSubmit**, you must call the **IDBTransactionStatus** function to determine when the transaction ends or to change the transaction to a synchronous type.

Your user application can check for transaction completion either by polling the transaction or by converting the asynchronous transaction to a synchronous transaction. Polling is achieved by coding a transaction query type of **IDB_CHECKFORCOMPLETION** on the **IDBTransactionStatus** service call. If the transaction is finished, an **IDBTransactionStatus** of **IDB_TCOMPLETE** indicates completion. The data returned from an asynchronous transaction is not stored in the HICA until the status value **IDB_TCOMPLETE** is returned from an **IDBTransactionStatus** call. If the transaction is not finished, a status of **IDB_TINPROGRESS** indicates the transaction has not finished processing.

However, when HLAPI/NT no longer needs the conversation between the requester and the server to process a given transaction, the conversation is immediately free to be used by another transaction. This way, processing can start on another transaction with a different HICA using the same conversation, even before the **IDBTransactionStatus** call is performed for the current HICA.

If you want your application to convert the asynchronous transaction into a synchronous transaction, you can code a query type of **IDB_WAITFORCOMPLETION** on the **IDBTransactionStatus** service call. With this option, control is not returned from the **IDBTransactionStatus** service call until the transaction finishes. Transaction completion includes both successful and unsuccessful completions.

CAUTION:

During asynchronous processing, do not modify a HICA or PDB that you have submitted until after an IDBTransactionStatus service call returns the value of IDB_TCOMPLETE to the calling process. Any changes to the HICA or its associated PDBs during transaction processing may cause unpredictable results.

Data Conversion Characteristics

Windows NT uses the ASCII character set. MVS uses the EBCDIC character set. Thus, the HLAPI/NT requester and server each use a different character set, and character data exchanged between the host system and workstation requires conversion for the data to be useful in both environments.

HLAPI/NT converts the data at the workstation using the **ICONV** routine supplied by VisualAge for C++ for Windows. This routine is used internally by HLAPI/NT, so you are

not required to have VisualAge for C++ for Windows installed in order to use HLAPI/NT. The **ICONV** routine allows you to create your own translation tables. More detailed information can be found in the *VisualAge for C++ Programming Guide* and *VisualAge for C++ User's Guide*.

The environment variable **LOCPATH** must be set to point to the directory that contains the **\ICONV** and **\UCONVTAB** subdirectories. For example,

```
LOCPATH=X:\INFOAPI\LOCALE
```

where the subdirectories **X:\INFOAPI\LOCALE\ICONV** and **X:\INFOAPI\LOCALE\UCONVTAB** are located.

The **\ICONV** subdirectory contains **ICONV.LST**, **UCSTBL.DLL**, and **UTF-8.DLL**. The **ICONV.LST** has one line per conversion and two column entries per line. The first entry contains the input/output character code set and the second entry is the name of the DLL used to support the conversion.

The **\UCONVTAB** subdirectory contains the tables used for data conversion. The HLAPI/NT installation will install all or some of the code set translation tables depending on what option is selected during installation. If the HLAPI/NT Requester component is selected, only the files needed to convert EBCDIC (IBM-037) to ASCII (IBM-850) are installed. If the HLAPI/NT Toolkit component is selected, all translation tables that are shipped with VisualAge for C++ for Windows are installed. The tables are **IBM-***xxx* with no file extension. They may be packaged and distributed, according to the VisualAge license agreement.

Database Profile Parameters

The **IDBCharCodeSet** parameter specifies the code set the client application is using. If this parameter is not specified, then the default code set 850 (ASCII U.S. English) is used.

The **IDBServCharCodeSet** parameter specifies the code set the server is using. This code set is used to retrieve and store data in the Tivoli Information Management for z/OS database. If this parameter is not specified, then the default code set 37 (EBCDIC U.S. English) is used. See “Database Profile Keywords” on page 155 for more information about the database profile parameters.

Possible Truncation of Mixed SBCS/DBCS Data

Some HLAPI/NT data fields, such as the external record identifier (user defined) and privilege class, have a maximum defined size. The HLAPI/NT requester does not restrict mixed data from being entered (up to the maximum size) when such data is presented to HLAPI/NT for processing. During conversion from ASCII to EBCDIC, HLAPI/NT adds two bytes of data to each contiguous group of DBCS characters. If any data, after being converted between code sets by HLAPI/NT, is larger than its maximum defined field size, then truncation occurs while maintaining proper DBCS truncation and padding.

Your application program must ensure that critical data is not lost because of DBCS truncation. Be sure that enough spaces appear at the end of each line of freeform text, or at the end of each data field, so that only spaces are truncated during data conversion.

Differences between HLAPI/NT and HLAPI Transactions

The HLAPI/NT requester enables a user application in the Windows NT environment to use many of the transactions available on the Tivoli Information Management for z/OS HLAPI. Some differences do exist between transactions originating on the host and those originating on the workstation.

One global consideration is that the HLAPI/NT does not support text data sets.

The HLAPI/NT also requires several PDBs not used by the HLAPI. They are:

SECURITY_ID --
your MVS userid (maximum length 8 characters)

PASSWORD --
your MVS password (maximum length 8 characters)

DATABASE_PROFILE --
your database profile, described in "Database Profile Keywords" on page 155

These PDBs are explained in "Initialize Tivoli Information Management for z/OS (HL01)".

Initialize Tivoli Information Management for z/OS (HL01)

The Tivoli Information Management for z/OS HLAPI transaction HL01 requests a connection to a database on a specific Tivoli Information Management for z/OS server. The steps that occur when a user application requests an HL01 transaction are outlined in this section.

1. The application author creates a database profile for the Tivoli Information Management for z/OS database connection. Use a text editor to create the database profile. See "HLAPI/NT Profiles, Environment Variables, and Data Logging" on page 153 for a list of valid profile keywords. You should, but are not required to, create a unique database profile for each database connection that the application uses.

   Information obtained from the database profile includes the particular server you want to establish a conversation with. If you are using a RES or an MRES with APPC, then the database profile must include the symbolic destination name to use when establishing the APPC conversation to the server. If you are using an MRES with TCP/IP, then the database profile must include the host name or IP address of the MVS host where the server resides and optionally, you can specify a service name associated with the server. See "Database Profile Keywords" on page 155 for more information.

2. The user application initializes a HICA structure for a given logical database. It inserts a **DATABASE_PROFILE** PDB onto the Control PDB chain to specify the name of the database profile that is to be used for the database connection.

   The database profile file name can be fully qualified (drive and subdirectory path), or just the name can be specified. If only the name is specified, the current directory is searched for the specified database profile. If it is not found in the current directory, the drive and path are obtained from the **IDBDBPath** environment variable.

3. Create a **SECURITY_ID** PDB and a **PASSWORD** PDB, and any additional PDBs for the HL01 transaction, and place them on the Control PDB chain.

4. The user application submits the transaction for processing by using the HLAPI/NT service call **IDBTransactionSubmit**.

5. HLAPI/NT looks for the **DATABASE_PROFILE** PDB in the Control PDB chain in the HICA. The database profile is read, and the specifications are recorded for use throughout the specified logical Tivoli Information Management for z/OS session.
6. The server information is obtained from the database profile. If a symbolic destination name is specified, and a conversation associated with the symbolic destination name is not already established, it is established now. If a server host and optionally, a server service name are specified, and a conversation associated with the server host and service is not already established, it is established now. If the HLAPI/NT requester was started with a system profile in which IDBSHARECMS was set to 1, an established conversation is used for the session when the same server is specified, and the same **SECURITY_ID** and **PASSWORD** are specified. Otherwise, a new conversation is established.
7. The **TIMEOUT_INTERVAL** PDB applies to the HLAPI that is running on Tivoli Information Management for z/OS. (Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* for more information.) If you specify a timeout interval, it determines the interval of the HLAPI running on the server. HLAPI/NT processing time and communications time between the workstation and the host are not considered for this timeout interval. Therefore, a transaction submitted from the workstation may take longer to time out than this value indicates.
8. Code sets for workstation and host processing are established when an HL01 transaction is submitted. They remain in effect until an HL02 (terminate) transaction is submitted for this particular HICA.
9. For information about logging, see “Transaction Logging” on page 160 and and “Server Logging” on page 160.
10. The rest of the processing for the HL01 transaction is normal HLAPI/NT processing.

After you establish the main database connection with your first HL01 transaction, your application can issue multiple HL01 transactions in the same conversation to the server that is started. Each of these transactions creates a new database connection, and each of them performs tasks without affecting the others. Each of these connections can also be stopped without affecting any of the others. A graphic representation of this transaction nesting effect looks like this:

```
HL01 (initialize main session, session 1)
    HL01  (initialize session 2)
    HL01  (initialize session 3)
    HL02  (terminate session 2)
    HL02  (terminate session 3)
HL02  (terminate session 1)
```

Sessions 2 and 3 run independently of each other, and ending either of them does not affect session 1. Multiple logical sessions on the same conversation can affect each other, because the requester waits on APPC or TCP/IP to send and receive a request before processing the next request.

Logical Session and Process Ownership

When an HL01 transaction is performed from within a client application, all further transactions associated with this HL01 transaction must be performed by the same process. The HICA and PDB chains can be shared across threads within the same process. For example, one thread in a client application can submit a transaction, and a different thread in the same application can process the results.

Terminate Tivoli Information Management for z/OS (HL02)

The Tivoli Information Management for z/OS HLAPI transaction HL02 closes a database connection on a specific Tivoli Information Management for z/OS server. The steps that occur when your user application requests an HL02 transaction follow.

1. The user application creates the normal Control PDBs requesting a disconnect from the logical database associated with the HICA and submits the transaction using the **IDBTransactionSubmit** service call.
2. Normal HLAPI/NT transaction processing requests a database disconnect from the corresponding Tivoli Information Management for z/OS server. See “Basic Transaction Flow” on page 139 for a description of this processing.
3. The HLAPI/NT examines the resulting return code to determine that the disconnect finished successfully. If this is the last active HL01 connection on this conversation, then the conversation is closed. If this is not the last active HL01 connection, then the conversation continues.
4. The appropriate return codes are set and control returns to the caller.

Retrieve Record (HL06)

The difference between this transaction on the HLAPI/NT and the HLAPI is:

- The optional PDB called **TEXT_MEDIUM** supports only one storage media type for HLAPI/NT. The only type supported is type **B**. If you omit this value or specify any character other than **B**, the HLAPI/NT assumes the value of **B**.
- If you want to retrieve freeform text as a continuous stream of data with carriage return / line feed characters (ASCII X'0D0A') after each text line, set the control PDB **TEXT_STREAM** to **YES**. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information about the **TEXT_STREAM** PDB.

Create Record (HL08)

The difference between this transaction on the HLAPI/NT and the HLAPI is:

- Text data sets are not supported. The **PDB_DATW** field in the input PDBs should always be specified with a nonzero value for text data.
- If you are creating a record that contains freeform text, and the input text contains either line feed characters (ASCII X'0A') or carriage return / line feed characters (ASCII X'0D0A'), set the control PDB **TEXT_STREAM** to **YES**. This will ensure that text formatting information is stored in the record. When the text is retrieved, it will be formatted exactly as it was entered. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information about the **TEXT_STREAM** PDB.

Update Record (HL09)

The difference between this transaction on the HLAPI/NT and the HLAPI is:

- Text data sets are not supported in HLAPI/NT. The **PDB_DATW** field in the input PDBs should always be specified with a nonzero value for text data.
- If you are updating a record that contains freeform text, and the input text contains either line feed characters (ASCII X'0A') or carriage return / line feed characters (ASCII X'0D0A'), set the control PDB **TEXT_STREAM** to **YES**. This will ensure that text formatting information is stored in the record. When the text is retrieved, it will be

formatted exactly as it was entered. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information about the **TEXT_STREAM** PDB.

20

Tips for Writing a HLAPI/NT Application

This chapter describes the steps typically involved in creating an application that uses the Tivoli Information Management for z/OS HLAPI/NT. Every programmer has a certain technique or style for designing applications, so think of this chapter as more of a set of guidelines than as a set of rules. Refer to the sections about ″Tips for Writing An API Application″ and ″Tailoring the APIs″ in the *Tivoli Information Management for z/OS Application Program Interface Guide* for more information on this subject. Refer to “Choosing a Server” on page 13 for more information on the Tivoli Information Management for z/OS servers.

Determine What You Want Your Application to Do

The first step in creating an application is to determine exactly what you want it to do. After you decide that, consider:

- Which Tivoli Information Management for z/OS functions (for example, create or update) do you use?
- Which record types (for example, problem or change) do you use?
- Which fields (for example, status or assignee name) do you use?
- On which MVS system is your Tivoli Information Management for z/OS database located?
- Do you need to connect to more than one Tivoli Information Management for z/OS database?
- Are the databases on the same or different MVS systems?
- Which server do you use to access the databases?
- Do you want different Windows NT processes (or threads) to manage the different database connections, or do you want to use just one?
- How much storage do you need in the host address space to handle requests from the workstation?
- Do you want data logging enabled on the host? On the workstation?
- What data validation (if any) do you want to perform?

Converting C Programs

If you want to convert an existing C program that uses the HLAPI to a C program that uses HLAPI/NT, here are some general instructions on how to do it.

- Make any general modifications to modify the program to run on Windows NT, including fixing {} pairs that might have been mistranslated when the files were transferred from one environment to another, and modifying the parameters passed to the main procedure. This step is required if the download program you use to copy the code from MVS to Windows NT translates the source improperly.
- Be sure you:
  - Include **IDBH.H** header file. Do *not* include **BLGUHLC** header file.
  - Include **IDBECH.H** if you want constant declarations of HLAPI/NT return codes.
  - Do *not* include **spc.h**.
- Delete #pragmas used for the MVS program.
- Convert any data set method freeform text processing for HL06, HL08, and HL09 to use buffer method freeform text processing.
- Convert HLAPI call syntax to HLAPI/NT format: use **IDBTransactionSubmit** and **IDBTransactionStatus**. Do not define variables to point to the **BLGYHLPI** module. Do not call the **BLGYHLPI** module.
- Add processing to create and initialize the three HLAPI/NT-specific control PDBs: **SECURITY_ID**, **PASSWORD**, and **DATABASE_PROFILE**. Add them to the control PDB linked list.
- Build a database profile for use with the HLAPI/NT sessions you will be starting (or multiple database profiles if necessary).
- Review the error handling you use after the call to the HLAPI to see if changes are required to handle the error codes specific to the HLAPI/NT.
- If you are using an MRES with APPC or a RES:
  - Set up the CPIC side information entries to communicate with the particular MVS host. Specify these names in the database profile.
  - Set up APPC/MVS on the host to accept conversations from the workstations that your application runs on.
  - Set up the MRES with APPC or the RES on the host.
- If you are using an MRES with TCP/IP:
  - Set up TCP/IP to communicate with the particular MVS host, and the particular MRES you want to use. Specify the information in the database profile.
  - Set up the MRES with TCP/IP on the particular MVS host you want to use.

Installation and Setup Summary for HLAPI/NT Sample Applications

1. If you are using an MRES with TCP/IP, install and configure TCP/IP to connect your Windows NT workstation to your MVS host system. If you are using a RES or an MRES with APPC, install and configure APPC/MVS and your APPC client software to connect your Windows NT workstation to your MVS host system.
2. If you are using a RES, create a TP profile on the MVS host that brings up Tivoli Information Management for z/OS and the Remote Environment. This must include the appropriate BLX-SP load module. Refer to the *Tivoli Information Management for z/OS Planning and Installation Guide and Reference* and *Tivoli Information Management for z/OS Operation and Maintenance Reference* manuals, or “Configuring and Running a Remote Environment Server (RES)” on page 25 in this manual, for more information.

3. If you are using an MRES with APPC or an MRES with TCP/IP, create a cataloged procedure with the JCL to start the MRES. If you are using an MRES with APPC, define the MRES to APPC and VTAM. The MRES must be started before you attempt to establish a conversation with it. See “Configuring and Running a Multiclient Remote Environment Server (MRES) with APPC” on page 35 and “Configuring and Running a Multiclient Remote Environment Server (MRES) with TCP/IP” on page 53 for additional information.
4. Install Tivoli Information Management for z/OS including building a session parameters member. You can also build the session parameter **BLGSES00** at this time, which enables you to skip one modification to the sample application.
5. Bring up Tivoli Information Management for z/OS as an interactive user using the session parameters member that you want the sample application to use. This ensures that Tivoli Information Management for z/OS can be brought up using that session member.
6. Install HLAPI/NT. Refer to “Installing and Configuring HLAPI/NT” on page 143 for information about how to do this.
7. Set up a database profile for use by the sample program. A sample database profile is copied to your Windows NT workstation during the installation of HLAPI/NT. This profile is in the **SAMPLE** subdirectory in whatever directory you chose for HLAPI/NT. Look for **DATABASE.PRO**. To fit your particular system, if you are using TCP/IP, be sure to modify the server host and server service names, or, if you are using APPC, modify the symbolic destination name.
8. To complete the setup and run the sample program, you will need to perform additional steps listed in “Steps Required to Run the HLAPI/NT C Sample Program” on page 184.

21

HLAPI/NT C Language Application Program

The HLAPI/NT interface enables you to access the HLAPI transactions from a Windows NT C language application. A C language client application program communicates with the Tivoli Information Management for z/OS system by creating a high-level application communication area (HICA) and its related parameter data blocks (PDBs). The client application program then submits the HICA transaction by making HLAPI/NT program service calls. The HLAPI/NT program service routines exist on the user's workstation as a dynamic link library (DLL). When the calls are made by the client application program, the supporting HLAPI/NT routines are loaded from the DLL and started.

Allocating HICAs and PDBs

To utilize HLAPI/NT program service calls, your user application must include a C language-based header file (named IDBH.H) in its source file and specify **IDBHLAPI.LIB** as one of the link libraries. Linking to **IDBHLAPI.LIB** gives you access to the HLAPI/NT service calls. To use the service calls, your user application must:

- Allocate HICA and PDB structures using the types provided in the header file
- Include the code for the HLAPI/NT program service calls.

The HICA and PDB data structures that you submit to HLAPI/NT must exist for the entire time that a transaction using them is being processed. Make sure that your program does not accidentally deallocate them, such as by using local variables in a subroutine, then exiting the subroutine while HLAPI/NT is processing your request.

The required header file **IDBH.H** provides the type definitions for both the HICA and PDB data structures. See "Allocating and Initializing a HICA" on page 180 and "Allocating and Initializing a PDB" on page 180 for more information.

Including the Header File in Your Program

Before using HLAPI/NT, you must first include the HLAPI/NT header file into your program source. This file identifies the HLAPI/NT call prototypes and the various type definitions and constants your program uses to communicate information to the HLAPI/NT-connected system. To include the HLAPI/NT header file into your source file, put the following line into your source file before you make any HLAPI/NT service calls and before you define any HICAs or PDBs.

```
#include  "IDBH.H"   /* Include the HLAPI/NT header file. */
```

The header file is in the **H** subdirectory in the directory in which you installed HLAPI/NT.

Allocating and Initializing a HICA

Your application must allocate and initialize at least one HICA data structure to communicate through the HLAPI/NT. The values you put in the control and input PDB chains depend on the specific HLAPI/NT transaction you want to use. PDB allocation is discussed in “Allocating and Initializing a PDB”. The way in which the user application initializes the control and input PDB chains is covered in the *Tivoli Information Management for z/OS Application Program Interface Guide* under the individual transaction discussions. The following shows part of an application that illustrates the declaration and partial initialization of a HICA structure for HLAPI/NT.

```
/* Allocate a HICA structure.  */
static HICA     MyHICA;

/* Initialize HICA eyecatcher to "HICA".  */
memcpy(MyHICA.ACRO, HICAACRO_TEXT,
       HICAACRO_MAX_SIZE);

/* Initialize the HICA fields to NULL  */
MyHICA.ENVP = NULL;     /* pointer to environment block  */

MyHICA.OUTP = NULL;     /* pointer to output pdb chain   */

MyHICA.CTLP = NULL;     /* pointer to control pdb chain  */

MyHICA.INPP = NULL;     /* pointer to input pdb chain    */

MyHICA.MSGP = NULL;     /* pointer to message pdb chain  */

MyHICA.ERRP = NULL;     /* pointer to error pdb chain    */

MyHICA.LENG = sizeof(HICA);    /* HICA data block length */
```

An alternate method of allocating the HICA structure using typical C language calls is:

```
pMyHICA = malloc(sizeof(HICA));
```

Addressing the elements of the HICA then follows the **’pMyHICA->’** (pointer) notation rather than the **’MyHICA.’** (dot) notation.

A full description of the contents of a HICA appears in the *Tivoli Information Management for z/OS Application Program Interface Guide* in the section about HLAPI structures.

Allocating and Initializing a PDB

Parameter information is communicated between the user application and the HLAPI/NT-connected system through PDBs. PDBs can be several types: control, input, output, error, or message. All PDBs of the same type are organized as a linked list. Each linked list type is anchored to a specific field within a HICA. For a description of the HICA fields, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*.

Your user application must create any required control and input type PDBs before it submits a transaction. Your user application should examine the output, error, and message PDB chains that are returned when a transaction completes processing.

Part of an application follows. It illustrates the declaration and initialization of an input PDB chain containing one PDB.

Note: The PDB data structure has a variable length whose value can be very large. To conserve storage, the example below shows allocation of the PDB with only as much storage as it needs. The total storage in this case is the sum of the length of the data fields all PDBs have, plus the length of the variable field at the end of the PDB. The file **IDBH.H** contains the variable **PDBFIX_SIZE** to make this programming more convenient.

```
/* Temporary PDB pointer variable.  */
PPDB     pTempPDB;

/* Allocate the PDB memory.  */
pTempPDB = malloc(PDBFIX_SIZE + strlen("John Doe"));

/* Because this is the first PDB, init the NEXT PDB  */
/* pointer to null. */
pTempPDB->Next = NULL;

/* Init the previous PDB pointer to null.   */
pTempPDB->Prev = NULL;

/* set the PDB eyecatcher to "PDB "  */
memcpy(pTempPDB->Acro, PDBACRO_TEXT,
       PDBACRO_MAX_SIZE);

/* Set the parameter data item name to REPORTER_NAME */
memset(pTempPDB->Name, ' ', PDBNAME_MAX_SIZE);
memcpy(pTempPDB->Name, "REPORTER_NAME",
       strlen("REPORTER_NAME"));

/* fill the parameter with the name John Doe  */
memcpy(pTempPDB->Data, "John Doe",
             strlen("John Doe"));

/* Perform data response validation.   */
pTempPDB->Proc = 'V';

/* Initialize the PDB error code to a blank.   */
pTempPDB->Code = ' ';

... and the rest of the PDB is initialized in similar ways ...


/* Place the new PDB on the INPUT PDB chain */
MyHICA.INPP = pTempPDB;
```

Binding Prototypes

The two entry points into HLAPI/NT are:

- “IDBTransactionSubmit” on page 182
- “IDBTransactionStatus” on page 182

Your application must allocate data structures conforming to the data types defined in the **IDBH.H** header file. HLAPI/NT program options must be specified by using the constants also supplied by the **IDBH.H** header file. Descriptions of the bindings and related data types follow.

Note: Fields are specified differently for the HLAPI/NT HICA and PDB structures than for those in the HLAPI. In the Windows NT client environment, field names are similar, but they might have been extended to conform with standard C language naming conventions. For more information about the HICA and PDBs, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*.

IDBTransactionSubmit

The IDBTransactionSubmit program call is used to submit a transaction to the HLAPI/NT system. In a C language program, the call looks like the following:

```
rc = IDBTransactionSubmit(pHICA, TranType);
```

The two variables that your application must provide are:

- pHICA

  A pointer to a structure of the type HICA that contains the HICA that you want to submit to HLAPI/NT

- TranType

  Your selection of the type of transaction to perform. This variable has a type definition of **TRANTYPE_TYPE**, and can be one of these two values:
  IDB_SYNC --- Synchronous transaction processing
  IDB_ASYNC -- Asynchronous transaction processing

HLAPI/NT returns a value from this function call that you should examine before looking at the HICA return and reason codes. This return code (rc) is a variable of type **IDBRC_TYPE**. The values that can be returned for it appear in “HLAPI Service Call Return Codes” on page 367. This is an example of how this routine might be used:

```
#include "idbh.h"
#include <stdio.h>
#include "MYCODE.H"

main()
{
HICA MyHICA;
IDBRC_TYPE rc;

Initialize_HICA(&MyHICA);         /* routine that sets up the data */
                                    /* for an HL01 transaction.      */

rc = IDBTransactionSubmit(&MyHICA,IDB_SYNC);
switch (rc)
  {
  case (IDBRC_NOERR) : Main_Loop(&MyHICA);  /* The rest of your    */
                                              /* application program.*/
                     break;
  case (IDBRC_XERR)  : Process_Error(&MyHICA);  /* A non-zero HICA */
                                    /* return and reason code was    */
                                    /* detected.                     */
                     break;
  case (IDBRC_BADHICA): printf("The HICA data structure is corrupt\n");
                      return(-1);
  case (IDBRC_BADPARM): printf("Incorrect value passed on call\n");
                      return(-2);
  case (IDBRC_SYSERR) : printf("HLAPI/NT encountered a System Error\n");
                      printf("Check the file IDBPROBE.LOG.\n");
                      return(-3);
  }
}
```

IDBTransactionStatus

The **IDBTransactionStatus** program call is used to request the status of an asynchronous transaction. It can also be used to convert an asynchronous transaction to a synchronous transaction. In a C language program, the call looks like this:

```
rc = IDBTransactionStatus(pHICA, QueryType, pTStatus);
```

The three variables that your application must provide are:

- **pHICA**

  A pointer to a structure of type HICA that contains the HICA whose status you want to check.

- **QueryType**

  This input value is of the type **QUERYTYPE_TYPE**, and has one of the following two values:

  - **IDB_CHECKFORCOMPLETION**

    This value causes HLAPI/NT to check the current status of a transaction submitted on the HICA provided in pHICA. This program call returns to your application immediately, no matter what state the transaction is in.

  - **IDB_WAITFORCOMPLETION**

    This value causes HLAPI/NT to change the asynchronous transaction to a synchronous transaction. This program call does not return to your application until the transaction processing has finished.

- pTStatus

  This is a pointer to a variable of type **TRANSTATUS_TYPE**. It can have one of the following two values:

  - **IDB_TCOMPLETE**

    The transaction has completed processing. Any Output, Error, and Message PDBs returned from the host are attached to the HICA and available for your program's use. The **RETC** and **REAS** fields of the HICA are set with values indicating the result of the transaction (the *Tivoli Information Management for z/OS Application Program Interface Guide* contains possible values).

  - **IDB_TINPROGRESS**

    The transaction is still in progress. No data has been returned on the HICA, and the HICA is unavailable for use by your application.

HLAPI/NT returns a value from this function call which you should examine before looking at the HICA return and reason codes. This return code (rc) is defined as a variable of type **IDBRC_TYPE**. The values that can be returned for it appear in “HLAPI Service Call Return Codes” on page 367. Here is an example of how this routine might be used:

```
#include "idbh.h"
#include <stdio.h>
#include "MYCODE.H"
main()
{
HICA MyHICA;
TRANSTATUS_TYPE MyStatus;
IDBRC_TYPE rc;

Initialize_HICA(&MyHICA);   /* routine that sets up the data   */
                               /* for an HL01 transaction.       */

rc = IDBTransactionSubmit(&MyHICA,IDB_ASYNC);  /* Use Asynchronous */
                               /* processing so this code has the */
                               /* ability to maintain a window    */
if (rc!=IDBRC_NOERR)           /* Detected a non-zero return code */
```

```
     Process_rc(rc,&MyHICA);
  else
    {
    MyStatus = IDB_TINPROGESS;    /* Start the while loop out right  */
    while(MyStatus == IDB_TINPROGRESS)
      {
      rc = IDBTransactionStatus(&MyHICA,
                                IDB_CHECKFORCOMPLETION,
                                &MyStatus);  /* Keep checking this*/
                                  /* until MyStatus == IDB_TCOMPLETE  */
      if (rc != IDBRC_NOERR)
        Process_rc(rc,&MyHICA);
      Process_Window();           /* Handle the window duties        */

/* Note:  If you do not have time-critical duties to perform in this */
/*        loop, you may want to use the Sleep() routine, or some     */
/*        similar routine to prevent 100% CPU usage.                 */

      }
    }
 .
 .
 .
 }
```

Linking Your Program

Before using HLAPI/NT, you must link to **IDBHLAPI.LIB**. This is an import library that resolves the external references to the HLAPI/NT service routines you use to perform HLAPI/NT functions. Look at the **BLM2SAMI.BAT** or **BLM2SAMM.BAT** file in the **SAMPLE\C** subdirectory of the directory in which you installed HLAPI/NT. It illustrates one way to link **IDBHLAPI.LIB** with your application.

Sample HLAPI/NT C Program

A sample program **BLM2SAM1.C** is installed on your workstation during installation of HLAPI/NT. Look for it in the directory **SAMPLE\C**, which is a subdirectory of the directory where HLAPI/NT is installed. The sample shows the setup and start of HLAPI/NT. It includes the header file **IDBH.H**. The sample C code performs the following functions:

1. Initialize the HLAPI by performing an HL01 transaction.
2. Obtain a system-assigned record ID and save it to use for the create transaction. (This step is not mandatory because the HL08 transaction generates a record ID if one is not specified).
3. Create a record using the previously obtained record ID.
4. Update several fields in the record just created.
5. Retrieve the record just created and updated, and print the fields just retrieved.
6. Search for all records created today by this program and print the search results.
7. End the HLAPI with an HL02 transaction.
8. Perform cleanup.

Steps Required to Run the HLAPI/NT C Sample Program

1. Perform the steps described in “Installation and Setup Summary for HLAPI/NT Sample Applications” on page 176.

2. Modify the file **BLM2SAM1.C**. You may want to back up **BLM2SAM1.C** before making any changes to it. This program is in the **SAMPLE\C** subdirectory in the directory you chose for HLAPI/NT.

   - Find the **SESSMBR** #define near the beginning of **BLM2SAM1**. Change the value of this #define to the name of the session parameters member you want to use. If you want to use **BLGSES00**, no changes are required.
   - Find the **PRIVCLAS** #define. Change the value of this #define to the name of the privilege class to be used on the HL01 (initialize) transaction. This privilege class must be in the database defined in your chosen session parameters member. The privilege class must have authority to display, create, and update problem records.

     If you want to use **MASTER** privilege class, no changes are required.
   - Find the **APPLID** #define. Change the value of this #define to the name of the application ID to be used on the HL01 (initialize) transaction. The value you choose must be defined as an eligible user in the privilege class you use.
   - Find the **SECID** and **PASSWORD** #defines. Put in the appropriate values for the security ID and password for the ID you want to use on the MVS system.
   - Find the **DBPROF** #define. Use the name of the database profile or the sample profile shipped with HLAPI/NT (**DATABASE.PRO**).

     The database profile must be in the subdirectory where the sample program resides or in the path defined on your system in the **IDBDBPATH** variable. See "IDBDBPATH" on page 159 for more information about this variable.
3. To compile and link **BLM2SAM1.C** using VisualAge for C++ for Windows, run **BLM2SAMI.BAT** or to compile and link using Microsoft Visual C++, run **BLM2SAMM.BAT**. First, verify that the **LIB** environment variable contains the directory that holds **IDBHLAPI.LIB**. Verify that the **INCLUDE** path contains the directory that holds **IDBH.H**.
4. Start the HLAPI/NT requester. Refer to the "The HLAPI/NT Requester" on page 163 for more information.
5. If you are running an MRES with TCP/IP, be sure that TCP/IP is running. If you are running a RES or an MRES with APPC, be sure that your APPC server is running.
6. Run the program **BLM2SAM1**.

22

Introduction to HLAPI/CICS

Application programmers can develop CICS applications that issue Tivoli Information Management for z/OS HLAPI calls for any Tivoli Information Management for z/OS database. You can do this by using the Tivoli Information Management for z/OS High-Level Application Program Interface Client for CICS (HLAPI/CICS or client), a remote environment client that Tivoli Information Management for z/OS supports. This client does not extend the CICS function set; however, it does enable CICS transactions to retrieve and update Tivoli Information Management for z/OS data, just as the existing Tivoli Information Management for z/OS programs do. The HLAPI/CICS client can connect to either Tivoli Information Management for z/OS server that supports APPC.

HLAPI/CICS Overview

HLAPI/CICS is the client interface for Tivoli Information Management for z/OS that enables the CICS programmer to access and update data in the Tivoli Information Management for z/OS database through a Tivoli Information Management for z/OS server. The HLAPI/CICS uses the APPC protocol to access a Tivoli Information Management for z/OS server. This allows a CICS client application program to access a server on the same machine or on a different machine. The CICS client can even access multiple Tivoli Information Management for z/OS servers on different machines.

With the HLAPI/CICS, multiple CICS clients can access the Tivoli Information Management for z/OS data at the same time.

The HLAPI/CICS can use synchronous processing only, which requires the client application program to submit Tivoli Information Management for z/OS work and wait for its completion. The client application program cannot do any other work until the transaction finishes.

The HLAPI CICS interface provides functions and transactions that are similar to those of the HLAPI. The transactions available to an application using the HLAPI/CICS are listed in Table 1 on page 3. For a complete description of all these transactions, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*.

The HLAPI/CICS components are:
- CICS transactions that use CICS commands
- Installed as CICS transactions using CICS functions
- Managed as CICS transactions using CICS functions.

No calls or interfaces to system functions other than CICS functions are supported. The CICS client does not require any interface or binding code because all the interfaces are through CICS commands. You can write a client application program in any language supported by both CICS and Tivoli Information Management for z/OS.

The CICS client is made up of three pieces that interface to a Tivoli Information Management for z/OS server:
- CICS interface
- Communication Manager (CM)
- Termination handler (TH)

Note: It is important to note that the Communication Manager (CM) component of HLAPI/CICS is in no way related to the OS/2 Communications Manager/2 program.

The Communication Manager (CM) is a long running CICS transaction that manages an APPC session connected with a Tivoli Information Management for z/OS server. Because the CM maintains an active communication environment with the server, CICS pseudoconversational design techniques are maintained. The CICS interface is established with a CICS transaction performed by an **EXEC LINK** command from the client application program. It locates the correct CM instance for the Tivoli Information Management for z/OS environment and connects to it, making the connection to the server. When the server returns the information to the CM, the CM in turn sends the information to the CICS interface, which returns the data to the client application program and then ends. The termination handler, a transaction called by the CICS ending process, closes all existing conversations with the Tivoli Information Management for z/OS server, ensuring nothing is left unfinished if CICS is stopped.

Server Overview

A Tivoli Information Management for z/OS server is an MVS/ESA transaction program that handles all communication between an HLAPI/CICS application and any Tivoli Information Management for z/OS databases that reside on the MVS system where the server is installed. The Tivoli Information Management for z/OS server processes transactions from HLAPI/CICS by converting calls into HLAPI high-level application communication areas (HICAs) and parameter data blocks (PDBs). A CICS client application program cannot directly access the layer of software that runs on the OS/390 host machine.

The HLAPI/CICS client can communicate with either a RES or an MRES with APPC. A server must be set up on every MVS/ESA machine running a Tivoli Information Management for z/OS database that an application using the HLAPI/CICS needs to access.

HLAPI/CICS Basic Transaction Flow

Figure 12 shows the organization of the functional components and the sequencing of a call from a CICS transaction to a Tivoli Information Management for z/OS RES and back.

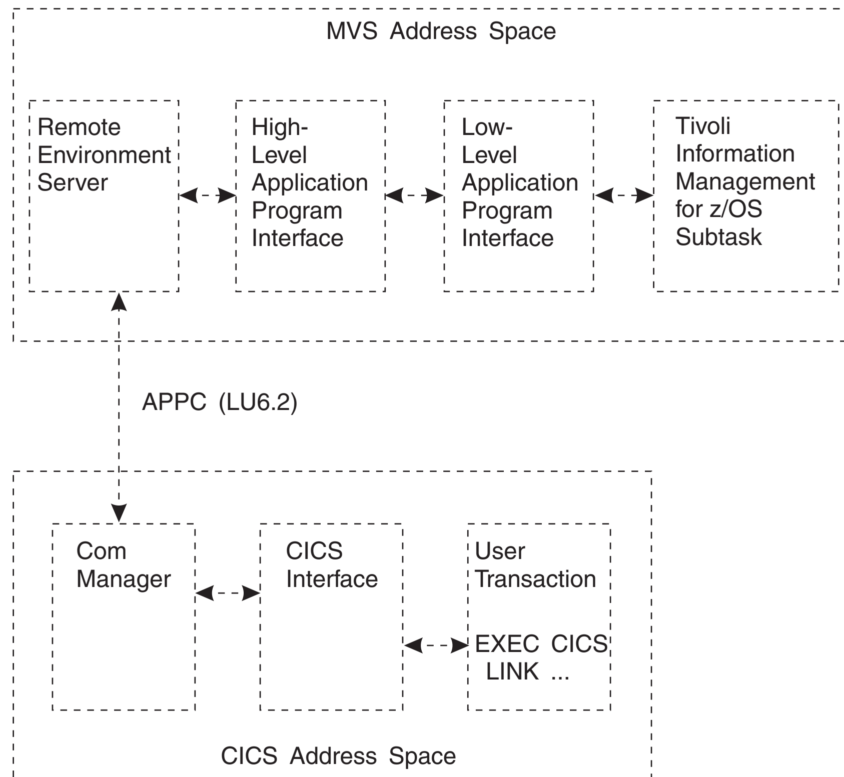


Figure 12. HLAPI/CICS Overview

The CICS application calls a HLAPI/CICS function that passes the Tivoli Information Management for z/OS HLAPI requests to the correct target system. The first call from the CICS application to the HLAPI/CICS must be an HL01 transaction that causes the dynamic invocation of the HLAPI/CICS Communication Manager (CM). The call passes information to start a conversation with a Tivoli Information Management for z/OS server.

The HLAPI requests are in the form of Tivoli Information Management for z/OS HICAs and PDBs (control and input) that the CICS client application program passes by means of a HICA address. The HLAPI/CICS initiates an APPC conversation between the Tivoli Information Management for z/OS server identified in the control PDBs and the HLAPI/CICS Communication Manager. The Communication Manager anchors the CICS side of the APPC conversation and identifies the Tivoli Information Management for z/OS server it wants to communicate with. One Communication Manager is active for each combination of terminal ID and user ID that is making Tivoli Information Management for z/OS HLAPI calls to a Tivoli Information Management for z/OS server (Partner ID).

The HLAPI/CICS returns all Tivoli Information Management for z/OS HICA and PDB blocks sent from the HLAPI to the calling CICS user application by means of HICA and PDB addresses using shared memory. The calling CICS user application can then retrieve this information and perform the required processing. The calling CICS user application is responsible for freeing the memory it allocated for the HICA, control PDBs, and input PDBs. The Communication Manager frees the memory it allocated for the output, error, and message PDBs when the next transaction for a specific Tivoli Information Management for z/OS environment is processed.

Requirements

Software

The software requirements for the HLAPI/CICS follow:

- OS/390 Version 2.5, or a later version
- CICS/ESA Version 4 Release 1 (5655–018) installed on the machine where the HLAPI/CICS client resides. This release provides support for CPI-C.
- An APPC/APPN network link to a machine with a Tivoli Information Management for z/OS server installed.

Note: The sample CICS application that is shipped by Tivoli requires VS COBOL II Version 1.4 or Language Environment® (LE) to run. HLAPI/CICS itself does not require VS COBOL II.

Hardware

The HLAPI/CICS client must be installed on a host system that can run CICS/ESA. If Tivoli Information Management for z/OS and the HLAPI/CICS do not reside on the same MVS system, you must have a communications link between the two systems.

23

Installing HLAPI/CICS and Customizing CICS/ESA

This chapter describes the tasks required to enable your CICS installation for Tivoli Information Management for z/OS HLAPI/CICS:

- Install HLAPI/CICS object files and link-edit the load modules
- Customize the CICS environment
- Online customizing of the CICS environment.

Installing HLAPI/CICS

To install HLAPI/CICS, use SMP/E to extract the object files from the distribution tape, and link-edit them into the CICS load libraries. At the same time, you can install the sample COBOL application. The FMID of HOYA201 identifies the component to be installed and the load modules should be placed in **SDFHLOAD** for running as normal CICS transactions. The modules are all re-entrant and reusable and should be linked **RMODE=ANY**. The load modules have the following names:

- BLMYKCOM
- BLMYKINF
- BLMYKTRM

Note: Upgrades or patches that can be downloaded from a Tivoli Web site may be available for HLAPI/CICS. Visit the Tivoli Information Management for z/OS Web site for more information:

http://www.tivoli.com/infoman

Customizing CICS/ESA for HLAPI/CICS

The following areas of CICS require customization for HLAPI/CICS:

- System initialization table (DFHSIT)
- Destination control table (DFHDCT)
- ShutDown Program Load Table (DFHPLTSD)
- Startup JCL
- System definition data set

You can define, assemble, and link-edit more than one version of a table, and you can use a suffix to distinguish them. Start with the version of the definition that you want for the HLAPI/CICS definition and choose a new suffix to apply to all the tables that you have to modify for this installation. In the examples, the value chosen is **im** for illustration purposes only. You can choose any suffix.

Customizing the System Initialization Table (DFHSIT)

You can customize the **DFHSIT** in three ways:

- Add parameters to CICS startup procedure.
- At run time, use the **DFHSIT** startup override option.
- Alter the source code of the **DFHSIT** and reassemble.

Startup Procedure Parameter Changes

You can modify the parameter string on the **EXEC** statement of the JCL for the CICS program (**DFHSIP**). Refer to the chapter about processing system initialization parameters in the *CICS/ESA System Definition Guide*.

Using Overrides

To use the **DFHSIT** startup overrides, define a **SYSIN** data set in the startup JCL. You can use these data sets to select the correct **DFHSIT** suffix for your purposes. Add **ICS=YES**, **PLTSD=IM**, and **DCT=IM** to these overrides, if they are not already there.

Altering the DFHSIT Source Code

To alter the **DFHSIT** source, do the following tasks:

1. Locate the present source statements for **DFHSIT**, and copy them to another member name. The examples in this book use the name **DFHSIT***im*.
2. Edit the **DFHSIT***im* file. Insert or modify the table (do not forget the nonblank character in column 72):

```
SUFFIX=im,                                                  X
ICS=YES,                                                    X
PLTSD=im,                                                   X
DCT=im,                                                     X
INTTR=ON,                                                   X
USERTR=ON,                                                  X
```

3. Ensure that the value specified in the **DFLTUSER** parameter is a valid user ID on the system where the Tivoli Information Management for z/OS server resides and is properly authorized.
4. Assemble and link-edit the file using JCL as shown in the following figure.

```
//jobname  JOB  your-job-card
//ASMTAB   EXEC PROC=DFHAUPLE,NAME=SDFHAUTH
//******
//*   Make sure that the DSN= in the following statement addresses
//*    the data set that you modified.
//ASSEM.SYSUT1  DD  DSN=DFHSITim,DISP=SHR
```

Customizing the Destination Control Table (DFHDCT)

For message output from HLAPI/CICS, set up a destination. Add an entry to the **DCT** source code for destination **BLML** to receive all messages from the output of the HLAPI/CICS system. Perform the following steps to alter the **DCT** source:

1. Locate the current copy of the **DCT**.
2. Copy the current copy of the **DCT** to another member name, such as **DFHDCT***im*.
3. Edit **DFHDCT***im* and make the following insertions:

```
*
BLMLOG   DFHDCT TYPE=SDSCI,       FOR HLAPI/CICS APPLICATIONS          X
               BLKSIZE=136,                                            X
               BUFNO=1,                                                X
               DSCNAME=BLMLOG,                                         X
               RECFORM=VARUNB,                                         X
               RECSIZE=132,                                            X
               TYPEFLE=OUTPUT
*
BLML     DFHDCT TYPE=EXTRA,       DESTINATION USED BY HLAPI/CICS       X
               DESTID=BLML,                                            X
               DSCNAME=BLMLOG
```

4. Assemble and link-edit the file using JCL as shown in this example.

```
//jobname  JOB  your-job-card
//ASMTAB   EXEC PROC=DFHAUPLE,NAME=SDFHLOAD
//******
//*   Make sure that the DSN= in the following statement addresses
//*    the DCT data set that you modified.
//ASSEM.SYSUT1  DD  DSN=DFHDCTim,DISP=SHR
```

Customizing the Shut-Down Program Load Table (DFHPLT)

For initiating the HLAPI/CICS termination handler at CICS shutdown, add an entry to the **DFHPLTSD** source code for program **BLMYKTRM**.

1. Locate the current working copy of the **PLT** table.
2. Copy the current copy to a new member with the name **DFHPLT***im*.
3. Edit the new member and add the `DFHPLT ... PROGRAM=BLMYKTRM` statement before the `DFHPLT ....Program=DFHDELIM` statement. Put the entry with the other 'First Pass' programs as shown in the following example.

```
*
         DFHPLT TYPE=INITIAL,                                      X
               SUFFIX=im,                                          X
               STARTER=YES         ALLOWS $ IN SUFFIX
* FIRST PASS SHUTDOWN PROGRAMS
*        DFHPLT .........    existing or other entries
         DFHPLT TYPE=ENTRY,PROGRAM=BLMYKTRM   TERMINATION HANDLER
*        DFHPLT .........    existing or other entries
         DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
* SECOND PASS SHUTDOWN PROGRAMS
*
         DFHPLT TYPE=FINAL
```

4. Assemble and link-edit the file using JCL as shown in this example.

```
//jobname  JOB  your-job-card
//ASMTAB   EXEC PROC=DFHAUPLE,NAME=SDFHLOAD
//******
//*   Make sure that the DSN= in the following statement addresses
//*    the PLT data set that you modified.
//ASSEM.SYSUT1  DD  DSN=DFHPLTim,DISP=SHR
```

Customizing the Startup JCL

Include the data set name for HLAPI/CICS in the CICS startup JCL. Because a separate data set is used for the destination **BLML**, a DD record, such as the following, is required:

```
//BLMLOG DD DSN=data set name
         or
//BLMLOG DD SYSOUT=class
```

Ensure the startup JCL you use references the data set with the overrides that specify the version of the **DFHSIT** you want to use, and the other customization steps you performed in this section.

Customizing the CICS/ESA System Definition Data Set - JCL

You can modify the CICS system definition (**CSD**) data set by using JCL, or you can modify the data set online. The following example contains JCL that you can use to define the programs, transactions, connections, sessions, partner, and groups necessary to use HLAPI/CICS. It is a complete substitution for the **CEDA DEFINE** transactions defined in "Customizing the CICS/ESA Systems Definition Data Set - Online" on page 196. You can use either process alone, or parts from both processes. To use the example JCL, make the following changes:

1. Modify the **JOB** statement to your site's requirements.
   - Ensure that the **STEPLIB DD** statement's **DSNAME** parameter contains the **DFHCSDUP** member.
   - Verify that the **DFHCSD DD** statement references the **CSD** file used to start CICS.

```
//jobname  JOB   your jobcard
//* UPDATE THE CSD WITH THE VALUES FOR THE INFO CLIENT
//*
//CSDUP    EXEC PGM=DFHCSDUP,REGION=4M
//STEPLIB   DD DISP=SHR,DSN=CICS330.SDFHLOAD
//DFHCSD    DD DISP=SHR,DSN=CICS330.DFHCSD
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD *
```

2. Leave the **DEFINE PROGRAM** statements as specified, except to modify the group parameter to meet your site's requirements.

```
*
 DEFINE PROGRAM(BLMYKINF) GROUP(BLM610PP)
   LANGUAGE(ASSEMBLER) DATALOCATION(ANY)
   RES(YES)
*
 DEFINE PROGRAM(BLMYKCOM) GROUP(BLM610PP)
   LANGUAGE(ASSEMBLER) DATALOCATION(ANY)
   RES(YES)
*
 DEFINE PROGRAM(BLMYKTRM) GROUP(BLM610PP)
   LANGUAGE(ASSEMBLER) DATALOCATION(ANY)
   RES(YES)
```

3. Leave the **DEFINE TRANSACTION** statements as specified, except to modify the group parameter to meet your site's requirements.

```
*
 DEFINE TRANSACTION(BLMK) GROUP(BLM610PC)
   TASKDATALOC(ANY) SPURGE(YES)
   PROGRAM(BLMYKCOM)
```

```
*
 DEFINE TRANSACTION(BLMT) GROUP(BLM610PC)
   TASKDATALOC(ANY)
   PROGRAM(BLMYKTRM)
```

4. You can modify the **CONNECTION** parameter on both the **DEFINE CONNECTION** and the **DEFINE SESSION**, but they must both be modified to the same value.

```
*
 DEFINE CONNECTION(BLM1) GROUP(BLM610TC)
*
 DEFINE SESSION(BLM1) CONNECTION(BLM1) GROUP(BLM610TC)
```

5. The **DEFINE PARTNER** name must match the partner defined in the **CICS_Partner_ID PDB**, as described in “CICS_Partner_ID PDB” on page 210. Specify this same value in the **PARTNER** field in the sample program panel “HL01 - Starting the Session” on page 217. This name must match the LU name defined for the Tivoli Information Management for z/OS server. Set the **NETNAME**, **NETWORK** and **TPNAME** values to those appropriate for your site. Ask your local APPC administrator for the values.

```
*
*
 DEFINE PARTNER(BLM1PART) GROUP(BLM610TC)
     NETNAME(TBA)
     NETWORK(NETLAND)
     TPNAME(TBA)
```

6. Set the **ADD GROUP** statements to match what is correct for your site and what is coded in the **GROUP** parameters in this example.

```
*
*
 ADD GROUP(BLM610PP) LIST(BLM6LIST)
 ADD GROUP(BLM610PC) LIST(BLM6LIST)
 ADD GROUP(BLM610TC) LIST(BLM6LIST)
/*
```

Customizing the CICS/ESA Systems Definition Data Set - Online

The systems definition data set can also be updated for programs, transactions, connections, sessions, and partners, using resource definition online (RDO). The following sections explain the RDO process.

Program Entries

1. Start CICS by typing **CEDA DEFINE PROGRAM** as the transaction and pressing **Enter**.
2. Make the changes on the panel as indicated to define program **BLMYKINF** field values.

Field	Value
PROGram	==> BLMYKINF
Group	==> BLM610PP
Language	==> Assembler
RESident	==> Yes
DAtalocation	==> ANY

```
 DEF PROGRAM
 OVERTYPE TO MODIFY OR PRESS ENTER TO EXECUTE     CICS RELEASE = 0330
  CEDA  DEFine
   PROGram      ==> BLMKYINF
   Group        ==> BLM610PP
   DEscription  ==>
   Language     ==> Assembler          CObol | Assembler | Le370 | C |
                                       Pli   | Rpg
   RELoad       ==> No                 No | Yes
   RESident     ==> Yes                No | Yes
   USAge        ==> Normal             Normal | Transient
   USElpacopy   ==> No                 No | Yes
   Status       ==> Enabled            Enabled | Disabled
   RSl            : 00                 0-24 | Public
   Cedf         ==> Yes                Yes | No
   DAtalocation ==> ANY                Below | Any
   EXECKey      ==> User               User | Cics
  REMOTE ATTRIBUTES
   REMOTESystem ==>
+  REMOTEName   ==>
```

3. Press **Enter**.
4. Type **CEDA DEFINE PROGRAM** and press **Enter** again, this time to define the Communication Manager values.
5. Make the changes on the panel as indicated to define program **BLMYKCOM** field values.

Field	Value
PROGram	==> BLMYKCOM
Group	==> BLM610PP
Language	==> Assembler
RESident	==> Yes
DAtalocation	==> ANY

```
DEF PROGRAM
OVERTYPE TO MODIFY OR PRESS ENTER TO EXECUTE     CICS RELEASE = 0330
 CEDA  DEFine
  PROGram      ==> BLMKYCOM
  Group        ==> BLM610PP
  DEscription  ==>
  Language     ==> Assembler          CObol | Assembler | Le370 | C |
                                      Pli   | Rpg
  RELoad       ==> No                 No | Yes
  RESident     ==> Yes                No | Yes
  USAge        ==> Normal             Normal | Transient
  USElpacopy   ==> No                 No | Yes
  Status       ==> Enabled            Enabled | Disabled
  RSl            : 00                 0-24 | Public
  Cedf         ==> Yes                Yes | No
  DAtalocation ==> ANY                Below | Any
  EXECKey      ==> User               User | Cics
 REMOTE ATTRIBUTES
  REMOTESystem ==>
+ REMOTEName   ==>
```

6. Press **Enter**.
7. Type **CEDA DEFINE PROGRAM** and press **Enter** again, this time to define the termination handler values.
8. Make the changes on the panel as indicated to define program **BLMYKTRM** field values.

Field	Value
PROGram	==> BLMYKTRM
Group	==> BLM610PP
Language	==> Assembler
DAtalocation	==> ANY

```
DEF PROGRAM
OVERTYPE TO MODIFY OR PRESS ENTER TO EXECUTE     CICS RELEASE = 0330
 CEDA  DEFine
  PROGram      ==> BLMKYTRM
  Group        ==> BLM610PP
  DEscription  ==>
  Language     ==> Assembler          CObol | Assembler | Le370 | C |
                                      Pli   | Rpg
  RELoad       ==> No                 No | Yes
  RESident     ==> Yes                No | Yes
  USAge        ==> Normal             Normal | Transient
  USElpacopy   ==> No                 No | Yes
  Status       ==> Enabled            Enabled | Disabled
  RSl            : 00                 0-24 | Public
  Cedf         ==> Yes                Yes | No
  DAtalocation ==> ANY                Below | Any
  EXECKey      ==> User               User | Cics
 REMOTE ATTRIBUTES
  REMOTESystem ==>
+ REMOTEName   ==>
```

9. Press **Enter**.
10. Go on to the section on transaction entries.

Transaction Entries

Continuing from the **DEFINE PROGRAM** transaction perform the following steps to define the transactions:

1. Type **CEDA DEFINE TRANSACTION** as the transaction.
2. Define the transaction for HLAPI/CICS Communication Manager (**BLMK**). Make the changes on the panel as indicated to define transaction **BLMK** field values.

Field	Value
TRansaction	==> BLMK
Group	==> BLM610PC
PROGram	==> BLMYKCOM
TASKDATALoc	==> Any
SPurge	==> Yes

Note: The **BLMK** transaction is an internal transaction. Do not try to start it directly from a user terminal. Unpredictable results can occur.

```
 DEF TRAN
 OVERTYPE TO MODIFY OR PRESS ENTER TO EXECUTE     CICS RELEASE = 0330
  CEDA  DEFine
   TRansaction  ==> BLMK
   Group        ==> BLM610PC
   DEscription  ==>
   PROGram      ==> BLMYKCOM
   TWasize      ==> 00000             0-32767
   PROFile      ==> DFHCICST
   PArtitionset ==>
   STatus       ==> Enabled           Enabled | Disabled
   PRIMedsize     : 00000             0-65520
   TASKDATALoc  ==> Any               Below | Any
   TASKDATAKey  ==> User              User | Cics
  REMOTE ATTRIBUTES
   DYnamic      ==> No                No | Yes
   REMOTESystem ==>
   REMOTEName   ==>
   TRProf       ==>
+  Localq       ==>                   No | Yes
  S An object must be specified.
```

Note: You might want to use TClass to limit HLAPI/CICS. Use PF7 or PF10 to scroll backward, PF8 or PF11 to scroll forward.

```
 OVERTYPE TO MODIFY OR PRESS ENTER TO EXECUTE       CICS RELEASE = 0330
  CEDA  DEFine
+ SCHEDULING
   PRIOrity     ==> 001               0-255
   TClass       ==> No                No | 1-10
  ALIASES
   Alias        ==>
   TASKReq      ==>
   XTRanid      ==>
   TPName       ==>
                ==>
   XTPname      ==>
                ==>
                ==>
  RECOVERY
   DTimout      ==> No                No | 1-6800
   Indoubt      ==> Backout           Backout | Commit | Wait
   RESTart      ==> No                No | Yes
+  SPurge       ==> Yes               No | Yes
```

3. Press **Enter**.
4. Repeat the action for the Termination handler transaction. Make the changes on the panel as indicated to define transaction **BLMT** field values.

Field	Value
Transaction	==> BLMT
Group	==> BLM610PC
PROGram	==> BLMYKTRM

TASKDATALoc
==> Any

```
DEF TRAN
OVERTYPE TO MODIFY OR PRESS ENTER TO EXECUTE     CICS RELEASE = 0330
 CEDA  DEFine
  TRansaction  ==> BLMT
  Group        ==> BLM610PC
  DEscription  ==>
  PROGram      ==> BLMYKTRM
  TWasize      ==> 00000             0-32767
  PROFile      ==> DFHCICST
  PArtitionset ==>
  STatus       ==> Enabled           Enabled | Disabled
  PRIMedsize     : 00000             0-65520
  TASKDATALoc  ==> Any               Below | Any
  TASKDATAKey  ==> User              User | Cics
 REMOTE ATTRIBUTES
  DYnamic      ==> No                No | Yes
  REMOTESystem ==>
  REMOTEName   ==>
  TRProf       ==>
+ Localq       ==>                   No | Yes
 S An object must be specified.
```

5. Press **Enter**.
6. Go to the section about connection entries.

Connection Entries

Continuing from the previous transaction, perform the following steps to define the connections:

1. Type **CEDA DEFINE CONNECTION** as the transaction.
2. Define the transaction for HLAPI/CICS connections. Make the changes on the panel as indicated to define field values.

Field	**Value**
Connection	==> BLM1
Group	==> BLM610TC

```
 DEF CONNECTION
 OVERTYPE TO MODIFY                                CICS RELEASE = 0330
  CEDA  DEFine
   Connection   ==> BLM1
   Group        ==> BLM610TC
   DEscription  ==>
  CONNECTION IDENTIFIERS
   Netname      ==>
   INDsys       ==>
  REMOTE ATTRIBUTES
   REMOTESystem ==>
   REMOTEName   ==>
  CONNECTION PROPERTIES
   ACcessmethod ==> Vtam            Vtam | IRc | INdirect | Xm
   Protocol     ==> APPC            Appc | Lu61
   SInglesess   ==> No              No | Yes
   DAtastream   ==> User            User | 3270 | SCs | STrfield | Lms
   RECordformat ==> U               U | Vb
  OPERATIONAL PROPERTIES
+  AUtoconnect  ==> No              No | Yes | All
  S An object must be specified.
```

```
 DEF CONNECTION
 OVERTYPE TO MODIFY                                CICS RELEASE = 0330
  CEDA  DEFine
+ INService    ==> Yes          Yes | No
  SECURITY
   SEcurityname ==>
   ATtachsec    ==> Local       Local | Identify | Verify | Persistent
                                | Mixidpe
   BINDPassword ==>             PASSWORD NOT SPECIFIED
   BINDSecurity ==> No          No | Yes
```

3. Press **Enter**.
4. Go on to Session Entries.

Session Entries

1. Type **CEDA DEFINE SESSION** as the transaction.
2. Make the changes in the panels as indicated below.

Field	Value
Sessions	==> BLM1 (Name of session)
Group	BLM610TC (The name of the group used in the connection entry)
Connection	==> BLM1 (Name of connection)

MOdename Refer to CICS/ESA Version 3 Release 3 RDO under sessions modename option

Protocol ==> APPC

MAximum ==> (00,00)

Refer to CICS/ESA Version 3 Release 3 RDO under sessions maximum option.

SENDSize ==> Default to 4096 negotiated at CNOS time

RECEIVESize
==> Default to 4096 negotiated at CNOS time

```
DEF SESSION
OVERTYPE TO MODIFY OR PRESS ENTER TO EXECUTE      CICS RELEASE = 0330
 CEDA  DEFine
  Sessions     ==> BLM1
  Group        ==> BLM610TC
  DEscription  ==>
 SESSION IDENTIFIERS
  Connection   ==> BLM1
  SESSName     ==>
  NETnameq     ==>
  MOdename     ==>
 SESSION PROPERTIES
  Protocol     ==> Appc              Appc | Lu61
  MAximum      ==> 000 , 000         0-999
  RECEIVEPfx   ==>
  RECEIVECount ==>                   1-999
  SENDPfx      ==>
  SENDCount    ==>                   1-999
  SENDSize     ==> 4096              1-30720
+ RECEIVESize  ==> 4096              1-30720
 S An object must be specified.
```

```
 DEF SESSION
 OVERTYPE TO MODIFY OR PRESS ENTER TO EXECUTE      CICS RELEASE = 0330
  CEDA  DEFine
+  SESSPriority ==> 000               0-255
   Transaction    :
  OPERATOR DEFAULTS
   OPERId         :
   OPERPriority   : 000               0-255
   OPERRsl        : 0                                      0-24,...
   OPERSecurity   : 1                                      1-64,...
  PRESET SECURITY
   USERId       ==>
  OPERATIONAL PROPERTIES
   Autoconnect  ==> No                No | Yes | All
   INservice      :                   No | Yes
   Buildchain   ==> Yes               Yes | No
   USERArealen  ==> 000               0-255
   IOarealen    ==> 00000 , 00000     0-32767
   RELreq       ==> No                No | Yes
+  DIscreq      ==> No                No | Yes
  S An object must be specified.
```

```
 DEF SESSION
 OVERTYPE TO MODIFY                                 CICS RELEASE = 0330
  CEDA  DEFine
+  NEPclass     ==> 000          0-255
  RECOVERY
   RECOVOption  ==> Sysdefault   Sysdefault | Clearconv | Releasesess
                                 | Uncondrel | None
   RECOVNotify  ==> None         None | Message | Transaction
```

3. Press **Enter** to perform the transactions and go on to the section on Partner entries.

Partner Entries

Add a partner entry for each Tivoli Information Management for z/OS database system with which this CICS/ESA exchanges information.

1. Type **CEDA DEFINE PARTNER** as the transaction.
2. Make the changes in the panels as indicated below.

Field	Value
PARTNer	==> BLM1PART (Name of partner passed in **CICS_PARTNER_ID** PDB)
Group	==> BLM610TC (The name of the group used in connection)

NETName ==> Application ID of target system APPC/MVS (TBA in example)

NETWork ==> Netland

Tpname ==> The name assigned to the TP in the active APPC profile data set if the server is a RES. If the server is an MRES, the value given for **TPNAME** in the side information entry for the MRES in the active side information file on the MVS system where the server resides. (TBA in example)


```
DEF PARTNER
OVERTYPE TO MODIFY                                CICS RELEASE = 0330
 CEDA  DEFine
  PARTNer      ==> BLM1PART
  Group        ==> BLM610TC
  Description  ==> The side information used for CPI-C conversations
 REMOTE LU NAME
  NETName      ==> TBA
  NETWork      ==> Netland
 SESSION PROPERTIES
  Profile      ==> DFHCICSA
 REMOTE TP NAME
  Tpname       ==> TBA
               ==>
  Xtpname      ==>
               ==>
               ==>
```

3. Press **Enter** to perform the transactions.

Add the Groups to a List

Add the groups you defined into a single list. Perform the following steps.

1. Type **CEDA ADD GROUP** as the transaction.
2. Make the changes in the panels as indicated below, to add the program group to the list.

Field	Value
Group	==> BLM610PP
List	==> BLM6LIST

```
ADD GROUP
OVERTYPE TO MODIFY
 CEDA  ADd
  Group        ==>  BLM610PP
  List         ==>  BLM6LIST
  Before       ==>
  After        ==>
```

3. Press **Enter** to perform the transactions.
4. Type **CEDA ADD GROUP** again, this time to add the transaction group.
5. Make the changes in the panels as indicated below.

Field	**Value**
Group	==> BLM610PC
List	==> BLM6LIST

```
ADD GROUP
OVERTYPE TO MODIFY
 CEDA  ADd
  Group        ==>  BLM610PC
  List         ==>  BLM6LIST
  Before       ==>
  After        ==>
```

6. Press **Enter** to perform the transactions.
7. Type **CEDA ADD GROUP** again, this time to add the connection group.
8. Make the changes in the panels as indicated below.

Field	**Value**
Group	==> BLM610TC
List	==> BLM6LIST

```
ADD GROUP
OVERTYPE TO MODIFY
 CEDA  ADd
  Group        ==>  BLM610TC
  List         ==>  BLM6LIST
  Before       ==>
  After        ==>
```

9. Press **Enter** to perform the transactions.

This finishes online customization to define the CICS Base.

24

HLAPI/CICS Transaction Coding

To code CICS transactions that utilize the Tivoli Information Management for z/OS HLAPI/CICS client, you must consider the following:

- The calls to the HLAPI/CICS are made by means of the **EXEC CICS LINK** methodology, passing the address of the HICA as a parameter in the **COMMAREA**.
- Whether pseudoconversational or conversational mode transactions are being utilized, the HICA and any control or input PDBs must be placed in shared storage that is addressed by a value passed in the first four bytes of the **COMMAREA**.
- The input PDBs and control PDBs must reside in shared storage controlled by the client application transactions.
- The output PDBs, error PDBs, and message PDBs reside in shared storage controlled by HLAPI/CICS, and that storage should not be freed by the client application transactions.
- The return codes and reason codes generated by the HLAPI/CICS are returned in the HICA return code and reason code fields.
- The storage management rules above take precedence over the rules in the *Tivoli Information Management for z/OS Application Program Interface Guide*. The only time that the output, error, and message pointers should be changed is before a HL01 transaction. At that time, the pointers should all be set to 0.

Linking to the HLAPI/CICS

The HLAPI/CICS is called by a CICS link function. The call looks similar to the following COBOL fragment.

```
.
.
01 DFHCOMMAREA.
   03 HICA_ADDR       USAGE POINTER.
.
.
.
.
EXEC CICS LINK PROGRAM("BLMYKINF") COMMAREA(DFHCOMMAREA)
          LENGTH(64);
.
.
.
```

The **HICA_ADDR** must be an ordinary Tivoli Information Management for z/OS HICA with the self-identifying text contained in it. If the data passed as the parameter cannot be

validated as a HICA, then an abend occurs because there is no place to return the return code and reason codes. Thus, the client application need never process any return codes other than those in the HICA.

The PDBs are all addressed from the HICA, and the environment identifier is also contained in the HICA. Use the same HICA to make a series of calls into the same Tivoli Information Management for z/OS database. The transaction must allocate enough shared storage to contain the HICA and PDB storage.

Control PDBs for HLAPI Transactions

The HLAPI/CICS supports many of the transactions that you can normally use with the Tivoli Information Management for z/OS HLAPI. However, there are slight differences that exist between the two, concerning the use of PDBs. The HLAPI/CICS uses the normal HLAPI PDBs, plus some that the HLAPI alone does not use. This section describes the PDBs for the HL01 call from HLAPI/CICS.

The following control PDBs are used only in the CICS environment that uses the HLAPI/CICS functions. The additional PDBs supply information about the CICS UserID, which is used to identify instances of the client Communication Manager. The Partner ID is the name given the Tivoli Information Management for z/OS host in the CICS system's APPC tables. The other parameters are timeout values used to shutdown:

- Inactive sessions and the applications that started them but did not complete them
- The communications functions if the host has not responded in the stated amount of time.

CICS_User_ID PDB

This PDB supplies a name that identifies which Communication Manager instance this CICS transaction wants to use. The value you specify here must match one specified on the **DFLTUSER** parameter in the **DFHSIT** table. The value must be a valid user ID on the server system and properly authorized. Subsequent HL01 calls from the same CICS terminal to the same partner can be forced to a different Communication Manager instance by altering the value of this PDB. If the value is the same as in previous HL01 calls, the same Communication Manager instance and conversation is used.

This PDB is required and has no default value.

CICS_Partner_ID PDB

The Partner ID is the name of the server, which is the same value as the partner parameter of the CICS set session definition shown in “Partner Entries” on page 204. The partner ID is the name of the APPC node containing the Tivoli Information Management for z/OS server with the data that the client application actions address.

This PDB is required and has no default value.

CICS_CM_Time_Out_Value PDB

The **CICS_CM_Time_Out_Value** is the interval of time that the Communication Manager waits for the CICS transaction program to submit its next transaction. When the interval passes, all of the sessions that this instance of the Communication Manager is involved in are canceled. This prevents a client application from keeping a session and its resources unavailable to other client applications while he has suspended use of the terminal for

whatever reason. The value specified in the HL01 transaction that starts the Communication Manager is used; values for this interval that are found in subsequent HL01s are ignored.

The value for this PDB should not be set too low. A value less than several minutes might not be enough. When this interval passes, the client application must restart the entire transaction, thus wasting the time necessary to establish the session in the first place.

This timeout interval and the **CICS_Inter_Time_Out_Value** interval are independent of each other. This PDB is optional and has a default value of 99:59:59 (99 hours, 59 minutes, and 59 seconds).

CICS_Inter_Time_Out_Value PDB

The **CICS_Inter_Time_Out_Value** is the interval of time that the HLAPI/CICS interface waits for the Communication Manager to complete its task. If the Communication Manager does not respond within the time interval, the client interface assumes that the host has failed and is not available. All of the sessions that this instance of the Communication Manager is involved in are canceled, the Communication Manager transaction ends, and the CICS transaction is given a return and reason code that indicates that a timeout has occurred.

This PDB is optional and has a default value of 29:59:59 (29 hours, 59 minutes, and 59 seconds).

As a matter of processing, do not set these values too low. If the server is a RES, it is not unusual for an HL01 transaction to take a considerable amount of time as APPC sessions are established, the RES is started, and the subtask interface is started. If the server is an MRES, this time should be shorter.

25

Running the Sample CICS Application

The Tivoli Information Management for z/OS product tape includes some sample HLAPI/CICS programs for your use. After you perform the installation steps described in "Installing HLAPI/CICS and Customizing CICS/ESA" on page 191 you can access the samples. Note, however, that the sample application uses the Program Interface Data Table (PIDT) and Program Interface Pattern Table (PIPT) tables that are shipped with Tivoli Information Management for z/OS. This sample CICS application runs against the Tivoli Information Management for z/OS problem record **BLGYPRC**. If you change this record substantially from the way it is shipped, the sample application program might not run correctly.

Sample Programs

The HLAPI/CICS sample programs are coded in VS COBOL II. If your installation does not currently run VS COBOL II programs, ensure that your CICS startup JCL is modified to link to the VS COBOL II libraries at execution time.

BLMYKMNU
The sample menu program (tranid **BLMM**).

The program displays a map with standard selections. The user chooses one by entering a character in the selection field to the left of the description. When the enter key is pressed, the program displays the selected transaction panel (which may be overtyped) and sets the CICS transaction to process the panel to be called next. There is a check to prevent ending with Tivoli Information Management for z/OS logical sessions hanging.

BLMYKCTL
The sample control program.

This program serves a dual purpose. It processes both the HL01 and the HL02 panels.

After the HLAPI/CICS interface returns, the results are displayed on the terminal and the next transaction is set to the menu transaction.

- tranid BLM1 generates HICA and PDBs for HL01
- tranid BLM2 generates HICA and PDBs for HL02

BLMYKCRE
The sample create program (tranid **BLM8**).

This program processes the HL08 panel that builds data to do the create record transaction.

After the HLAPI/CICS interface returns, the results are displayed on the terminal and the next transaction is set to the menu transaction.

BLMYKDEL
The sample delete program (tranid **BLMD**).

This program processes the HL13 panel that builds data to do the delete record function.

After the HLAPI/CICS interface returns, the results are displayed on the terminal and the next transaction is set to the menu transaction.

BLMYKRTV
The sample retrieve program (tranid **BLM6**).

This program processes the HL06 panel that builds data to do the retrieve record transaction.

After the HLAPI/CICS interface returns, the results are displayed on the terminal as unformatted data, and the next transaction is set to the menu transaction.

BLMYKMAP
The sample BMS Maps, cataloged as **BLMMAPS**.

This is the source for the maps required to compile and run the sample programs.

Installing the Sample Programs

The following must be defined in CICS for the sample system.

1. Programs written in VS COBOL II
   - BLMYKMNU
   - BLMYKCTL
   - BLMYKCRE
   - BLMYKDEL
   - BLMYKRTV
2. Map set
   - BLMYKMAP
3. Transactions
   - BLMM
   - BLM1
   - BLM2
   - BLM6
   - BLM8
   - BLMD
   - BLME

Defining the Programs and Transactions to CICS

You can use the following sample job to define the sample programs and transactions to CICS. This presumes that the CSD has been updated for the **SESSION**, **CONNECTION** and **PARTNER** definitions. See "Customizing the CICS/ESA System Definition Data Set - JCL" on page 195 and "Customizing the CICS/ESA Systems Definition Data Set - Online" on page 196 for more information about these definitions. To modify the sample for your location, follow these steps:

1. Change the **JOB** Statement to your site's requirements.
2. Ensure that the **STEPLIB DD** statement's **DSNAME** parameter contains the data set that contains the **DFHCSDUP** program.
3. Verify that the **DFHCSD DD** statement addresses the **CSD** file that you use to start CICS.
4. Leave the **DEFINE PROGRAM** and **DEFINE TRANSACTION** statements as defined, except that you can change the **GROUP** parameter.

```
//jobname  JOB   your jobcard
//* UPDATE THE CSD
//*
//CSDUP    EXEC PGM=DFHCSDUP,REGION=4M
//STEPLIB   DD DISP=SHR,DSN=CICS330.SDFHLOAD
//DFHCSD    DD DISP=SHR,DSN=CICS330.DFHCSD
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD *
* DEFINE PROGRAM(BLMYKMNU) GROUP(BLM610SP)
   LANGUAGE(COBOL) DATALOCATION(ANY)
   RES(YES)
*
 DEFINE PROGRAM(BLMYKCTL) GROUP(BLM610SP)
   LANGUAGE(COBOL) DATALOCATION(ANY)
   RES(YES)
*
 DEFINE PROGRAM(BLMYKRTV) GROUP(BLM610SP)
   LANGUAGE(COBOL) DATALOCATION(ANY)
   RES(YES)
*
 DEFINE PROGRAM(BLMYKCRE) GROUP(BLM610SP)
   LANGUAGE(COBOL) DATALOCATION(ANY)
   RES(YES)
*
 DEFINE PROGRAM(BLMYKDEL) GROUP(BLM610SP)
   LANGUAGE(COBOL) DATALOCATION(ANY)
   RES(YES)
*
 DEFINE MAPSET(BLMMAPS) GROUP(BLM610SP)
   RES(YES)
*
 DEFINE TRANSACTION(BLMM) GROUP(BLM610ST)
   TASKDATALOC(ANY)   PROGRAM(BLMYKMNU)
*
 DEFINE TRANSACTION(BLM1) GROUP(BLM610ST)
   TASKDATALOC(ANY)   PROGRAM(BLMYKCTL)
*
 DEFINE TRANSACTION(BLM2) GROUP(BLM610ST)
   TASKDATALOC(ANY)   PROGRAM(BLMYKCTL)
*
 DEFINE TRANSACTION(BLM6) GROUP(BLM610ST)
   TASKDATALOC(ANY)   PROGRAM(BLMYKRTV)
*
 DEFINE TRANSACTION(BLM8) GROUP(BLM610ST)
   TASKDATALOC(ANY)   PROGRAM(BLMYKCRE)
```

```
*
 DEFINE TRANSACTION(BLMD) GROUP(BLM610ST)
   TASKDATALOC(ANY)   PROGRAM(BLMYKDEL)
*
 DEFINE TRANSACTION(BLME) GROUP(BLM610ST)
   TASKDATALOC(ANY)   PROGRAM(BLMYKMNU)
*
 ADD GROUP(BLM610SP) LIST(BLM6LIST)
 ADD GROUP(BLM610ST) LIST(BLM6LIST)
/*
```

Starting the Sample Application

The following illustrates the process for starting the sample application. When Figure 13 appears, the sample program can be started.

Figure 13. CICS/ESA Logo Panel

Entering the BLMM Transaction

Clear the CICS/ESA logo screen and enter the transaction code for the sample transaction, **BLMM**, to start the HLAPI/CICS client program and display the screen shown in Figure 14 on page 217.

```
   CHOOSE AN OPTION BY PLACING ANY CHARACTER BEFORE IT

           START LOGICAL CONVERSATION HL01 TRANSACTION

           CREATE RECORD HL08 TRANSACTION

           RETRIEVE RECORD HL06 TRANSACTION

           DELETE RECORD HL13 TRANSACTION

           END LOGICAL SESSION HL02 TRANSACTION
           AN HL01 WILL BE REQUIRED TO RESTART.

           END SAMPLE APPLICATION

  CURRENT ENVIRONMENT  00000000

  RETURN CODE  00000000 REASON CODE  00000000


PRESS ENTER TO PROCESS, CLEAR TO ABORT
```

Figure 14. HLAPI/CICS Sample Program - Main Menu

The **Current Environment** field on the menu contains a value that identifies the connection with the Tivoli Information Management for z/OS server. This value is copied from the environment field (**HICAENVP**). The **return code** and **reason code** fields on the menu reflect the return code (**HICARETC**) and reason code (**HICAREAS**) in the HICA used with the host.

If you place a nonblank character in the line that has the menu item to be processed, that item is selected for processing. If more than one is selected, the first item on the menu is chosen.

HL01 - Starting the Session

When you select **Start Logical Conversation HL01 Transaction**, the following panel appears. The values represent pieces of information necessary to the HLAPI conversation with the Tivoli Information Management for z/OS server. Each item is a control PDB to be passed to the server. The last four PDBs are unique to the CICS environment. Of these four PDBs, the **User ID** and **Partner** names are required, and the **Interface Timeout** and **Communication Manager Timeout** are optional, both having the default values as shown.

```
 CONTROL

TRANSACTION ID   HL01

APPLICATION ID   IBMUSER

DEFAULT OPTION   NONE

PRIVILEGE CLASS  MASTER

SESSION MEMBER   BLGSES00

USER ID          IBMUSER

PARTNER          INFOPTNR

INTERFACE TIMEOUT HHMMSS 002959

COM MGR TIMEOUT   HHMMSS 095050

PRESS ENTER TO PROCESS, CLEAR TO ABORT
```

Figure 15. HL01 Input Panel with Default Settings

Modifying the HL01 Panel

You can change a series of defaults on the HL01 input screen. The **Application ID, User ID**, and **Partner** field values should be changed to values appropriate for your installation. The other values can be changed to match your specific Tivoli Information Management for z/OS database setup, with the exception of the **Transaction ID**. Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* for more information. The last four control PDBs are unique to the CICS client.

```
 CONTROL

TRANSACTION ID   HL01

APPLICATION ID   hanover

DEFAULT OPTION   NONE

PRIVILEGE CLASS  MASTER

SESSION MEMBER   BLGSES00

USER ID          hanover

PARTNER          blm1part

INTERFACE TIMEOUT HHMMSS 002959

COM MGR TIMEOUT   HHMMSS 095050

PRESS ENTER TO PROCESS, CLEAR TO ABORT
```

Figure 16. HL01 Input Panel Modified for Local Variables

The HL01 transaction initializes the Tivoli Information Management for z/OS transaction, the APPC session on the CICS machine, the connection to the Tivoli Information Management for z/OS server, and the Tivoli Information Management for z/OS session. If the server is a RES, an address space is started for the conversation. If the server is an MRES, the request goes to a pre-started address space where it is assigned to the first available Client Communication Processor. It takes time to perform all these functions, but only at startup. After Communication Manager starts, it reuses what it can without destroying the audit trail. The output of the transaction is updated **Current Environment** and **Return Code** and **Reason Code** fields on the main menu.

HL01 Output - Main Menu

At this point, the conversation with the host is established and is available to run the transaction types.

```
       CHOOSE AN OPTION BY PLACING ANY CHARACTER BEFORE IT

               START LOGICAL CONVERSATION HL01 TRANSACTION

               CREATE RECORD HL08 TRANSACTION

               RETRIEVE RECORD HL06 TRANSACTION

               DELETE RECORD HL13 TRANSACTION

               END LOGICAL SESSION HL02 TRANSACTION
               AN HL01 WILL BE REQUIRED TO RESTART.

               END SAMPLE APPLICATION

      CURRENT ENVIRONMENT  00000001

      RETURN CODE  00000000 REASON CODE  00000000


  PRESS ENTER TO PROCESS, CLEAR TO ABORT
```

Figure 17. HL01 Output Panel

HL08 - Creating a Record

Select **Create Record HL08 Transaction** from the main menu to view the panel shown in Figure 18 on page 220. Press Enter to add the data to the host Tivoli Information Management for z/OS database.

The input PDBs in the following example include the internal symbol values.

```
  CONTROL:
 TRANSACTION ID   HL08
 DEFAULT OPTION   NONE
 PIDT NAME        BLGYPRC
 SEPARATOR CHAR   ,
 PRIVILEGE_CLASS  MASTER
 ALIAS_TABLE


  INPUT:
 S0CCF            SAMPLE01
 S0E0F            THIS IS THE DESCRIPTION OF SAMPLE #01
 S0B59            IBMUSER
 S0B9B            T50
 S0BEE            INITIAL


 ENVIRONMENT ID   00000001


PRESS ENTER TO PROCESS, CLEAR TO ABORT
```

Figure 18. HL08 Input Panel

HL08 Output

The output from the HL08 transaction is a panel with the output and message PDB information, and the return code and reason code values.

```
     DISPLAYING THE OUTPUT OF INFORMATION/MANAGEMENT
SAMPLE01
 BLG03058I Record SAMPLE01 was stored successfully.

PRESS ENTER TO CONTINUE                RC= 00000000 REASON= 00000000
HL08 WAS SUCCESSFUL
```

Figure 19. HL08 Output Panel

HL06 - Retrieving a Record

When you select **Retrieve Record HL06 Transaction** from the main menu to retrieve a record, the input panel shown in Figure 20 appears. The default values retrieve the sample record created in “HL08 - Creating a Record” on page 219.

```
     CONTROL:
    TRANSACTION ID   HL06
    RNID SYMBOL      SAMPLE01
    PIDT NAME        BLGYPRR

    ENVIRONMENT ID   00000001








PRESS ENTER TO PROCESS, CLEAR TO ABORT
```

Figure 20. HL06 Input Panel

HL06 Output

The output of the retrieve transaction displays just the actual data at one line per field, without the field names.

```
        DISPLAYING THE OUTPUT OF INFORMATION/MANAGEMENT
,
RECS=PROBLEM
IBMUSER
T50
INITIAL
THIS IS THE DESCRIPTION OF SAMPLE #01
MASTER
10/17/97
14:42
10/17/97
14:42
HANOVER




PRESS ENTER TO CONTINUE                 RC= 00000000 REASON= 00000000
HL06 WAS SUCCESS
```

Figure 21. HL06 Output Panel

HL13 - Deleting a Tivoli Information Management for z/OS Record

You can delete the data record created and retrieved previously. The default values provide the information required to delete the record from the Tivoli Information Management for z/OS database if you did not alter the create data.

HL13 Input

```
     CONTROL:
    TRANSACTION ID   HL13
    RNID SYMBOL      SAMPLE01


    ENVIRONMENT ID   00000001















PRESS ENTER TO PROCESS, CLEAR TO ABORT
```

Figure 22. HL13 Input Panel

HL13 Output

```
      DISPLAYING THE OUTPUT OF INFORMATION/MANAGEMENT
HL13
SAMPLE01
 BLG03034I The specified record SAMPLE01 was successfully deleted.
















PRESS ENTER TO CONTINUE                RC= 00000000 REASON= 00000000
HL13 WAS SUCCESS
```

Figure 23. HL13 Output Panel

HL02 - Ending the Logical Session

Selecting the **End Logical Session HL02 Transaction** closes the logical session associated with the environment and the CICS client using that environment. However, if there are other environments using the same APPC session, then an APPC use count is decremented and just the environment in question is closed. When the use count is zero, the conversation between the server and the client is closed.

HL02 Input

```
     CONTROL:
    TRANSACTION ID   HL02

    ENVIRONMENT ID   00000001


    THIS WILL END THE LOGICAL SESSION
    AN HL01 WILL BE REQUIRED TO RESTART











PRESS ENTER TO PROCESS, CLEAR TO ABORT
```

Figure 24. HL02 Input Panel

When the HL02 transaction finishes, the menu panel displays the new **Current Environment** value, now zero, and the **Return Code** and **Reason Code** values.

Ending the Sample Application

This menu choice closes the sample program. Type a nonblank character next to the **End Sample Application** field.

```
      CHOOSE AN OPTION BY PLACING ANY CHARACTER BEFORE IT

              START LOGICAL CONVERSATION HL01 TRANSACTION

              CREATE RECORD HL08 TRANSACTION

              RETRIEVE RECORD HL06 TRANSACTION

              DELETE RECORD HL13 TRANSACTION

              END LOGICAL SESSION HL02 TRANSACTION
              AN HL01 WILL BE REQUIRED TO RESTART.

            X END SAMPLE APPLICATION

     CURRENT ENVIRONMENT  00000000

     RETURN CODE  00000000 REASON CODE  00000000

 PRESS ENTER TO PROCESS, CLEAR TO ABORT
 LOGICAL SESSION HAS ENDED SUCCESSFULLY
```

Figure 25. End Sample Application Menu Choice

Sample Closing Screen

```
                        CHOOSE TO END TASK




          THANKS FOR USING THE INFORMATION/MANAGEMENT SAMPLES
```

Running Multiple Environments

You can run multiple environments at the same time using the sample program by doing the following:

1. Before the HL01 for the second environment, reset the current environment field on the main menu to all zeros.
2. Change the environment field on the main menu when you want to operate on a different environment.

26

Introduction to HLAPI/UNIX

Tivoli Information Management for z/OS supports remote access from an application program that runs on AIX, HP-UX, and Sun Solaris. It does this through the High-Level Application Program Interface (HLAPI) and one of the Tivoli Information Management for z/OS HLAPI clients for UNIX (HLAPI/UNIX). The HLAPI/UNIX clients are:

- HLAPI/AIX
- HLAPI/HP
- HLAPI/Solaris.

Throughout the chapters dealing with these HLAPI/UNIX clients, the information and instructions pertain to all three clients unless otherwise noted. Where there is a difference, it is noted.

The HLAPI/UNIX provides remote access to Tivoli Information Management for z/OS data and data manipulation services. Each one consists of three parts:

- A Tivoli Information Management for z/OS server, an MVS-based transaction program that resides on the MVS host system. It provides the link between Tivoli Information Management for z/OS and the HLAPI/UNIX system. These servers are the RES (HLAPI/AIX), MRES with APPC (HLAPI/AIX), and MRES with TCP/IP (HLAPI/AIX, HLAPI/HP, and HLAPI/Solaris).
- The Tivoli Information Management for z/OS HLAPI/UNIX requester (requester), a UNIX-based transaction program that provides access to the HLAPI through a Tivoli Information Management for z/OS server.
- The Tivoli Information Management for z/OS HLAPI/UNIX client interface (client interface), a UNIX-based shared library and bindings for the C language. HLAPI/AIX also provides a REXX HLAPI/AIX feature which provides access to the HLAPI/AIX client interface from AIX REXX/6000 programs.

Like the HLAPI, the HLAPI/UNIX is a transaction-based application programming interface. User application programs interact with Tivoli Information Management for z/OS from the UNIX environment in basically the same way as they do from MVS using the HLAPI. These remote environment user application programs can be thought of as the remote *clients* to the Tivoli Information Management for z/OS *server*. The remote environment offers a subset of HLAPI transactions, which are listed in Table 1 on page 3; the *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information.

The HLAPI/UNIX enables application programmers to write applications for use in their specific work environment. The task described in “A Typical Scenario” on page 226 is typical of the problems that can be solved using HLAPI/UNIX.

A Typical Scenario

Suppose an application programming group in an enterprise has written two AIX-based help desk applications that interact with Tivoli Information Management for z/OS through HLAPI/AIX. One is a problem management database application, and the other is a configuration management database application. The databases reside on two MVS systems. For efficiency, each help desk operator maintains two user IDs on each MVS system with privilege classes as follows:

- Basic privilege class authority for queries sent through the configuration management application
- A higher privilege class authority for creating records through the problem management application.

The AIX system administrator has already provided the help desk operators with the information necessary to install and start both application programs successfully.

1. The AIX administrator for the host RS/6000® machine starts the HLAPI/AIX requester on the RS/6000 network.
2. A help desk operator logs on to another RS/6000 machine in the same network as the one that is running the requester. The operator's **.xinitrc** file starts the problem management and the configuration management application programs.
3. When a problem call arrives, the operator uses the problem management application to enter preliminary information and open a problem record in Tivoli Information Management for z/OS through the HLAPI/AIX.
4. In another window on the same machine, the operator uses the configuration management application to query Tivoli Information Management for z/OS through HLAPI/AIX for information about the configuration of the caller.
5. Meanwhile, Tivoli Information Management for z/OS returns a problem record number through HLAPI/AIX, and the operator gives the number to the caller and promises a response to the problem report.
6. By this time, Tivoli Information Management for z/OS has returned results of the configuration query through HLAPI/AIX. The operator investigates the problem and updates the problem record if necessary.

The same Tivoli Information Management for z/OS functions that once required direct MVS access are now performed on a RS/6000 machine.

The remaining sections of this chapter help you understand the interactions of the HLAPI/UNIX and Tivoli Information Management for z/OS.

Server Overview

A Tivoli Information Management for z/OS server is an MVS/ESA transaction program that handles communication between a HLAPI/UNIX requester and a Tivoli Information Management for z/OS database that resides on the MVS system where the server is installed. A UNIX client application program must use a HLAPI/UNIX client interface to access a server through a HLAPI/UNIX requester. Any UNIX client interface can access any UNIX requester. For example, an AIX client interface can access an HP requester. Only the AIX requester, however, provides APPC communication support to a RES or MRES with APPC. So although the client application chooses the server, the requester supporting that client

application must provide the appropriate communications support for the server. The client application should use a requester that supports the protocol for the server it has chosen.

A server must be installed and available on every MVS/ESA machine with a Tivoli Information Management for z/OS database that HLAPI/UNIX needs to access. See "Configuring and Running a Remote Environment Server (RES)" on page 25, "Configuring and Running a Multiclient Remote Environment Server (MRES) with APPC" on page 35, and "Configuring and Running a Multiclient Remote Environment Server (MRES) with TCP/IP" on page 53 for information about installing the servers.

Specific to AIX:

When deciding which server, RES, MRES with APPC, or MRES with TCP/IP, to use, consider the communication protocol each one supports. Also consider the security requirements of your application and how you will implement them.

Requester Overview

The requester receives information from the client application program through the client interface and transfers the information to the appropriate server. It receives information from the server and transfers the information back to the client application program through the client interface. A requester can communicate with multiple servers and multiple client interfaces. In addition, any HLAPI/UNIX requester can communicate with any HLAPI/UNIX client interface. However, only an AIX requester can use APPC communications to access a server (RES or MRES with APPC).

A HLAPI/UNIX requester includes the following components:

- A daemon program that runs on a requester host to serve as a communication link between a server on MVS and a client interface on UNIX.

  The communication link from the HLAPI/AIX requester to the server can use the APPC and TCP/IP communication protocols. The communication link from the HLAPI/HP or HLAPI/Solaris requester to the server must use the TCP/IP communication protocol.

  A requester and a client interface communicate using TCP/IP sockets. Each requester can communicate with multiple client interfaces on multiple hosts, and each client interface can communicate with multiple requesters on multiple requester hosts. The requester need not be located on the same requester host as a client interface that uses the requester.
- An optional system profile that specifies parameters affecting the requester's function.

You can run more than one requester on a single UNIX host. You can have AIX requesters communicating with a RES, an MRES with APPC, and an MRES with TCP/IP all on a single RS/6000 machine. You can have one or more than one HP requester communicating with different MRESs with TCP/IP on a single HP Series 800 machine. You can direct UNIX to start a HLAPI/UNIX requester when a client interface attempts to establish the initial contact with the requester. This eliminates the need to start a requester before starting a client application program. "Starting the Requester Automatically" on page 263 explains how to configure the requester so that it starts automatically.

Client Interface Overview

The HLAPI/UNIX client interface transfers information from the client application program to the requester. It also transfers information it receives from the requester back to the client application program. Any of the HLAPI/UNIX client interfaces can communicate with any of the HLAPI/UNIX requesters. However, only an AIX requester can use APPC communications to access a server (RES or MRES with APPC).

The HLAPI/UNIX client interfaces use C programming language bindings. A client interface includes the following components:

- A C-language header file, **idbh.h**, which must be included by applications that use HLAPI/UNIX services. This file contains the declarations for the functions and data structures that application programs must use to communicate with Tivoli Information Management for z/OS through HLAPI/UNIX.
- An optional C-language header file, **idbech.h**, which contains named constants representing the HLAPI, LLAPI, and HLAPI/UNIX return and reason codes used by HLAPI/UNIX.
- A shared runtime library, which application programs using HLAPI/UNIX services link to dynamically.

  Specific to AIX:
  `libidb.a`

  Specific to HP:
  `libidb.sl`

  Specific to Solaris:
  `libidb.so`

  This library contains the entry points and executable code for the client interface functions.
- One or more database profiles, each of which specifies parameters that apply to entire sequences of transactions.

A client interface communicates with a requester using TCP/IP protocol. A client interface can communicate with multiple requesters.

HLAPI/AIX also provides a REXX HLAPI/AIX interface. The REXX HLAPI/AIX interface enables you to access HLAPI/AIX functions from AIX REXX/6000 programs, in the same manner as HLAPI/REXX on MVS enables you to access HLAPI functions from MVS REXX programs. See “Using the REXX HLAPI/AIX Interface” on page 283 for more information about REXX HLAPI/AIX.

Communication Overview

The client chooses the communication protocol that the requester uses to communicate with the server on its behalf. The communication protocol is used for the entire transaction sequence submitted by the client.

An AIX requester and a Tivoli Information Management for z/OS server can communicate using APPC or TCP/IP protocols so an AIX requester can use a RES, MRES with APPC, or MRES with TCP/IP. An HP or Solaris requester and a Tivoli Information Management for z/OS server can communicate using TCP/IP protocol so they can use only an MRES with

TCP/IP. Each requester can communicate with multiple servers on multiple MVS hosts, and each server can communicate with multiple requesters on multiple requester hosts. The HLAPI/UNIX application developer chooses the communication protocol in the database profile on the client host. The **IDBSYMDESTNAME** database profile keyword indicates that the requester is to establish an APPC conversation on behalf of the UNIX client. The **IDBSERVERHOST** database profile keyword indicates the client wants a TCP/IP connection.

A requester and a client interface communicate using sockets. Each requester can communicate with multiple client interfaces on multiple hosts, and each client interface can communicate with multiple requesters on multiple hosts.

A requester need not be located on the same host as a client interface using that requester.

Basic Transaction Flow

A *transaction sequence* is a series of HLAPI/UNIX transactions that begins with an initialize Tivoli Information Management for z/OS transaction (HL01), followed by other supported transactions in any order and ends with a terminate Tivoli Information Management for z/OS transaction (HL02). Client application programs submit transactions in a transaction sequence, which is referred to as a logical session.

Each HLAPI/UNIX transaction request travels from a client application program on UNIX to Tivoli Information Management for z/OS on MVS along the route shown in Figure 26. This example illustrates AIX with a RES. The path would be similar for an AIX MRES with APPC or a UNIX MRES with TCP/IP. With an MRES with TCP/IP, of course, the communication protocol would be TCP/IP instead of APPC.

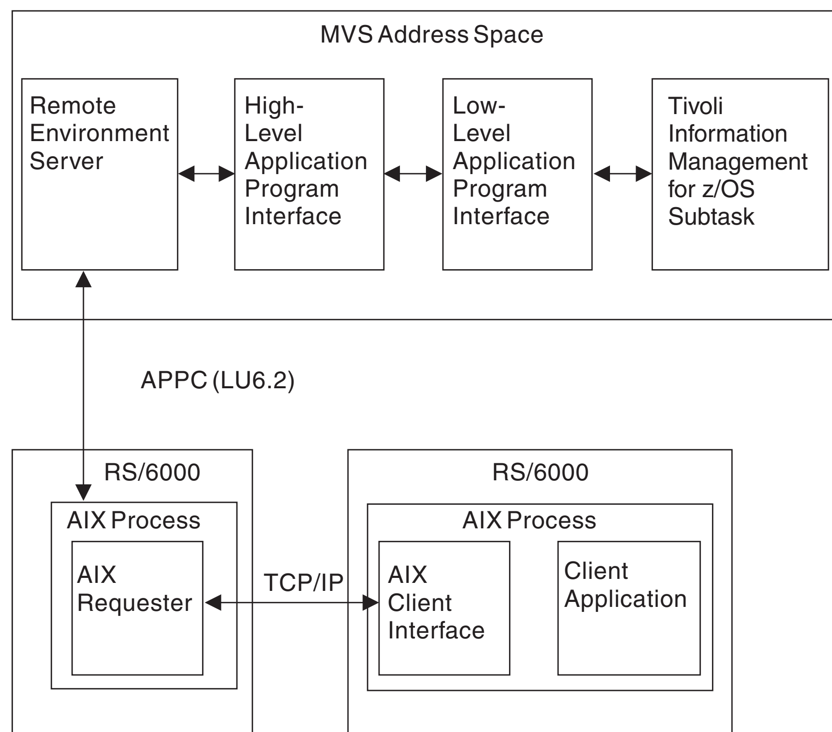


Figure 26. HLAPI/AIX Transaction Flow

The transaction reply travels from Tivoli Information Management for z/OS to the client application along the same route in reverse.

The following steps describe the events that occur when a HLAPI/UNIX transaction is processed. For simplicity, assume that the transaction is synchronous and that it is neither the HL01 transaction nor the HL02 transaction of the transaction sequence.

On the UNIX host running the client application program

A client application program initiates a transaction by calling the HLAPI/UNIX function IDBTransactionSubmit(). One of the arguments passed to this function is a HICA structure and its control PDBs and input PDBs. Together, these structures represent a transaction request. The HLAPI/UNIX client interface frees any output, error, and message PDBs it allocated for the client application program during the previous transaction in the sequence. The client interface translates the HICA and PDBs from the specified client code set to the specified server code set. The client interface converts the data in the HICA and PDBs from data structure format to byte stream format, then transmits the data via TCP/IP sockets to the requester identified in the database profile specified in a control PDB.

On the UNIX host running the requester

The HLAPI/UNIX requester forwards the transaction request to the server on MVS over the communication link established during the HL01 transaction that initiated the transaction sequence. For more information about communication links between requesters and servers, see "Defining the Client Interface to Requester Communication Link" on page 246.

On the MVS host running the server and Tivoli Information Management for z/OS

When the transaction request arrives at the MVS host, the server submits the request to the Tivoli Information Management for z/OS HLAPI. After the requested HLAPI transaction finishes, the server transmits the HICA, the output, error, and message PDBs, and the **PDBCODE** field of the input PDBs to the requester.

On the UNIX host running the requester

The requester forwards the transaction reply to the HLAPI/UNIX client interface that submitted the transaction.

On the UNIX host running the client application program

The HLAPI/UNIX client interface translates the HICA and PDBs from the specified server code set to the specified client code set. The order of the PDBs in each chain is maintained from the server to the client interface. The client interface updates the HICA and PDBs of the client application program with data received from the transaction. The transaction is complete, and the client application program returns from the call to **IDBTransactionSubmit()**.

The flow of an asynchronous transaction is similar to the flow of a synchronous transaction. In synchronous processing, the client application program returns from **IDBTransactionSubmit()** after the entire transaction is complete. In asynchronous processing, the client application program returns from **IDBTransactionSubmit()** after the transaction is initiated, and the client application program must call **IDBTransactionStatus()** to retrieve the transaction reply.

HLAPI/UNIX Configuration Considerations

The following information is helpful in configuring HLAPI/UNIX.

A UNIX client application program can manage multiple concurrent transaction sequences through HLAPI/UNIX. Each HICA is associated with a specific transaction sequence. Each transaction sequence is associated with a Tivoli Information Management for z/OS logical session on MVS. Therefore, there is a one-to-one correspondence between HICAs and Tivoli Information Management for z/OS logical sessions.

A client application program specifies the parameters listed below when it submits an HL01 transaction. Each of these parameters remains in effect for the duration of a transaction sequence. If conversation sharing (described in "IDBSHARECMS" on page 253) is enabled, the requester can assign multiple transaction sequences to the communication connection if each transaction sequence uses the same values for the following parameters:

- Requester host and service name (**IDBREQUESTERHOST** and **IDBREQUESTERSERVICE**)

  The requester host and the requester service name are parameters in the database profile. Together, they identify the requester host and TCP/IP service name or alias, which the client interface uses to establish communication with the requester.
- For APPC: Symbolic destination name of the server (**IDBSYMDESTNAME**)

  The server symbolic destination name is a parameter in the database profile. The symbolic destination name specifies an APPC side information entry that the requester uses to establish an APPC conversation with the server on MVS.
- For TCP/IP: **IDBSERVERHOST** and **IDBSERVERSERVICE**

  The **IDBSERVERHOST** and **IDBSERVERSERVICE** keywords are specified in the database profile. Together, they identify the MVS host and the MRES with TCP/IP service name that the requester uses to establish a TCP/IP conversation with the server on MVS.
- User ID for the MVS logon

  The MVS user ID is specified in the **SECURITY_ID** control PDB for the HL01 transaction.
- User password for the MVS logon

  The MVS password is specified on the **PASSWORD** control PDB for the HL01 transaction.

The requester assigns a new transaction sequence to an existing conversation if the following conditions apply when the HL01 transaction is processed:

- Conversation sharing is enabled.
- The parameters of the conversation match the parameters specified by the client application program for the transaction sequence (as just identified).
- Fewer than 10 transaction sequences are assigned to the conversation.

When the server is a RES, each APPC conversation between an AIX requester and the server corresponds to an address space on the MVS host. The correspondence between conversations and address spaces is one-to-one. Each address space manages one or more

Tivoli Information Management for z/OS logical sessions. The logical sessions are in a one-to-one correspondence with the transaction sequences sharing the APPC conversation.

A server can perform transactions against multiple Tivoli Information Management for z/OS databases within the same BLX-SP on the server's MVS host. However, a logical session can perform transactions against only one Tivoli Information Management for z/OS database. When a server receives a valid HL01 transaction request, the server establishes a Tivoli Information Management for z/OS logical session with the database specified in a control PDB. The association between the logical session and the database lasts until the transaction sequence ends.

Multiple transaction sequences can be routed through a single APPC conversation and its associated Tivoli Information Management for z/OS logical session. Each session processes transactions serially on a first-in, first-out (FIFO) basis. A transaction sequence cannot have more than one transaction pending. All previously submitted transactions must be complete before a client application program can submit another transaction in the same transaction sequence. This also applies to conversations established using MRES with APPC or MRES with TCP/IP.

Resources Needed for the Client Interface

When you write a client application program, consider that the HLAPI/UNIX client interface uses the following types of resources:

- Processes

  A client interface requires that the process ID and the effective user ID of a calling process remain constant from call to call for a transaction sequence. This allows HLAPI/UNIX to reaccess and release resources without compromising resources or data on the UNIX system.

- Sockets

  A client interface uses sockets from two address families: **AF_UNIX** and **AF_INET**. A client interface uses fewer than four sockets to process an individual transaction. Of these sockets, at most two are **AF_UNIX** sockets; at most three are **AF_INET** sockets.

  The descriptor table of any process calling HLAPI/UNIX services must allow enough entries for the sockets and files used by the client interface.

- Memory

  The amount of memory that a client interface uses to process a transaction is difficult to predict. The type of transactions that the client application calls and the database contents determine the memory requirements of the client interface. The client application program is responsible for allocating and freeing memory used for the HICA, and the input and control PDBs. The client interface allocates and frees memory used for output, error, and message PDBs, and other HLAPI/UNIX structures. In general, search transactions are more memory-intensive than other transactions.

- Shared memory segments

  A client interface creates and attaches a shared memory segment during the processing of each transaction. The shared memory segment is detached normally before the process calling HLAPI/UNIX services returns from the call. The shared memory segment is deleted normally before the calling process returns with the results of the transaction.

A process calling HLAPI/UNIX services must allow for the attachment of one shared memory segment during the call.

Specific to AIX:
With AIX, a process cannot have more than 16 segments attached at any time.

Specific to HP:
With HP-UX, a shared memory segment size of 8 388 608 is required.

Specific to Solaris:
With Sun Solaris, a shared memory segment size of 8 388 608 is required.

- Files

  A client interface accesses the following UNIX system files during transaction processing:
  - /etc/services
  - /etc/hosts
  - /etc/utmp

  A client interface also accesses the following HLAPI/UNIX files during transaction processing. The client interface does not delete these files except when replacing an old archived file with a new archived file:
  - Database profile
  - Active trace log file
  - Archived trace log file
  - Probe log file (**idbprobe.log**)

  Additional files created during installation of a client interface are listed in "Components of HLAPI/AIX" on page 348, "Components of HLAPI/HP" on page 349, and "Components of HLAPI/Solaris" on page 350. Most of these files, with the exception of the files in the directory ***/idbhlapi/examples**, are used during transaction processing (where ***** is the directory where the HLAPI/UNIX is installed).

  The descriptor table of any process calling HLAPI/UNIX services must allow enough entries for the sockets and files used by the client interface.

Resources Needed for the Requester

The HLAPI/UNIX requester uses the following types of resources:

- Logical units (LUs)

  Specific to AIX:
  For APPC communication between an AIX requester and server, at least one LU on the requester host AIX machine is required. You may need more depending on your APPC configuration and runtime demands.

- Files
  - **/etc/services**
  - **/etc/hosts**
  - **/etc/inittab**

A requester also accesses a system profile, if one is specified, and the files listed for the requester in "Components of HLAPI/AIX" on page 348, "Components of HLAPI/HP" on page 349, and "Components of HLAPI/Solaris" on page 350.

Hardware and Software Requirements

The HLAPI/UNIX clients consist of two parts:

- Requester
- Client interface.

Both the UNIX requester and client interface can be run on the same machine or on different machines.

Hardware for HLAPI/UNIX

Specific to AIX:

To use the HLAPI/AIX client, you need an RS/6000 machine capable of running AIX for RS/6000. Machines that run the requester require either APPC/APPN or TCP/IP connectivity to the MVS host and TCP/IP connectivity to the machines that run the client interface. Machines that run the client interface (this can be the same machine as the requester) require TCP/IP connectivity to the machine that runs the requester.

The amount of fixed disk space you need to install the HLAPI/AIX is:

- Requester, approximately 310 KB
- Client, approximately 2340 KB

Specific to HP:

To use the HLAPI/HP client, you need an HP Series 700 or 800 workstation capable of running HP-UX Version 10 (up to and including Version 10.2). HP-UX includes TCP/IP.

To use the optional HLAPI for Java provided with the client, you need one of the following:

- HP 9000 Enterprise Business Server
- HP 9000 Workstation
- HP Visualize Workstation

The amount of fixed disk space you need to install the HLAPI/HP is:

- Requester, approximately 150 KB
- Client, approximately 2350 KB

Specific to Solaris:

To use the HLAPI/Solaris client, you need a Sun SPARCstation workstation capable of running Solaris Version 2.3, Version 2.4, or Version 2.5, all of which include TCP/IP.

The amount of fixed disk space you need to install the HLAPI/Solaris is:

- Requester, approximately 160 KB
- Client, approximately 375 KB

Software for HLAPI/UNIX

Specific to AIX:

Each RS/6000 machine that runs any part of the HLAPI/AIX requires the following software:

- IBM AIX Version 4.2 (5765-C34), or a subsequent release.

Each RS/6000 machine that runs the requester options of HLAPI/AIX to communicate with either a RES or an MRES with APPC requires the following additional software:

- IBM AIX SNA Server/6000 Version 2.1 (5765–247) or a subsequent release.

Each RS/6000 machine that runs REXX HLAPI/AIX requires the following software:

- IBM AIX REXX/6000 (5764-057)

To use the optional HLAPI for JAVA provided with the client, you must have JDK Version 1.1 or higher.

Specific to HP:

Each HP machine that runs any part of the HLAPI/HP requires the following software:

- HP-UX Version 10 (up to and including Version 10.2)

To use the optional HLAPI for JAVA provided with the client, you must have JDK Version 1.1 or higher and HP-UX Version 10.2.

Specific to Solaris:

Each Sun machine that runs any part of the HLAPI/Solaris requires the following software:

- Solaris Version 2.5.1.

To use the optional HLAPI for JAVA provided with the client, you must have JDK Version 1.1.6 or higher and Solaris Version 2.5.1.

27

Installing and Setting Up HLAPI/UNIX

Installing and Setting Up HLAPI/UNIX The two components of HLAPI/UNIX are packaged as two separately installable options, the requester option and the client interface option. You can install either option separately or both options together on a UNIX host.

Specific to AIX:
If you do not specify the commit option on **installp** when you install a HLAPI/AIX option, you can also remove the option. This enables you to tailor the HLAPI/AIX environment on all AIX hosts in your particular HLAPI/AIX configuration. Refer to the *AIX Installation Guide* for information about **installp**.

Installing HLAPI/UNIX involves the following steps:

1. Plan your HLAPI/UNIX configuration.
2. Distribute the HLAPI/UNIX application package.
3. Install HLAPI/UNIX.
4. Configure HLAPI/UNIX and associated software.

If you need to remove a HLAPI/UNIX option, see "Removing HLAPI/UNIX Options" on page 248.

Note: Upgrades or patches that can be downloaded from a Tivoli Web site may be available for HLAPI/UNIX. Visit the Tivoli Information Management for z/OS Web site for more information.

http://www.tivoli.com/infoman

Planning a HLAPI/UNIX Configuration

Decide which UNIX hosts will run your HLAPI/UNIX client application programs. You need to install the client interface option on each of these UNIX hosts.

Decide which UNIX hosts will enable communication between client interfaces on UNIX and Tivoli Information Management for z/OS servers on MVS. You must install the requester option on each UNIX host that communicates with MVS.

Decide which communication protocol the client application programs will want the requester to use.

Specific to AIX:
If an application program will use APPC, install SNA Server⁄6000 on each AIX requester host that the application program will use.

Setting Up HLAPI/AIX

After you decide which combination of HLAPI/AIX options (requester, client interface, or both) your installation requires on each AIX host in your HLAPI/AIX configuration, decide how to distribute the HLAPI/AIX **installp** image to the AIX hosts.

The **installp** image of HLAPI/AIX is delivered on a CD-ROM.

You can use one of the following methods to distribute the HLAPI/AIX options to your AIX hosts:

- Install the HLAPI/AIX options directly from a CD-ROM to the target AIX hosts.
- Copy the HLAPI/AIX options from a CD-ROM to an RS/6000 hard drive. If your AIX systems do not use a distributed file system, copy the HLAPI/AIX options to the target AIX hosts.

The methods are described in the following sections.

Distributing HLAPI/AIX from a CD-ROM

This method directly distributes the HLAPI/AIX **installp** image from a CD-ROM to an AIX host. Do these steps on each AIX host that requires an HLAPI/AIX option:

1. Log on to the RS/6000 as the root user.
2. Insert the HLAPI/AIX installation CD-ROM into a CD-ROM drive.
3. Mount the CD-ROM on a file system, such as **/cdrom** on the AIX host.
4. Start the **installp** utility to read the **installp** image on the CD-ROM. See “Installing Options from a CD-ROM” on page 239 for the command syntax.

Distributing HLAPI/AIX from a File System

This method copies the HLAPI/AIX **installp** image from a CD-ROM to a file system on a source AIX host. The HLAPI/AIX `installp` image is then copied from the source AIX host to target AIX hosts.

To copy the **installp** image from the HLAPI/AIX installation CD-ROM to a source AIX host, do the following steps:

1. Log on to the RS/6000 as the root user.
2. Insert the HLAPI/AIX installation CD-ROM into a CD-ROM drive.
3. Mount the CD-ROM on a file system, such as **/cdrom** on the AIX host.
4. Create an AIX file. Use the **bffcreate** utility to copy the **installp** image. In addition to copying the **installp** image, **bffcreate** creates a hidden file named **.toc** that contains a table of contents for the **installp** file. The syntax of the bffcreate utility is:

   ```
   bffcreate -qvd' <source>' -t'<target_directory>' '-X' idbhlapi
   ```

   <source>
   The full path name of the **INSTALLP** image on the mounted CD

   <target_directory>
   The path of the directory to contain the **installp** image.

   For example, to copy the **installp** image from the CD-ROM drive **/cdrom** to the directory **/usr/sys/inst.images**, specify the command as follows:

```
bffcreate -qvd'/cdrom/hlapi/aix/idbhlapi.inst_images'
-t'/usr/sys/inst.images' '-X' idbhlapi
```

After this command finishes, the **installp** image exists in the directory with the name **/usr/sys/inst.images/idbhlapi.usr.1.0.9609.2700**. The numeric part of the name varies with the release level of the **installp** image. The **.toc** file is in this directory as well. Note that the **.toc** file is a hidden file.

5. If your AIX hosts do not support a distributed file system that allows transparent sharing of files, start the ftp utility to copy both the **installp** image and **.toc** file from the source AIX host to the target AIX hosts.

   Note: If you use a file transfer utility to distribute the **.toc** file and **installp** image, make sure you perform the file transfer in binary mode to avoid changing their contents during the transfer.

Installing HLAPI/AIX on the RS/6000 System

After distributing the HLAPI/AIX **installp** image and **.toc** file, install the appropriate HLAPI/AIX options on the AIX hosts. The following two methods describe how to install HLAPI/AIX options from a CD-ROM and from a file system. Use whichever method is appropriate for your installation.

Note: You must log on as the root user to install software with the **installp** utility.

Installing Options from a CD-ROM

To install HLAPI/AIX options from an **installp** image on a CD-ROM, do the following steps:

1. Insert the CD-ROM into the appropriate drive.
2. Mount the CD-ROM on a file system, such as **/cdrom**
3. Start the **installp** utility. Type the following command on an AIX command line:

   ```
   installp -qaId'<source>' '-X' <option>
   ```

   <source>
   The full path name of the **installp** image on the mounted CD-ROM.

   <option>
   One of the following:

   idbhlapi.cli
   Installs only the client interface option

   idbhlapi.req
   Installs only the requester option

   idbhlapi.all
   Installs both the client interface option and the requester option

   For example, if you want to install only the requester option, type the following command:

   ```
   installp -qaId'/cdrom/hlapi/aix/idbhlapi.inst_image' '-X' idbhlapi.req
   ```

4. You must set the environment variable TISDIR to **/usr/lpp/idbhlapi** by using either the **setenv TISDIR=/usr/lpp/idbhlapi** command or the **export TISDIR=/usr/lpp/idbhlapi** command. You may want to put the appropriate command into your login script.

5. If you do not have Systems Network Architecture (SNA) installed on your requester system and you are using MRES with TCP/IP, you must complete your requester installation by typing the following command:

```
/usr/lpp/idbhlapi/idbinsta
```

Installing Options from a File System

To install HLAPI/AIX options from an **installp** image in a file system, start the **installp** utility by typing the following command on an AIX command line:

```
installp -qaId'<source_directory>' '-X' <option>
```

<source_directory>
The complete path of the directory containing the **installp** image

<option>
One of the following:

idbhlapi.cli
Installs only the client interface option

idbhlapi.req
Installs only the requester option

idbhlapi.all
Installs both the client interface option and the requester option

For example, if the **installp** image is the file **/usr/sys/inst.images/idbhlapi.usr.1.0.9609.2700** and you want to install both the client interface and requester options, type the following command:

```
installp -qaId'/usr/sys/inst.images' '-X' idbhlapi.all
```

If you do not have Systems Network Architecture (SNA) installed on your requester system and you are using MRES with TCP/IP, you must complete your requester installation by typing the following command:

```
/usr/lpp/idbhlapi/idbinsta
```

After you install the HLAPI/AIX options, delete the **installp** image and **.toc** file to save space on the host file system.

Setting Up HLAPI/HP and HLAPI/Solaris

After you decide which combination of HLAPI/UNIX options (requester, client interface, or both) your installation requires on each UNIX host in your HLAPI/HP or HLAPI/Solaris configuration, decide how to distribute the archive format (**tar**) to the UNIX hosts.

The **tar** archive of HLAPI/UNIX is delivered on a CD-ROM.

To distribute the HLAPI/UNIX options to your UNIX hosts, copy the HLAPI/UNIX options directly from the CD-ROM to the target UNIX hosts.

This is described in the following section.

Distributing HLAPI/HP and HLAPI/Solaris from CD-ROM

Do these steps on each UNIX host that requires a HLAPI/HP or HLAPI/Solaris option:

1. Log on on the UNIX system as the root user.

2. Create a directory where you want the HLAPI code installed.
3. Change to the directory you just created.
4. Insert the CD-ROM into the appropriate drive and mount on a file system, such as **/cdrom** if necessary.
5. Copy the **tar** image from the appropriate directory (**/cdrom/hlapi/hpux** for HLAPI/HP or **/cdrom/hlapi/solaris** for Solaris) on the CD-ROM file system.

Installing HLAPI/HP and HLAPI/Solaris

After copying the HLAPI/HP or HLAPI/Solaris **tar** archive, install the appropriate HLAPI/HP or HLAPI/Solaris options on the respective host system.

1. Log on on the UNIX system as the root user.
2. Change to the directory where you copied the `tar` archive.
3. Extract the files from the archive. Type the following command on a UNIX command line:
   ```
   tar -xvf idbhlapi.tar
   ```
4. You must set the environment variable TISDIR to **/usr/lpp/idbhlapi** by using either the **setenv TISDIR=/usr/lpp/idbhlapi** command or the **export TISDIR=/usr/lpp/idbhlapi** command. You may want to put the appropriate command into your login script.
5. Run the **IDB** installation script. Type the following command on a UNIX command line:
   ```
   ./idbhlapi/idbinstl
   ```
   You must respond to the prompt and specify which option you want to install:

   c Installs only the client interface option

   r Installs only the requester option

   b Installs both the client interface option and the requester option

Configuring HLAPI/UNIX and Associated Software

After you install HLAPI/UNIX options on your UNIX hosts, you must configure the communication link between the requester host and a Tivoli Information Management for z/OS server. You must also enable communication between the client hosts and requester hosts. Then you must define database profiles to enable your client application programs to use HLAPI/UNIX services. You may also need to create system profiles for your requester hosts. The following sections tell you how to configure your communications software and update various files. “HLAPI/UNIX Profiles, Environment Variables, and Data Logging” on page 251 provides information about defining database and system profiles.

The configuration steps depend on whether client application programs will use a server that supports APPC or TCP/IP. “Configuring HLAPI/UNIX for TCP/IP” on page 245 gives you the steps for a TCP/IP configuration. The following section gives you the steps for an AIX APPC configuration.

Configuring HLAPI/AIX for APPC

The machine to be configured must have AIX Version 4.2 and SNA Server/6000 Version 2.1 installed. It must also have a token-ring adapter. AIX SNA Server/6000 is IBM's implementation of APPN for AIX. Version 2.1 supports APPN network nodes, end nodes, and low entry networking nodes.

For more detailed configuration information, refer to the following publications:

- *AIX SNA Server/6000: User's Guide*
- *AIX SNA Server/6000: Configuration Reference*
- *Multiplatform Configuration Guide* (This guide is available through your IBM marketing representative.)

SNA Server/6000 stores its configuration information within encoded profiles that are accessed using the System Management Interface Tool (SMIT). SMIT presents panels on which you enter your configuration values.

All of the values used in SNA Server/6000 are case-sensitive, so be sure to always enter profile names and other configuration values in the same case throughout the SMIT panels.

Within SMIT, SNA Server/6000 configuration panels can be accessed by selecting the following series of panels:

```
--Communications Applications
  --SNA Server/6000
    --Configure SNA Profiles
```

All other panels referenced in this section are accessed from this base level within **SMIT**.

You must have the base configuration of AIX SNA Server/6000 installed and have system group access to the system before you can start the SNA subsystem. You need system group authority to make the changes described. Refer to the *AIX SNA Server/6000: User's Guide* for information on SNA Server/6000 installation.

A token ring data link connection (DLC) must already exist on the system. Other DLCs will require link station profile definitions that would replace the token ring link station profile described below.

You need to define:

- Control point profile
- Token ring link station profile
- Token ring SNA DLC profile
- Local LU (6.2) profile
- Side information profile.

Control Point Profile

SNA Server/6000 has one control point profile, **node_cp**, which is used to identify the local node to the network. You must complete the control point profile before you can start SNA and use APPC. Configuration of this profile is handled through initial node setup. The initial node setup function also allows you to configure a single link station to provide a link to one remote station. This gives you a single entry point to define the minimum amount of information for SNA Server/6000 to operate.

From within SMIT, initial node setup can be accessed on this panel:

```
--Initial Node Setup
```

The following figures show the panels that display using initial node setup. The first panel requires you to select the primary link type you will use for this configuration. Select **token_ring**.

```
Choose the DLC type you wish this configuration to       token_ring
 represent.
```

The second panel requests the information necessary to configure the control point, link station, and SNA DLC profiles. Replace the words that appear **in bold** with the appropriate values for your network configuration.

```
Control Point name     = cp_name
Control Point type     = appn_end_node         # See note
Local network name     = network_name
XID node ID            = *                     # See note

Optional link station information:

Link station type      = token_ring
Link station name      = ls_profile_name
Calling link station?  = yes
Link address           = LAN_address
```

Note: By default, your machine is configured as an APPN end node. Change this value only if you are certain that you are to function as an APPN network node.

Note: Using an XID node ID for configuration is not recommended.

Defining Side Information

From within **SMIT**, the Side Information panel can be accessed by following this series of panels:

```
--Advanced Configuration
  --Sessions
    --LU 6.2
      --LU 6.2 Side Information
        --Add Profile
```

The following figure shows the Side Information panel. Replace the words that appear **in bold**

```
Profile name                              = "symdest"
Local LU or Control Point alias           = ""
  Partner LU alias                          = ""
  Fully qualified partner LU name           = "network_name.imserver_lu"
Mode name                                 = "mode"
Remote transaction program name (RTPN)    = "tpname"
RTPN in hexadecimal                       = no
Comments                                  = ""
```

symdest
Declares the symbolic destination name to use on the workstation to locate a server. Use this same symbolic destination name when you define your database profile on the workstation.

network_name.imserver_lu
Specifies a partner LU that is defined in this same **NDF** file on a **DEFINE_PARTNER_LU** statement. If the symbolic destination name is for a RES, the partner LU must be one that is defined on the MVS system as

scheduled. If the symbolic destination name is for an MRES, the partner LU must be one that is defined on the MVS system as nonscheduled.

network_name
Is the ID for the network the server's LU is on.

imserver_lu
Is the name of the server's logical unit.

mode
Specifies the name of a compiled log-on mode to use for the conversation. The mode name must match the mode specified in the TP profile for a RES or in the side information entry for an MRES. The log-on mode is an entry in **SYS1.VTAMLIB**. See "Defining the Log-on Mode" on page 30 for information on defining a mode.

tpname
If the symbolic destination name represents a RES, this value must match the name used for the TP profile on the MVS system where the RES resides.

If the symbolic destination name represents an MRES, this value must match the value on the **TPNAME** parameter of the side information entry that *symdest* maps to.

Creating a Local LU 6.2 Profile

You can also add the local LU 6.2 profile if you plan to use a local LU name that is different from the local CP name. If you create a local LU name using a LU 6.2 local LU profile, then the local LU alias field in that profile must be specified in the **Local LU or Control Point alias** field in the side information profile. If no local LU alias is specified in the side information profile, the local CP name is used as the local LU by default.

From within SMIT, the local LU panel can be accessed by following this series of panels:

```
--Advanced Configuration
  --Sessions
    --LU 6.2
      --LU 6.2 Local LU
```

The following figure shows the Local LU profile panel. Replace the words that appear **like this**.

```
Profile name                                     = profile_name
Local LU name                                    = locallu
Local LU alias                                   = locallu
Local LU is dependent?                           = no
   If yes,
       Local LU address                            = 1
       SSCP ID                                     = *
       Link Station Profile name                   = ""
Conversation Security Access List Profile name   = ""
```

Verifying Configuration

After the profiles are created, they must be verified using either the command **verifysna**, or in SMIT by following this hierarchy of panels:

```
--Advanced SNA Configuration
  --Verify Configuration Profiles
```

Before profiles can be used by SNA and APPC, they must be verified with the update option (either normal or dynamic, depending whether SNA is running). Verification ensures that changes are correct and that no profiles are in conflict with each other. Profile additions, changes, or deletions will not take effect until the profile database is verified and updated.

Starting and Stopping APPC

APPC is part of the AIX SNA Server/6000 subsystem. You can use the following command to test the status of the SNA subsystem:

```
sna -display global
```

or, for short,

```
sna -d g
```

If the SNA subsystem is not started, you can start it with the command:

```
sna -start sna
```

or, for short,

```
sna -s
```

SNA Server/6000 can be configured to accept incoming link activation requests from remote stations automatically. This function, known as dynamic link station support, is configured by default.

To stop the SNA subsystem, use this command:

```
sna -stop sna
```

Determining Values

- LU name

  The LU name is specified in a Local LU 6.2 profile as the ″Local LU name″ parameter.

- Network name

  The network name is specified in the Control Point profile, **node_cp** as the ″Local network name″ parameter.

- Control Point (CP) name

  The CP name is specified in the Control Point profile **node_cp** as the ″Control Point name″ parameter.

- LAN address

  The LAN address is hard-coded on the Token-Ring adapter in the AIX machine or is over-ridden by a value called the “locally administered LAN address.” It is a 12-digit hexadecimal value. You can find the LAN address by running the command **lscfg -v**. The LAN address is specified by the Network Address parameter under the **tok0** resource. This parameter will specify the locally administered address if there is one.

Configuring HLAPI/UNIX for TCP/IP

Update the file `/etc/services` on a requester host to associate a service name or alias with a TCP/IP port number of an MRES with TCP/IP server. You must specify a service name and port number of each server host the requester needs to be able to connect to. The port numbers must match those on the MVS host designated for the Tivoli Information Management for z/OS MRES with TCP/IP servers. The general format of an entry in **/etc/services** is:

```
<service>     <port>/tcp   <alias_list>       #<comment>
```

<service>
: The server service name

<port>
The server port number

<alias_list>
Alias definitions for the service

<comment>
Comment text that describes the service.

For example, to associate the default server service name (infoman) with the default server port number (1451), you must place the following line in the file **/etc/services** before you run the HLAPI/UNIX:

```
infoman    1451/tcp      #default HLAPI/UNIX requester and MRES server
```

The default service name and default port number are reserved for Tivoli Information Management for z/OS. You can use them to designate your server service. If a client does not specify a server service (**IDBSERVERSERVICE**) in the database profile specified on the HL01 transaction, **infoman** will be assumed. Therefore, be sure to include it in the **/etc/services** file.

Be sure that your client application programs are aware of the service names that you define. If you access an MRES for TCP/IP that uses a port number other than the default (**infoman/1451**), you must specify the server service name in the **IDBSERVERSERVICE** keyword in your database profile.

You must update the **/etc/hosts** file on the requester to include the host names and corresponding dotted-decimal IP address of any server hosts that the requester needs to be able to connect to on behalf of its clients. If the clients use host names to identify the server host, you must update **/etc/hosts** on the requester to include those hostnames and their corresponding dotted-decimal IP address.

Defining the Client Interface to Requester Communication Link

To define the communication link between a requester and a client interface, you must update the **/etc/services** file on all hosts where either the requester or the client interface is installed. You must also update the **/etc/hosts** file on all the hosts where the client interface is installed. The following sections tell you how to do that.

Note: Service names, host names, and aliases are case-sensitive.

Updating /etc/services on a Requester Host

Update the file **/etc/services** on a requester host to associate the requester service name or alias with the TCP/IP port number of the requester. The general format of a requester entry in **/etc/services** is:

```
<service>     <port>/tcp   <alias_list>       #<comment>
```

<service>
The requester service name

<port>
The requester port number

<alias_list>
Alias definitions for the service

<comment>
Comment text that describes the service.

For example, to associate the default requester service name (infoman) with the default requester port number (1451), you must place the following line in the file **/etc/services** before you run the HLAPI/UNIX:

```
infoman    1451/tcp      #default HLAPI/UNIX requester and MRES server
```

The default service name and default port number are reserved for Tivoli Information Management for z/OS. You can use a different combination of service name (or alias) and port number for a requester. For each combination, you must specify a corresponding line in the **/etc/services** file to associate the port number with the service name or alias. Port numbers greater than 6000 are user-definable.

Only one service can use a given port number and a given service name or alias in an **/etc/services** file on a UNIX host. However, you can use the same port number or the same service name (or alias) on different UNIX hosts.

You can run multiple requesters on a single UNIX host. Use a different service name for each requester and associate the service name with a port number in the requester's **/etc/services** file. You also need to create a system profile for each requester and specify the appropriate service name in each profile.

You might need to run the following UNIX commands after making changes to the **/etc/services** file:

Specific to AIX:

```
inetimp refresh -s inetd
```

For more information, refer to the *AIX Version 4.2 Files Reference*.

Specific to HP:

```
inetd –c
```

For more information, refer to the *HP-UX Reference, Volume 3*

These commands enable the **inetd** daemon to recognize the changes.

You can direct UNIX to start a HLAPI/UNIX requester when a client interface attempts to establish the initial contact with the requester. This eliminates the need to start a requester before starting a client application program. See "The HLAPI/UNIX Requester" on page 263 for information on how to do this.

Updating /etc/services and /etc/hosts on a Client Host

You must associate the client interface with each local or remote requester that the client interface communicates with. To do this, update the file **/etc/services** on the client interface host with the requester service names and their corresponding TCP/IP port numbers. Also update the file **/etc/hosts** on the client interface host with the host name and corresponding dotted decimal IP address of the requester hosts.

The general format of a requester entry in **/etc/services** is:

```
<service>     <port>/tcp   <alias_list>       #<comment>
```

<service>
The requester service name

<port>
The requester port number

<alias_list>
Alias definitions for the service

<comment>
Comment text describing the service.

For example, to associate the default requester service name (`infoman`) with the default requester port number (1451), put the following line in **/etc/services** file before you attempt to run the HLAPI/UNIX:

```
infoman    1451/tcp      #default HLAPI/UNIX Requester
```

The default service name and default port number are reserved for the exclusive use of Tivoli Information Management for z/OS. However, you may use a different combination of service name (or alias) and port number for a requester. For each combination you specify for your requesters, put a corresponding line in the client's **/etc/services** file to associate the port number with the service name or alias. Port numbers greater than 6000 are user-definable; they should not conflict with reserved ports.

In an **/etc/services** file on a UNIX host, only one service can use a given port number and a given service name or alias. However, different **/etc/services** files on different UNIX hosts may use the same port number or the same service name (or alias). This is because each UNIX host manages its own set of ports and service names.

You might need to run the following commands after making changes to **/etc/services**:

Specific to AIX:

```
inetimp refresh -s inetd
```

For more information, refer to the *AIX Version 4.2 Files Reference*.

Specific to HP:

```
inetd —c
```

For more information, refer to the *HP-UX Reference, Volume 3*. These commands enable the **INETD** daemon to recognize the changes.

Removing HLAPI/UNIX Options

Specific to AIX:
If you did not specify the **commit** option on **installp** when you installed the HLAPI/AIX client interface or requester, you can remove them. To remove one or both of the options from an AIX host, do the following steps:

1. Log on as the root user.
2. Type the following on an AIX command line:

   ```
   installp -r '-X' <option>
   ```

   <option>
   Is one of the following:

idbhlapi.cli
Removes only the client interface option

idbhlapi.req
Removes only the requester option

idbhlapi.all
Removes both the client interface option and the requester option.

For example, to remove the client interface option from a host, type the following on an AIX command line:

```
installp -r '-X' idbhlapi.cli
```

Specific to HP:
To remove the HLAPI/HP client interface or requester, erase the files and directories listed in “Components of HLAPI/HP” on page 349.

Specific to Solaris:
To remove the HLAPI/Solaris client interface or requester, erase the files and directories listed in “Components of HLAPI/Solaris” on page 350.

28

HLAPI/UNIX Profiles, Environment Variables, and Data Logging

Certain aspects of HLAPI/UNIX can be customized to meet the requirements of your application. You do this by specifying profile keywords and values in two types of HLAPI/UNIX profiles. The profiles are UNIX text files; so you can use any text editor to create and update them.

The first type of profile is a *system profile*, which is associated with a requester. You can specify the name of a system profile as an optional argument of a command that starts a HLAPI/UNIX requester. The parameters in a system profile control aspects of HLAPI/UNIX operation. If no system profile is specified, the requester uses default values for all system profile parameters.

The second type of profile is a *database profile*, which is associated with a transaction sequence. The name of a database profile must be specified by a control PDB passed to HLAPI/UNIX as part of an HL01 transaction. The parameters in a database profile control aspects of HLAPI/UNIX operation.

Profile Syntax

Profile parameter entries are specified in the form

```
<keyword>=<data_value>
```

<keyword>
: Represents one of the keywords defined by HLAPI/UNIX.

=
: Is a literal character.

<data_value>
: Represents a data value to be associated with the keyword.

Whitespace characters (blanks or tabs) can precede or follow the value for *keyword* or *data value*. The *data value* includes all characters following the = to the end of the line. Each profile parameter must be specified entirely on a single line. For example,

```
IDBTIMEOUT = 60
```

Profile comments are specified in the form

```
REM <comment_text>
```

REM A literal keyword

<comment_text>
: Any sequence of characters up to the end of the line.

Whitespace characters (blanks or tabs) can precede the **REM** keyword; at least one whitespace character must immediately follow the **REM** keyword. Each profile comment must be specified entirely on a single line. For example,

```
REM    This is an example of a comment in a profile.
```

Each line of a profile must contain exactly one of the following:
- A profile parameter entry
- A profile comment
- Whitespace only.

Keywords cannot be duplicated in profiles. If duplicate keywords are detected, processing stops and an error is returned to the client application program.

Profile keywords and data values are case-sensitive. Profile keywords must be entered with uppercase characters only. Profile data values must match their definitions in your **/etc/services**, **/etc/hosts**, or other system configuration file.

When specifying a numeric value in a profile, use decimal digits to represent the value. Do not place any delimiter characters, such as commas or periods, among the digits of the value.

System Profile

You can specify the name of a single system profile as an optional parameter of a command that starts a HLAPI/UNIX requester. If a system profile is not specified when a requester is started or if some parameters are unspecified in the system profile, the requester uses the default values for the parameters. A requester reads a system profile only once when the requester is started. The developer of the client application program or UNIX system administrator creates system profiles. Multiple requesters can use the same system profile.

You can specify either the file name only or you can fully qualify the name with its path and drive. If you specify only a file name, the file is obtained from the current working directory. If the file is not found, the qualifying path name is obtained from the value of the UNIX environment variable **IDBSMPATH**. See “IDBSMPATH” on page 261 for additional information.

The valid keywords for system profiles follow.

IDBINBOUNDBUFSIZE

This value specifies the number of bytes to allocate for the communication buffers that the requester uses for receiving data from the server for transmittal to the client interface.

The buffer size can affect communication performance. For example, you might wish to specify a large buffer size, such as 28672, if the requester will be handling database search transactions that generate large amounts of reply data. For related information, see “IDBOUTBOUNDBUFSIZE” on page 253.

Valid values:

Any integer from 1 to 32767, inclusive.

Values are rounded up to the nearest multiple of 4096. If the value after rounding is 32768, it is adjusted to 32767.

Default value:
4096

IDBMAXCMS

This value specifies the maximum number of child processes the requester can create to manage conversations between the requester and the server. The conversations are in a one-to-one correspondence with the requester processes that manage the conversations. The actual number of conversations might be further limited by constraints on UNIX system resources. One example is the UNIX system variable **_SC_CHILD_MAX**, which limits the number of processes for each user.

Valid values:
Any integer from 1 to 65534, inclusive.

Default value:
65534

IDBOUTBOUNDBUFSIZE

This value specifies the number of bytes to allocate for the communication buffers that the requester uses to transmit transaction data received from the client interface to the server.

The buffer size can affect communication performance. For example, you might wish to specify a large buffer size, such as 8192, if the requester will be handling record creation transactions that involve large amounts of record data. For related information, see “IDBINBOUNDBUFSIZE” on page 252.

Valid values:
Any integer from 1 to 32767, inclusive.

Values are rounded up to the nearest multiple of 4096. If the value after rounding is 32768, it is adjusted to 32767.

Default value:
4096

IDBSERVICENAME

This value specifies the service name for the requester that is listed in the UNIX system file /etc/services.

Valid values:
Any valid service name or alias. Service names and aliases are case-sensitive.

Default value:
infoman

This is the default requester service name suggested during HLAPI/UNIX installation. If you specified a different service name in your **/etc/services** file during installation in “Defining the Client Interface to Requester Communication Link” on page 246, you must specify that service name or its alias on this parameter in a system profile.

IDBSHARECMS

This keyword determines whether the requester should enable or disable conversation sharing. When conversation sharing is enabled, the requester assigns new client applications to an existing conversation if criteria such as same server and same security ID are met.

When conversation sharing is disabled (the default), each client application is assigned its own dedicated conversation. A conversation is terminated when the last client assigned to it submits an HL02.

Note: If you choose to use conversation sharing, you must be aware that there is a potential for a delay because transactions are handled synchronously. Thus, if Client A and Client B share a conversation, and Client A submits a long search and Client B submits an update, Client B will wait for Client A's search to complete before its transaction can be processed.

Note: If you are using pre-started API sessions (described in "MRES with Pre-started API Sessions Considerations" on page 18), you must disable conversation sharing.

The number of conversations that the requester can start up is based on requester and operating system limitations. When conversation sharing is disabled, you may be more likely to reach these limits if your client applications hold on to conversations for an extended period. Any of the HLAPI/UNIX clients provides the ability to start multiple concurrent requesters as long as each requester is assigned a unique port number identified by the **IDBSERVICENAME** in the system profile. You may wish to route client applications that require dedicated conversation to requesters that you have started with conversation sharing disabled, and route your other client applications to other requesters with conversation sharing enabled.

Valid entries: **0** (conversation sharing disabled) or **1** (conversation sharing enabled).

Default value: **0** (conversation sharing disabled).

IDBTIMEOUT

This value specifies the maximum number of minutes that can elapse between transactions in a single transaction sequence. This is called *idle time*, which is defined as the time interval between the completion of one transaction and the start of the next transaction in the same transaction sequence. The start time is determined by when the transaction becomes the active transaction in the requester-server conversation. The end time is determined by when the transaction stops being the active transaction in the conversation.

If the idle time for a transaction sequence exceeds the smaller of this value or the value of **IDBIDLECLIENTTIMEOUT** in the database profile for the transaction sequence, the requester refuses to process additional transactions in the sequence (including any transaction pending on the conversation queue). This allows the requester to end logical sessions and conversations if a client application program does not submit an HL02 transaction for a transaction sequence.

Valid values:
Any integer from 0 to 35791394, inclusive.

The value is specified in minutes, except that a value of 0 corresponds to an infinite interval.

Default value:
35791394

System Profile Example

```
REM*******************************************************************
REM
REM          SAMPLE HLAPI/AIX System Profile
REM
REM*******************************************************************

IDBINBOUNDBUFSIZE       =     28672

IDBMAXCMS               =       128

IDBOUTBOUNDBUFSIZE      =      8192

IDBSERVICENAME          = hlapiaix9

IDBSHARECMS             =         0

IDBTIMEOUT              =        60
```

Database Profile

Your client application must specify the name of a single database profile in a **DATABASE_PROFILE** control PDB on the HL01 transaction of each transaction sequence. If a **DATABASE_PROFILE** PDB is not specified or if multiple **DATABASE_PROFILE** PDBs are specified for an HL01 transaction, an error is returned to the client application program. If a **DATABASE_PROFILE** PDB is specified on a transaction other than HL01, the PDB is ignored. A client interface reads the database profile only once per transaction sequence, and that is during the HL01 transaction. The UNIX system administrator, client application developer, or client application user creates the database profile for a transaction sequence. Multiple client application programs can use the same database profile.

The specified database profile must contain at least an entry for the **IDBSYMDESTNAME** keyword or the **IDBSERVERHOST** keyword. It must not contain both of these keywords. This information identifies to which Tivoli Information Management for z/OS server the requester establishes a communication link for the client application. See “IDBSYMDESTNAME” on page 259 and “IDBSERVERHOST” on page 258 for information on these keywords.

You can specify either the file name only or you can fully qualify the name with its path and drive. If you specify only a file name, the file is obtained from the current working directory. If the file is not found, the qualifying path name is obtained from the value of the UNIX environment variable **IDBDBPATH**. See “IDBDBPATH” on page 260 for more information.

The valid keywords for database profiles follow.

IDBCHARCODESET

This keyword specifies the code set to use on the client interface host for data conversion during the transaction sequence. The client interface converts the text of a transaction request from the client code set to the server code set. The client interface also converts the text of a transaction reply from the server code set to the client code set. The keyword **IDBSERVCHARCODESET** specifies the server code set. See “IDBSERVCHARCODESET” on page 258 for information on that keyword.

Valid values:

Any code set supported by the AIX or HP `iconv` subroutine. Refer to *AIX Version 4.2 General Programming Concepts* or *HP-UX Reference, Volume 1* for a list of supported code sets.

Default value:

The code set in use by the process submitting the transaction to HLAPI/AIX or HLAPI/HP when the database profile is read. The client interface queries the operating system to determine the code set.

Specific to Solaris:

HLAPI/Solaris does not support the **iconv** subroutine. IBM provides the translate tables to perform the necessary data conversion between the IBM-850 and IBM-037 code pages.

IDBDATALOGLEVEL

This keyword determines whether the client interface logs transaction data. The value specified with this keyword can be overridden by the UNIX environment variable **IDBDATALOGLEVEL**.

Valid values:

0 (logging disabled) 1 (logging enabled)

Default value:

0 (logging disabled).

The default value is applied only if neither the UNIX environment variable **IDBDATALOGLEVEL** nor the database profile keyword **IDBDATALOGLEVEL** specifies a value.

IDBIDLECLIENTTIMEOUT

This keyword specifies the maximum number of minutes that can elapse between transactions in a single transaction sequence. This is called *idle time*, which is defined as the time interval between the completion of one transaction and the start of the next transaction in the same transaction sequence. The start time is determined by when the transaction becomes the active transaction in the requester-server conversation. The end time is determined by when the transaction stops being the active transaction in the conversation.

If the idle time for a transaction sequence exceeds the smaller of this value or the value of **IDBTIMEOUT** in the system profile associated with the requester, the requester refuses to process additional transactions in the sequence (including any transaction pending on the conversation queue). This allows the requester to end logical sessions and conversations if a client application program does not submit an HL02 transaction for a transaction sequence.

Valid values:

Any integer from 0 to 35 791 394, inclusive. The value is specified in minutes, except that a value of 0 corresponds to an infinite interval.

Default value:

60

IDBLOGFILENAMEACTIVE

This keyword specifies the name of the active log file for the transaction sequence. Multiple transaction sequences can use the same log file. However, it is recommended that you limit the number of simultaneous transaction sequences per log file to reduce contention for write access to the file and to prevent the file from being archived frequently.

Valid values:
Any valid file name on your UNIX system. File names and path names are case-sensitive.

Default value:
IDB_LOG.ACT

IDBLOGFILENAMEOLD

This keyword specifies the name to give an active log file when it is archived. Active log files are archived when they reach their maximum size as specified on the keyword **IDBLOGFILESIZE**. See "IDBLOGFILESIZE".

Valid values:
Any valid file name on your UNIX system. File names and path names are case-sensitive.

Default value:
IDB_LOG.OLD

IDBLOGFILESIZE

This keyword specifies the approximate maximum size, in bytes, of the active log file. If logging a transaction causes the active log file to exceed this size, the active log file is archived. Archiving involves two steps:

1. Closing and renaming the active log file
2. Opening a new active log file.

Valid values:
Any integer from 0 to 10 485 760, inclusive. (The upper limit equals 10 megabytes.)

If a value between 1 and 4 095 is specified, then 4 096 is substituted. If the value 0 is specified, HLAPI/UNIX does not restrict the size of the log file.

Default value:
262 144 (Equal to 256 kilobytes.)

IDBREQUESTERHOST

This keyword identifies the UNIX host that is running the requester so that communication can be established between the client interface and the requester.

Valid values:
Any valid TCP/IP address in dotted-decimal format, or any valid alias, such as **host5**, that is associated with a dotted-decimal address in the UNIX system file **/etc/hosts**. Host aliases are case-sensitive.

Default value:
The UNIX host running the client interface that is handling the transaction sequence. The default host can be determined with the **gethostname()** system call.

IDBREQUESTERSERVICE

This keyword specifies the service name of the requester you want to establish communication with. The service name must be listed in the **/etc/services** file on the client host.

Valid values:
Any valid service name or alias.

Use the UNIX command **odmshow InetServ** to display the maximum length of service names and aliases. A service name has a maximum length of 19, while an alias has a maximum length of 49. Service names and aliases are case-sensitive.

Default value:
infoman

This is the default requester service name suggested during HLAPI/UNIX installation. If you specified a different service name in your **/etc/services** file during installation in "Defining the Client Interface to Requester Communication Link" on page 246, you must specify that service name or its alias on this parameter in a database profile.

IDBSERVCHARCODESET

This keyword specifies the code set used on the server. The client interface converts the text of a transaction request from the client code set to the server code set. The client interface also converts the text of a transaction reply from the server code set to the client code set. The keyword **IDBCHARCODESET** specifies the client code set. See "IDBCHARCODESET" on page 255 for information on that keyword.

Valid values:
Any code set supported by the AIX or HP **iconv** subroutine. Refer to *AIX Version 4.2 General Programming Concepts* or *HP-UX Reference, Volume 1* for a list of supported code sets.

Default value:

Specific to AIX:
IBM-037 (US English)

Specific to HP:
american_e

Specific to Solaris:
HLAPI/Solaris does not support the **iconv** subroutine. Tivoli provides the translate tables to perform the necessary data conversion between the IBM-850 and IBM-037 code pages.

IDBSERVERHOST

This keyword identifies the MVS host that is running the MRES with TCP/IP server you want the requester to establish communication with for your client application.

Valid values:
Any valid IP address in dotted-decimal format, or any valid host name, such as **mvshost**, that is associated with a dotted-decimal IP address in the UNIX system file **/etc/hosts** on the requester host. Host names are case-sensitive.

Default value:
None. This value is required if **IDBSYMDESTNAME** is not specified. There is no default.

If you use this keyword you must not use the **IDBSYMDESTNAME** keyword.

IDBSERVERSERVICE

This keyword specifies the service name of the MRES with TCP/IP you want to establish communication with. The service name must be listed in the **/etc/services** file on the requester host.

Valid values:
Any valid service name or alias.

Note: Use the UNIX command `odmshow InetServ` to display the maximum length of service names and aliases. A service name has a maximum length of 19, while an alias has a maximum length of 49. Service names and aliases are case-sensitive.

Default value:
infoman

IDBSERVERSERVICE is an optional keyword. If you do not specify it when you specify **IDBSERVERHOST**, the default is assumed. If you specify **IDBSYMDESTNAME**, **IDBSERVERSERVICE** is ignored.

IDBSYMDESTNAME

This keyword identifies either a RES or an MRES with APPC on the MVS host that you want the requester to establish communication with for your client application. This value must correspond to a side information entry on the requester host. Only an AIX requester can establish communication with a RES or MRES with APPC.

Note: You must use this keyword if your UNIX client connects to an AIX requester that uses APPC to connect to MVS. However, you cannot use this keyword if your UNIX client connects to an HP or Solaris requester.

Valid values:
Any valid APPC symbolic destination name defined in the requester's side information to identify the transaction program to run on the MVS host. A symbolic destination name is a character string from 1 to 8 bytes long. These names are case-sensitive.

Default value:
None. This value is required if **IDBSERVERHOST** is not specified. No default value is defined.

If you use this keyword you must not use the **IDBSERVERHOST** keyword.

Database Profile Example

```
REM***********************************************************************
REM
REM                    SAMPLE HLAPI/AIX Database Profile
REM
REM***********************************************************************

IDBREQUESTERHOST     =           zeus
IDBREQUESTERSERVICE  =         hlapi9
IDBIDLECLIENTTIMEOUT =             30

IDBSYMDESTNAME       =       infosrv2
IDBSERVCHARCODESET   =        IBM-037

IDBCHARCODESET       =        IBM-850

IDBDATALOGLEVEL      =              1
IDBLOGFILENAMEACTIVE =  helpdesk5.log
IDBLOGFILENAMEOLD    =  helpdesk5.old
IDBLOGFILESIZE       =         262144
```

Environment Variables

HLAPI/UNIX recognizes some UNIX environment variables for which you can define values. Korn shell and Bourne shell environment variables are case-sensitive, but C shell environment variables are not case-sensitive. If you are using a shell with case-sensitive environment variables, you must enter the names of HLAPI/UNIX environment variables using only uppercase characters.

You can use the HLAPI/UNIX environment variables described in the following sections to override the corresponding database profile parameters and to qualify the path names for the profiles. Refer to

AIX Version 4.2 Management Guide: Operating System and Devices
or
How HP-UX Works: Concepts for the System Administrator

for information on using environment variables.

IDBDATALOGLEVEL

If this environment variable is set to a valid value when a client interface reads the database profile, the value in the environment variable overrides any value given for the database profile keyword **IDBDATALOGLEVEL**. Valid values are **0** (transaction data logging disabled) and **1** (transaction data logging enabled). See “IDBDATALOGLEVEL” on page 256 for additional information.

IDBREQUESTERHOST

If this environment variable is set to a valid value when a client interface reads the database profile, the value in the environment variable overrides any value set for the database profile keyword **IDBREQUESTERHOST**. You can specify any valid TCP/IP address in dotted-decimal format, or any valid alias. See “IDBREQUESTERHOST” on page 257 for additional information.

IDBREQUESTERSERVICE

If this environment variable is set to a valid value when a client interface reads a database profile, the value in the environment variable overrides any value set for the database profile keyword **IDBREQUESTERSERVICE**. You can specify any valid service name or alias. See “IDBREQUESTERSERVICE” on page 257 for additional information.

IDBDBPATH

This environment variable specifies a search path for locating a database profile when both of the following conditions are true:

1. The database profile file name does not explicitly specify a complete path from the root directory to the database profile file. This is called a *relative path*.
2. The database profile is not in the current working directory of the calling process.

You can specify multiple paths on **IDBDBPATH**. For example, if the file name of a database profile is specified using a relative path, then the following assignment:

```
IDBDBPATH=/:/usr/profiles/:
```

causes a client interface to seek the database profile first in the current working directory, then in the root directory /, then in the directory **/usr/profiles**.

The ':' following the final path specification is optional. The trailing '/' of the second path specification is also optional. For example, the following two assignments are equivalent:

```
IDBDBPATH=/:/usr/profiles/:

IDBDBPATH=/:/usr/profiles
```

IDBSMPATH

This environment variable specifies a search path for locating a system profile when both of the following conditions are true:

1. The system profile file name does not explicitly specify a complete path from the root directory to the system profile file. This is called a *relative path*.
2. The system profile is not in the initial working directory of the requester process. The initial working directory of the requester process is the current working directory at the time the requester is started.

You can specify multiple paths on **IDBSMPATH**. For example, if the file name of a system profile is specified using a relative path, the following assignment:

```
IDBSMPATH=/:/usr/profiles/:
```

causes a requester to seek the system profile first in the initial working directory of the requester process, then in the root directory /, then in the directory **/usr/profiles**.

The ':' following the final path specification is optional. The trailing '/' of the second path specification is also optional. For example, the following two assignments are equivalent:

```
IDBSMPATH=/:/usr/profiles/:

IDBSMPATH=/:/usr/profiles
```

Transaction Logging

HLAPI/UNIX transactions can be logged by the server as well as by the client interface. When both server logging and client interface logging are active, you might notice differences between entries in the server log and the corresponding entries in the client interface log. For example, the client interface logs PDBs with a data length of zero but does not send those PDBs to the server. Therefore, the server log records no zero-length PDBs.

Transaction Logging by a Client Interface

Every HLAPI/UNIX transaction sequence has an associated log file in which the client interface can record transactions. The value specified for the HLAPI/UNIX environment variable **IDBDATALOGLEVEL** or the HLAPI/UNIX database profile keyword **IDBDATALOGLEVEL** determines whether transactions in a transaction sequence are logged by the client interface.

The client interface creates a log file automatically if it does not already exist. Log entries are appended to the end of a log file. In order to prevent the log file from growing indefinitely, HLAPI/UNIX provides an archiving mechanism. The client interface records individual transactions in the log file specified by the database profile keyword **IDBLOGFILENAMEACTIVE** until the file reaches or exceeds the maximum file size specified by the database profile keyword **IDBLOGFILESIZE**. The active log file is then renamed to the file name specified by the database profile keyword **IDBLOGFILENAMEOLD**. If a previously archived log file of the same name exists, it is

deleted before the active log file is archived. Finally, a new active log file is created with the name specified by the database profile keyword **IDBLOGFILENAMEACTIVE**.

Multiple transaction sequences can use the same log file. However, if multiple transaction sequences attempt to record transactions in the same file, there will probably be contention for access to the file. To log a transaction, a client interface opens a log file, records the transaction, and closes the file. While the file is open, the process has exclusive write access to the file. If a process attempts to open the log file while another process has exclusive write access to it, the attempt to open the file fails. When this happens, the process that failed to open the file repeatedly attempts to open the file until the process opens the file successfully or reaches an internal HLAPI/UNIX retry limit. If the client interface fails to open the file because it reaches the retry limit, a return code and reason code are returned in the HICA to reflect the logging failure.

For more information about transaction logging by the client interface, see “Database Profile” on page 255 and “Environment Variables” on page 260.

Error Probe Logging by a Requester or Client Interface

A HLAPI/UNIX requester or client interface might encounter an error condition that cannot be explained with available information. When this happens, the requester or client interface records an entry in the HLAPI/UNIX probe log file **/usr/tmp/idbprobe.log** on the local UNIX host. If a probe log file does not exist, one is created.

HLAPI/UNIX imposes no limit on the maximum size of a probe log file. You can delete or rename a probe log file at any time. However, avoid using a probe log file in any way that would prevent HLAPI/UNIX from opening the file for exclusive write access.

When diagnosing a problem associated with a HLAPI/UNIX client application program, you might find entries in HLAPI/UNIX probe log files to be useful supplements to any transaction reply data that HLAPI/UNIX returns to the program.

29

The HLAPI/UNIX Requester

You can start a HLAPI/UNIX requester manually or automatically using any of the facilities provided by UNIX for issuing commands. The HLAPI/UNIX requester is started as a daemon task; it runs without a console in the background. Messages from the requester are written in the file **idbprobe.log** on the requester host.

Regardless of how you start a requester, the UNIX system file **/etc/services** on both the requester host and the client interface host must contain an entry associating the service name of the requester with the TCP/IP port for the requester. The UNIX system file **/etc/hosts** on the client interface host must also contain an entry that identifies the requester host name. See "Configuring HLAPI/UNIX and Associated Software" on page 241 for information on updating the **/etc/services** file.

Starting the Requester Manually

The syntax of the command to start a HLAPI/UNIX requester is:

```
<Requester_executable> [ -p|-P <System_Profile> ]
```

<Requester_executable>
: Specifies the name of the requester executable file. The requester executable program name is **idbreq**. On an AIX host, it is located in **/usr/lpp/idbhlapi/bin**. On an HP or Solaris host, it is located in ***/idbhlapi/bin**, where * is the directory specified at installation.

-p <System_Profile>
: Specifies a system profile to use when starting the requester. The value for **System_Profile** must be preceded by either **-p** or **-P**. At least one blank must separate the **-p** from the value for **System_Profile**. This parameter and value are optional. If they are not specified, defaults for the system profile keywords are used.

For example, if you start a requester with the following command:

```
/usr/lpp/idbhlapi/bin/idbreq -p idbsys.pro
```

The requester will search for the system profile **idbsys.pro** in the current working directory first. If the profile is not there, it will search in any directories in the search path specified by the environment variable **IDBSMPATH**.

Starting the Requester Automatically

You can place an entry in the **/etc/inittab** file to direct UNIX to start a HLAPI/UNIX requester when the **init** process runs. Refer to *AIX Version 4.2 Files Reference, How HP-UX Works: Concepts for the System Administrator* or *SunOS 5.3 Reference Manual* for more information.

Stopping a Requester

Once started, a requester runs indefinitely even if all of its conversations have ended. To stop a requester, you must first determine the process ID (pid) of the requester's highest-level process. Use the following command to do that:

```
ps -ef | grep idbreq
```

Once you know the process ID, issue the following command to stop the requester:

```
kill <pid>
```

If the requester does not stop, you can use the **KILL** command with the **-9** parameter, but you should consult the *AIX Version 4.2 Files Reference* and *How HP-UX Works: Concepts for the System Administrator* for possible consequences of using this command:

```
kill -9 <pid>
```

Diagnosing Some Common Problems

Symptom:
Changing the setting of a parameter in a database profile or a system profile has no impact on HLAPI/UNIX operation.

Possible Causes:

1. HLAPI/UNIX failed to locate your profile because it is not in the profile search path.

   Action:
   Specify the complete path for the profile or define the profile search path by setting the value of the appropriate HLAPI/UNIX environment variable. Make sure the value of the environment variable applies to the calling process. For information about environment variables, see "Environment Variables" on page 260.

2. HLAPI/UNIX used another profile that was ahead of your profile in the search path.

   Action:
   Delete one of the profiles, change the search path by changing the value of the appropriate HLAPI/UNIX environment variable, or specify the complete path name for the profile in the **DATABASE_PROFILE** control PDB.

3. A HLAPI/UNIX environment variable is overriding the value for the parameter.

   Action:
   Redefine the HLAPI/UNIX environment variable.

Symptom:
During an HL01 transaction, a client application process takes much longer than usual (typically 1-2 minutes) to return from the call to **IDBTransactionSubmit()**.

Possible causes:

1. Establishing a new APPC conversation between the requester and a server could cause the number of conversations to exceed a system limit for the number of conversations. Because the number of APPC conversations equals the system limit, new conversations are suspended until an existing conversation ends.

Action:
Increase the requester or server limit for the maximum number of APPC conversations.

Symptom:
Your client application process takes much longer than usual to run or it is rejected by the requester.

Possible causes:

1. If the conversation limit on the host is set too low, applications trying to start will have to wait until an HL01 is honored. This wait time may be significant, and nothing is returned to the waiting application indicating that the conversation limit was reached. When one of the applications already running finishes, the waiting application begins processing.

   Action:
   Ensure that the conversation limit is set high enough to run your applications.

2. The time between the client application's last transaction and the current transaction may have exceeded the allowable wait time specified by the database or the system profile. The client application is forced off the requester when the wait time is exceeded.

   Action:
   Ensure that the wait time set in your system profile is set high enough to run your applications.

30

HLAPI/UNIX Transactions

The work done by HLAPI/UNIX takes place through the use of HLAPI transaction sequences. Each transaction sequence begins with an HL01 transaction, optionally followed by any of the transactions listed in Table 1 on page 3, and ended by an HL02 transaction. Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* for information on the HLAPI transactions. This chapter explains aspects of transaction processing that are specific to the HLAPI/UNIX.

Validation of the Calling Process

When a client application program submits an HL01 transaction to begin a transaction sequence, HLAPI/UNIX records the process ID and the effective user ID of the process submitting the transaction. On all subsequent HLAPI/UNIX function calls of the transaction sequence, the process ID and effective user ID of the calling process must match those recorded for the transaction sequence. If they do not match, the HLAPI/UNIX returns an error to the calling process. This ensures that HLAPI/UNIX has the permissions necessary to access and free any HLAPI/UNIX resources that persist between calls to HLAPI/UNIX services.

CAUTION:

Do not modify a HICA or PDB associated with a transaction submitted to the HLAPI/UNIX until the transaction ends successfully or unsuccessfully. Any changes to a HICA or its associated PDBs during transaction processing causes unpredictable results.

Transaction Processing Modes

A HLAPI/UNIX client application program can select from two modes of transaction processing: synchronous and asynchronous. Unlike the Tivoli Information Management for z/OS Low-Level Application Programming Interface (LLAPI), which applies a transaction processing mode globally to all transactions in a session, the HLAPI/UNIX applies a transaction processing mode individually to each transaction.

Regardless of the transaction processing mode, a call to **IDBTransactionStatus()** returns the status and the reply (if available) for the most recently submitted transaction in the transaction sequence.

Synchronous Processing

Synchronous processing forces the current process in the application program to wait for a Tivoli Information Management for z/OS transaction to end before the process returns from a HLAPI/UNIX function call. The process cannot do any other work until the transaction is complete. Transaction completion includes both successful and unsuccessful outcomes.

To choose synchronous transaction processing, code a transaction type of **IDB_SYNC** on a call to **IDBTransactionSubmit()**. The process submitting the transaction will return from the call when the transaction is complete.

A transaction originally submitted in asynchronous mode can be changed to synchronous mode. To convert an asynchronous transaction to a synchronous transaction, code a query type of **IDB_WAITFORCOMPLETION** on a call to **IDBTransactionStatus()**. The process will return from the call when the transaction is complete.

You can implement a client application program using the multitasking capabilities of UNIX and the synchronous mode of transaction processing. By using multiple processes within an application, you can dedicate some processes to HLAPI/UNIX transaction processing while other processes do other tasks. In this case, only the dedicated processes are unavailable until their synchronous transactions complete.

Asynchronous Processing

Asynchronous transaction processing allows a client application program to submit a HLAPI/UNIX transaction and then continue doing other work while HLAPI/UNIX handles the transaction. When a process calls **IDBTransactionSubmit()** to submit a transaction asynchronously, the process returns from the call before the transaction runs to normal completion. The process can do other work while the transaction is in process. Some time later, the process can call **IDBTransactionStatus()** to determine whether the transaction is complete and to obtain the transaction reply.

To choose asynchronous transaction processing, code a transaction type of **IDB_ASYNC** on a call to **IDBTransactionSubmit()**. Later, call **IDBTransactionStatus()** to determine whether the transaction is complete. If the transaction is complete, the calling process will return immediately with the transaction reply. If the transaction is not complete, when the process will return depends on the value coded for the query type parameter of the call to **IDBTransactionStatus()**:

- If the value coded for the query type is **IDB_CHECKFORCOMPLETION**, the process returns immediately from the call to **IDBTransactionStatus()**.
- If the value coded for the query type parameter is **IDB_WAITFORCOMPLETION**, the process does not return from the call to **IDBTransactionStatus()** until the transaction is complete. Thus, the transaction that was originally submitted in asynchronous mode is converted to synchronous mode.

Transaction Concurrency Limitations

A client application program can use multiple HICA structures, each corresponding to a different transaction sequence. At any time, any of the transaction sequences can have a transaction in process. **However, no transaction sequence can have more than one incomplete transaction outstanding at any time.**

For any single requester-server conversation, transactions are processed serially on a first-come, first-served basis.

Data Conversion Characteristics

UNIX uses the ASCII character set. MVS uses the EBCDIC character set. These operating systems also represent DBCS data differently. For these reasons, data conversion is an essential part of data exchange between UNIX and MVS.

The client interface does all data conversion for HLAPI/UNIX. The AIX or HP system routine `iconv()` is used for the conversion. The database profile parameters **IDBCHARCODESET** and **IDBSERVCHARCODESET** specify which code sets to use for the conversion.

Specific to Solaris:
The HLAPI/Solaris does not support the **iconv** subroutine. Tivoli provides the translate tables to perform the necessary data conversion between the IBM-850 and IBM-037 code pages.

See “IDBCHARCODESET” on page 255 and “IDBSERVCHARCODESET” on page 258 for more information.

Data length validation is always done on the MVS host rather than the client interface or requester.

Code set conversions cannot be inverted when the mapping between characters in the code sets is not one-to-one. If a code set conversion cannot be inverted, certain characters will have a different representation after conversion from one code set to the other code set then back to the first code set.

Special DBCS Considerations

This section is relevant only for client application programs that use Double-Byte Character Set (DBCS) data. The HLAPI/Solaris does not support DBCS data.

DBCS data conversions do not preserve data length. Data length is not preserved because MVS uses shift-in and shift-out characters (X'0F' and X'0E', respectively) to delimit double-byte character strings, and UNIX does not. When DBCS data is converted from UNIX format to MVS format, shift-in and shift-out characters are inserted into the data. When DBCS data is converted from MVS format to UNIX format, shift-in and shift-out characters are removed from the data. These transformations cause the length of DBCS data to change during each conversion.

Transaction data is in MVS format when the data length is validated. Therefore, your UNIX client application program should allow for the insertion of shift-in and shift-out characters during conversion of DBCS data. Allow for two extra bytes per DBCS string: one byte to delimit the beginning of the string, and one byte to delimit the end of the string. The field length values received by HLAPI/UNIX are adjusted automatically to reflect changes in data length when shift-in and shift-out characters are inserted or removed from the data.

For all DBCS-enabled PDB fields other than freeform text data, the field length is increased or decreased to reflect the insertion of any shift-in and shift-out characters. If the new field length exceeds the maximum valid length for the field, the data is truncated.

Developing HLAPI/UNIX Client Applications

To use HLAPI/UNIX data structures and function calls, your application source files must include the C language-based header file **idbh.h**. You can optionally include **idbech.h** to declare named constants for HLAPI/UNIX return and reason codes. In addition, you must specify the HLAPI/UNIX runtime library as one of the application's shared libraries for dynamic linking. The default path and file names for the HLAPI/UNIX runtime libraries are:

Specific to AIX:
`/usr/lpp/idbhlapi/lib/libidb.a`

Specific to HP:
`*/idbhlapi/lib/libidb.sl`

Specific to Solaris:
`*/idbhlapi/lib/libidb.so`

where * is the directory where the HLAPI/UNIX is installed. Linking to the runtime library permits runtime access to the client interface services for **IDBTransactionSubmit()** and **IDBTransactionStatus()**.

To use these functions, your application program must do the following:

1. Allocate memory for HICA and PDB structures using the data types declared in the header file **idbh.h**.
2. Assign valid values to the fields of the structures.
3. Pass the structures and other arguments on calls to the HLAPI/UNIX functions.

Note: HLAPI/AIX also provides a REXX HLAPI/AIX interface which allows you to access HLAPI/AIX functions from AIX REXX/6000 programs in the same manner as HLAPI/REXX on MVS allows you to access the HLAPI from MVS REXX programs. See "Using the REXX HLAPI/AIX Interface" on page 283 for more information about REXX HLAPI/AIX.

There are differences between the HICA and PDB structure definitions for the HLAPI and the corresponding structure definitions for HLAPI/UNIX. Corresponding field names are generally similar. Some field names have been modified for HLAPI/UNIX to conform to standard C-language naming conventions.

The client APIs create child processes to complete each high-level API. If your client application registers an exit routine, the child processes will inherit the registration. If you do not want the children to execute the routine, wrap it with an *if* statement to ensure that the client process is performed. For example, if the pid is pidClient, you could check a global containing the pid with this code:

```
if (getpid()==pidClient)
 [
    /* your exit routine */
]
```

If your client is the top-level parent, you could use:

```
if (getpid()==(pid_t)1
 [
    /* your exit routine */
]
```

Including the HLAPI/UNIX Header File idbh.h

You must include the HLAPI/UNIX header file **idbh.h** in application source code that references HLAPI/UNIX data types and functions. This header file declares the HLAPI/UNIX data types and function prototypes your program needs to communicate with HLAPI/UNIX runtime services. Ensure that each source file's references to HLAPI/UNIX data types and functions fall within the scope of the include file's declarations. To include the header file **idbh.h** in a source file, put the following line into the source file:

```
#include  <idbh.h>    /* HLAPI/UNIX header file */
```

The default path and file names for these include files are:

Specific to AIX:
`/usr/lpp/idbhlapi/include/idbh.h`

Installation of HLAPI/AIX creates the symbolic link `/usr/include/idbh.h` to reference the include file.

Specific to HP:
`*/idbhlapi/include/idbh.h`

Installation of HLAPI/HP creates the symbolic link `/usr/include/idbh.h` to reference the include file.

Specific to Solaris:
`*/idbhlapi/include/idbh.h`

Installation of HLAPI/Solaris creates the symbolic link `/usr/include/idbh.h` to reference the include file.

where * is the directory where the HLAPI/UNIX is installed.

The compiler's default search path for header files is generally sufficient to access the HLAPI/UNIX include file. If the header file resides in a location other than the default or if the specified symbolic link is absent, then you may need to explicitly specify a search path to enable the compiler to locate the header file.

Refer to the file **idbappl.c** in the HLAPI/UNIX examples subdirectory (listed below) for an example of how to include the HLAPI/UNIX header files.

Specific to AIX:
`/usr/lpp/idbhlapi/examples`

Specific to HP:
`*/idbhlapi/examples`

Specific to Solaris:
`*/idbhlapi/examples`

You must specify the following definition on the **COMPILE** command to include the correct version of the data structure:

Specific to AIX:
`—D_AIXINFO`

Specific to HP:
`—D_HPINFO`

Specific to Solaris:
`−D_SUNINFO`

Including the HLAPI/UNIX Header File idbech.h

You can include the HLAPI/UNIX header file **idbech.h** in application source files to declare named constants defined for HLAPI/UNIX return and reason codes. This header file declares the named constants for most LLAPI, HLAPI, and HLAPI/UNIX return and reason codes used by HLAPI/UNIX. Ensure that each source file's references to the named constants fall within the scope of the include file's declarations. To include the header file **idbech.h** in a source file, put the following line into the source file:

```
#include  <idbech.h>    /* HLAPI/UNIX header file */
```

The default path and file names for these include files are:

Specific to AIX:
`/usr/lpp/idbhlapi/include/idbech.h`

Installation of HLAPI/AIX creates the symbolic link `/usr/include/idbech.h` to reference the include file.

Specific to HP:
`*/idbhlapi/include/idbech.h`

Installation of HLAPI/HP creates the symbolic link `/usr/include/idbech.h` to reference the include file.

Specific to Solaris:
`*/idbhlapi/include/idbech.h`

Installation of HLAPI/Solaris creates the symbolic link `/usr/include/idbech.h` to reference the include file.

Refer to the file **idbappl.c** in the HLAPI/UNIX examples subdirectory (listed below) for an example of how to include the HLAPI/UNIX header files.

Specific to AIX:
`/usr/lpp/idbhlapi/examples`

Specific to HP:
`*/idbhlapi/examples`

Specific to Solaris:
`*/idbhlapi/examples`

Note: The default path name for this include file is `/usr/lpp/idbhlapi/include/idbech.h`. However, installation of HLAPI/UNIX creates the symbolic link `/usr/include/idbech.h` to reference the include file. Therefore, the compiler's default search path for header files is generally sufficient to access the HLAPI/UNIX include file. If the header file resides in a location other than the default or if the specified symbolic link is absent, then you may need to explicitly specify a search path to enable the compiler to locate the header file.

Overview of HICA and PDB Data Structures

The primary data structures that your application uses to communicate with HLAPI/UNIX are HICA structures and PDB structures. The arguments of the HLAPI/UNIX functions **IDBTransactionSubmit()** and **IDBTransactionStatus()** include the address of a HICA that represents a transaction sequence. The fields of the HICA include pointers to PDB structures.

Each non-null PDB pointer of the HICA is a pointer to the first element of a linked list of PDBs of a particular type. There are five types of PDBs, hence five PDB pointers in a HICA:

- Control PDB
- Input PDB
- Output PDB
- Error PDB
- Message PDB

Each type of PDB can have subtypes. For example, some of the subtypes of control PDBs are **SPOOL_INTERVAL**, **HLIMSG_OPTION**, **HLAPILOG_ID**, **DATABASE_PROFILE**, **SECURITY_ID**, and **PASSWORD**.

To submit a transaction request to HLAPI/UNIX, your application program calls **IDBTransactionSubmit()** and passes the address of a HICA with associated control PDB and input PDB structures. The HL01 transaction must include the three required control PDBs, **SECURITY_ID**, **PASSWORD**, and **DATABASE_PROFILE**, which are described in “Initialize Tivoli Information Management for z/OS (HL01)” on page 278. The HLAPI/UNIX communicates the transaction reply to your application program by associating linked lists of output PDBs, error PDBs, and message PDBs with the HICA. In addition, HLAPI/UNIX updates the list of input PDBs for some transaction replies.

The values you store in the control PDBs and input PDBs depend on the specific HLAPI/UNIX transaction you want to use. For example, if a control PDB specifies that the transaction type is for record creation, then input PDBs may specify the data values for individual fields of the record to be created. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information about HICAs and PDBs.

Allocating and Initializing a HICA structure

Your application program must allocate and initialize one HICA structure for each HLAPI/UNIX transaction sequence. The HICA is used throughout the transaction sequence. Therefore, your application program must ensure that the storage allocated for the HICA persists for the duration of the transaction sequence. Your application program can allocate any class of storage for a HICA, such as shared memory, automatic storage, and static storage. This example shows an application fragment that illustrates the allocation and initialization of a HICA. Note that the ENVP field of a HICA must be set to null before each HL01 transaction and must not be changed by the application program during the remainder of the transaction sequence.

```
/***************************************************************/
/* allocate a HICA for a transaction sequence                  */
/***************************************************************/
   static IDB_HICA    MyHICA;     /* allocate HICA             */
/***************************************************************/
/* initialize a HICA before an HL01 transaction                */
/***************************************************************/

   memset( &MyHICA,               /* fill HICA with nulls      */
           '\0',
           sizeof(MyHICA)       );

   memcpy( MyHICA.ACRO,           /* initialize HICA eyecatcher */
           HICAACRO_TEXT,
           sizeof(MyHICA.ACRO)  );

   MyHICA.LENG = sizeof(MyHICA);  /* set HICA length            */
```

```
    MyHICA.ENVP = (void *)0;       /* initialize ENVP field --   */
                                   /* ONLY FOR HL01 TRANSACTION  */

 ⋮
                         /* associate Control PDBs     */

 ⋮
                         /* and Input PDBs with HICA   */
```

Allocating and Initializing a PDB Structure

Your application program is responsible for allocating, initializing, and freeing control PDBs and input PDBs for each transaction. HLAPI/UNIX is responsible for allocating, initializing, and freeing output PDBs, error PDBs, and message PDBs for each transaction.

Your application must allocate and initialize any required control PDBs and input PDBs before it submits a transaction request. Your application program should examine any output PDBs, error PDBs, and message PDBs that are returned with a transaction reply. In addition, your application program should examine the input PDBs for updates made by HLAPI/UNIX as part of certain transaction replies. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information.

The last field of a PDB structure is the data field. This field is declared to be a one-element array of unsigned characters, but it is actually a variable-length field. The actual length of the field is determined by the number of extra bytes allocated for the PDB structure. Your application program should use the data field to record the actual number of bytes of data in the data field. This arrangement allows your application program to conserve memory by allocating each PDB with only as much storage as it needs. The total size of a PDB is the sum of **PDBFIX_SIZE** (a constant defined in **idbh.h**) and the number of bytes for the data field.

This example shows an application fragment that illustrates allocation and partial initialization of an input PDB. Note the calculation of PDB size.

```
int      PDBsize;
PDB     *pPDB;                /* pointer to PDB           */

PDBsize = PDBFIX_SIZE + strlen("DOE JOHN");

pPDB = malloc( PDBsize );       /* allocate memory for PDB   */

if (pPDB) {

    memset( pPDB,             /* fill PDB with nulls      */
            '\0',
            PDBsize          );
    memcpy( pPDB->Acro,        /* initialize PDB eyecatcher  */
            PDBACRO_TEXT,
            sizeof(pPDB->Acro));

    memset( pPDB->Name,        /* initialize with blanks     */
            ' ',
            sizeof(pPDB->Name));
    memcpy( pPDB->Name,        /* record field value          */
            "REPORTER_NAME",
            strlen("REPORTER_NAME"));
    pPDB->Datl = strlen("DOE JOHN");  /* initialize length     */
    memcpy( pPDB->Data,              /* initialize value      */
            "DOE JOHN",
```

```
            pPDB->DATL           );

        pPDB->Proc = 'V';           /* request validation          */

        pPDB->Code = ' ';           /* initialize data error code  */
```

⋮

```
                    /* initialize other fields */

        pPDB->Next = MyHICA.INPP;   /* insert PDB into linked list  */
        pPDB->Prev = NULL;          /*   of Input PDBs              */
```

HLAPI/UNIX Functions

Two C-language HLAPI/UNIX functions are available to client application programs:

- **IDBTransactionSubmit()** for submitting a transaction request
- **IDBTransactionStatus()** for checking the status of a pending transaction and retrieving the transaction reply.

The HLAPI/UNIX header file **idbh.h** contains prototypes for the functions and declarations of the data types and values associated with the functions.

IDBTransactionSubmit()

A client application program calls the function **IDBTransactionSubmit()** to submit a transaction request to HLAPI/UNIX. In a C-language program, the call to **IDBTransactionSubmit()** looks like

```
rc = IDBTransactionSubmit(pHICA, SubmitMode);
```

Your application must provide the following variables:

- pHICA

  A pointer to a structure of the type HICA that contains the HICA that you want to submit to HLAPI/UNIX.

- SubmitMode

  Your selection of the processing mode for the transaction. The variable has a type definition of **TRANTYPE_TYPE**. The following values are valid:

  IDB_SYNC

  Specifies synchronous processing mode

  IDB_ASYNC

  Specifies asynchronous processing mode

HLAPI/UNIX returns a value from this function call that you should examine before looking at the HICA return and reason codes. This return code (rc) is a variable of type **IDBRC_TYPE**. The values that can be returned for it are listed in “HLAPI Service Call Return Codes” on page 367.

```
#include <idbh.h>
IDBRC_TYPE     rc;
HICA           MyHICA;
TRANTYPE_TYPE  Mode;
```

⋮

```
Mode = IDB_ASYNC;
```

```
rc = IDBTransactionSubmit( &MyHICA, Mode );
⋮
```

Usage Notes

After calling **IDBTransactionSubmit()** to submit a transaction, a client application program should not alter the HICA, PDBs, or other associated structures until the transaction is complete.

After submitting the HL01 transaction to initiate a transaction sequence, a client application need only change the control PDB and input PDB chains to prepare the HICA for a subsequent transaction. The HICA pointers to the output PDB, message PDB, and error PDB chains may be set optionally to NULL to prepare for a subsequent transaction.

Additional information about error conditions for a particular transaction may be obtained from the following:

- The values of the HICA fields **RETC** and **REAS**, if **IDBTransactionSubmit()** returned the value **IDBRC_XERR**
- Any error PDBs and message PDBs chained to the HICA
- Any input PDBs updated by HLAPI/UNIX
- The HLAPI/UNIX file **idbprobe.log**
- The requester and client interface log files, if transaction logging is enabled for the transaction sequence.
- “Transaction Logging by a RES and by an MRES Without Pre-started API Sessions” on page 20.

IDBTransactionStatus()

A client application program calls the function **IDBTransactionStatus()** to request the status and obtain the reply of an asynchronous transaction. **IDBTransactionStatus()** can also be used to convert an asynchronous transaction to a synchronous transaction. In a C-language program, the call to **IDBTransactionStatus()** looks like this:

```
rc = IDBTransactionStatus(pHICA, StatusMode, pTranStatus);
```

Your application must provide the following variables:

- pHICA

  A pointer to a structure of the type HICA that contains the HICA that you want to submit to HLAPI/UNIX.

- StatusMode

  An input parameter specifying whether the calling process will wait for the transaction reply if it is not yet available. The variable has a type definition of **QUERY_TYPE** and the following values are valid:

  IDB_CHECKFORCOMPLETION

  This value specifies that the caller is to return immediately from the call to **IDBTransactionStatus()** even if the transaction is not yet complete. If the transaction is complete, the transaction reply will be available to the caller upon return.

IDB_WAITFORCOMPLETION
This value specifies that the caller is to return from the call to **IDBTransactionStatus()** only when the transaction is complete. The transaction results will be available to the caller upon return.

- pTranStatus

  A pointer to a variable of type **TRANSTATUS_TYPE**. Upon return from the call to **IDBTransactionStatus**, the variable will contain one of the following values:

  IDB_TCOMPLETE
  The transaction is complete. Any output, error, and message PDBs returned from the MVS host are attached to the HICA and available for your program's use. The **RETC** and **REAS** fields of the HICA are set with values indicating the result of the transaction. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains a listing of possible values.

  IDB_TINPROGRESS
  The transaction is still in progress. No data has been returned on the HICA, and the HICA is unavailable for use by your application.

IDBTransactionStatus() returns a value from this call. Examine this value before looking at the HICA return and reason codes. This return code (rc) is defined as a variable of type **IDBRC_TYPE**. The values that can be returned for it are listed in "HLAPI Service Call Return Codes" on page 367.

```
#include <idbh.h>
IDBRC_TYPE        rc;
HICA              MyHICA;
QUERYTYPE_TYPE    StatusMode;
TRANSTATUS_TYPE   TranStatus;
⋮
StatusMode = IDB_WAITFORCOMPLETION;
rc = IDBTransactionStatus( &MyHICA, StatusMode, &TranStatus);
⋮
```

Usage notes

Additional information about error conditions for a particular transaction may be obtained from the following:

- The values of the HICA fields **RETC** and **REAS**, if **IDBTransactionStatus()** returned the value **IDBRC_XERR**
- Any error PDBs and message PDBs chained to the HICA
- Any input PDBs updated by HLAPI/UNIX
- The HLAPI/UNIX file **idbprobe.log**
- The requester and client interface log files, if transaction logging is enabled for the transaction sequence.
- "Transaction Logging by a RES and by an MRES Without Pre-started API Sessions" on page 20.

Using HLAPI/UNIX Functions in a Transaction Sequence

All HLAPI/UNIX function calls for a given transaction sequence must be made by the same process. In addition, the effective user ID of the process must not vary from one HLAPI/UNIX function call to the next. HLAPI/UNIX returns an error to the calling process if the process ID or effective user ID does not match the value recorded during the call to **IDBTransactionSubmit()** for the HL01 transaction of the transaction sequence.

The HLAPI transactions supported by HLAPI/UNIX are listed in Table 1 on page 3. For a description of the transactions supported by HLAPI/UNIX, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*. The remainder of this section discusses aspects of each transaction that are specific to HLAPI/UNIX.

Initialize Tivoli Information Management for z/OS (HL01)

The HLAPI/UNIX transaction HL01 requests a connection to a Tivoli Information Management for z/OS database on a specific Tivoli Information Management for z/OS server. The HL01 transaction initiates a HLAPI/UNIX transaction sequence. This section describes the steps performed for an HL01 transaction.

1. The user creates a HLAPI/UNIX database profile for the database connection. A text editor can be used to create or update a database profile. Different database connections can share the same database profile, but it is generally advisable to create a different database profile for each database connection. The database profile specifies the server (either symbolic destination name for a server that supports APPC or host and service names for a server that supports TCP/IP), the requester host and service names, the code sets used during data conversion, and other parameters governing the transaction sequence for the database connection. See "HLAPI/UNIX Profiles, Environment Variables, and Data Logging" on page 251 for a description of database profile contents.
2. The client application program allocates and initializes a HICA structure for the transaction sequence. The client application program allocates and initializes three required control PDBs:
   - The **DATABASE_PROFILE** PDB, specifying the name of the database profile for the transaction sequence
   - The **SECURITY_ID** PDB, specifying the MVS user ID
   - The **PASSWORD** PDB, specifying the MVS user password.

   The client application places these PDBs on the HICA's chain of control PDBs. The **DATABASE_PROFILE** PDB can go anywhere on the chain of control PDBs. If the **DATABASE_PROFILE** PDB uses a relative path for the database profile, the value of the HLAPI/UNIX environment variable **IDBDBPATH** is used to qualify the path name and locate the profile. A relative path does not explicitly specify a complete path from the root directory to the file.
3. The client application program calls **IDBTransactionSubmit()**, passing the address of the HICA as a parameter. During the call to **IDBTransactionSubmit()**, HLAPI/UNIX reads the database profile and records its parameters for use during the entire transaction sequence. The client interface then uses the requester host and service names to establish communication with the requester.
4. At this time, the specified requester must be available to the client interface.
5. The client interface transmits the MVS user ID, MVS password, and symbolic destination name or server host and server service name to the requester. If sharing is

enabled, the requester examines these values to determine whether they match the corresponding values for any existing conversation between the requester and a server. If a matching conversation is identified and the new transaction sequence can be assigned to the conversation without exceeding the implicit HLAPI/UNIX limit on the number of transaction sequences per conversation (10), then the requester assigns the new transaction sequence to the conversation. Otherwise, a new conversation is started and the new transaction sequence is assigned to it.

6. The **TIMEOUT_INTERVAL** PDB applies to the HLAPI that is running on Tivoli Information Management for z/OS. If you specify a timeout interval, it determines the maximum time that may elapse during HLAPI processing of a single transaction. If the HLAPI processing time exceeds the timeout interval, the HLAPI ends the transaction. Note that the HLAPI processing time for a transaction does not include time spent by the server or components of HLAPI/UNIX to process the transaction. Therefore, a transaction submitted from HLAPI/UNIX may appear to require more processing time than the timeout interval indicates.
7. For information about transaction logging, see “Transaction Logging” on page 261.
8. The client application program returns from the call to **IDBTransactionSubmit()**. If the transaction was submitted asynchronously, the client application program must call **IDBTransactionStatus()** to retrieve the transaction reply before submitting other transactions of the transaction sequence.

Any process of a client application program can have multiple, concurrent transaction sequences. The transactions of a transaction sequence can be submitted independently of the transactions of all other transaction sequences. However, a different HICA must be associated with each transaction sequence.

Terminate Tivoli Information Management for z/OS (HL02)

The HLAPI/UNIX transaction HL02 allows a client application program to close a database connection on a specific server. The HL02 transaction completes a normal HLAPI/UNIX transaction sequence. This section describes the steps performed for an HL02 transaction.

1. The client application program allocates and initializes the normal control PDBs requesting a disconnect from the database, associates the PDBs with the HICA for the transaction sequence, and calls **IDBTransactionSubmit()** to submit the HL02 transaction to HLAPI/UNIX.
2. The client interface communicates the disconnect request through the requester to the server. If this is the last transaction sequence assigned to the conversation between the requester and the server, the conversation with the server is ended. If this is not the last transaction sequence assigned to the conversation, the conversation is not ended.
3. The client application program returns from the call to **IDBTransactionSubmit()**. If the transaction was submitted asynchronously, the client application program must call **IDBTransactionStatus()** to retrieve the transaction reply. For an HL02 transaction, the transaction reply does not include any output PDBs, message PDBs, or error PDBs. However, the transaction reply does include the return and reason codes returned in the HICA fields **RETC** and **REAS**.

Retrieve Record (HL06)

The optional **TEXT_MEDIUM** control PDB can specify the type of storage medium for the HLAPI. However, the HLAPI/UNIX only supports storage medium type **B**. The

TEXT_MEDIUM PDB is optional for the HLAPI/UNIX HL06 transactions. However, HLAPI/UNIX overrides any specified value with the value for storage medium type **B**.

If you want to retrieve freeform text as a continuous stream of data with carriage return / line feed characters (ASCII X'0D0A') after each text line, set the optional control PDB **TEXT_STREAM** to **YES**. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information about the **TEXT_STREAM** PDB.

Create Record (HL08)

HLAPI/UNIX does not support text data sets. Always specify a non-zero value for the **PDB_DATW** field of the input PDBs for text data.

If you are creating a record that contains freeform text, and the input text contains either line feed characters (ASCII X'0A') or carriage return / line feed characters (ASCII X'0D0A'), set the optional control PDB **TEXT_STREAM** to **YES**. This will ensure that text formatting information is stored in the record. When the text is retrieved, it will be formatted exactly as it was entered. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information about the **TEXT_STREAM** PDB.

Update Record (HL09)

HLAPI/UNIX does not support text data sets. Always specify a non-zero value for the **PDB_DATW** field of the input PDBs for text data.

If you are updating a record that contains freeform text, and the input text contains either line feed characters (ASCII X'0A') or carriage return / line feed characters (ASCII X'0D0A'), set the optional control PDB **TEXT_STREAM** to **YES**. This will ensure that text formatting information is stored in the record. When the text is retrieved, it will be formatted exactly as it was entered. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information about the **TEXT_STREAM** PDB.

Linking Your Application to HLAPI/UNIX Services

The HLAPI/UNIX shared runtime library contains the client interface runtime services accessed by calling the HLAPI/UNIX functions **IDBTransactionSubmit()** and **IDBTransactionStatus()**. Before using the runtime services of HLAPI/UNIX, you must link your application program to the HLAPI/UNIX shared runtime library. The default path and file names for the libraries are:

Specific to AIX:
`/usr/lpp/idbhlapi/lib/libidb.a`

Specific to HP:
`*/idbhlapi/lib/libidb.sl`

Specific to Solaris:
`*/idbhlapi/lib/libidb.so`

where * is the directory where the HLAPI/UNIX is installed.

However, installation of HLAPI/UNIX creates a symbolic link in the **/usr/lib** directory to reference the shared library.

When starting the link editor, specify the option

```
-lidb
```

to enable dynamic linking with the HLAPI/UNIX shared runtime library.

You may need to explicitly specify a search path to enable the link editor to locate the library if it resides in a location other than the default, if the specified symbolic link is absent, or if you are not starting the link editor through cc.

Refer to the file **idbappl.mak** in the HLAPI/UNIX examples subdirectory (listed below) for an example of how to link your application with the runtime library.

Specific to AIX:
`/usr/lpp/idbhlapi/examples`

Specific to HP:
`*/idbhlapi/examples`

Specific to Solaris:
`*/idbhlapi/examples`

Planning Your HLAPI/UNIX Application

This section lists some questions you need to answer when you plan and design your application that uses HLAPI/UNIX. Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* for more information on this subject.

- Which Tivoli Information Management for z/OS transactions (for example, create or update) do you use?
- Which record types (for example, problem or change) do you use?
- Which fields (for example, problem status or assignee name) do you use?
- Do you need to connect to more than one Tivoli Information Management for z/OS database?
- On which MVS systems are your Tivoli Information Management for z/OS databases located?
- Which communication protocol, APPC or TCP/IP, will your application use?
- Does your application require security?
- **Specific to AIX:**
  Do you want the application to be served by a dedicated address space (RES) or do you want to share an address space with other applications (MRES with APPC or MRES with TCP/IP)?
- **Specific to AIX:**
  Do you have symbolic destination names defined for the servers that support APPC?
- Do you have port numbers assigned for the sockets any servers that support TCP/IP will need?
- On which UNIX systems will your client application programs run?
- On which UNIX systems will your requesters be located?

- How should your UNIX system resource limits be configured to support requester and client interface requirements?
- Will the transactions be submitted synchronously or asynchronously?
- How are the processes of your application related to one another?
- How much storage do you need in a server to support sessions, given your transaction mix?
- Do you want to enable transaction logging by the client interface or by the server?
- Do you want to perform data validation? Data length validation is always performed for a transaction, but other validation can be controlled as described in the *Tivoli Information Management for z/OS Application Program Interface Guide*.

Converting HLAPI Programs to HLAPI/UNIX Programs

If you want to convert an existing C-language program that uses HLAPI to a C-language program that uses HLAPI/UNIX, here are some tips on how to do that:

- Make general modifications that are required to make the program run on UNIX:
  - If necessary, fix brackets ('{','}','[',']') after converting the program source code from EBCDIC (the MVS format) to ASCII (the UNIX format). Brackets and other characters may be translated incorrectly when a file is transferred from MVS to UNIX.
  - Change HICA and PDB field names to their HLAPI/UNIX counterparts.
- Update references to included header files:
  - Include the HLAPI/UNIX header file **idbh.h**.
  - Include the HLAPI/UNIX header file **idbech.h** if you want to use the named constants for HLAPI/UNIX return and reason codes.
  - Do not include the HLAPI header file **spc.h**.
- Delete compiler pragmas used by the MVS program.
- Convert any HL06, HL08, and HL09 transactions that use the data set method of freeform text processing to the buffer method of freeform text processing.
- Convert HLAPI function calls to the HLAPI/UNIX function calls **IDBTransactionSubmit()** and **IDBTransactionStatus()**. Do not define variables to reference the **BLGYHLPI** module, and do not fetch the **BLGYHLPI** module.
- Add processing to allocate and initialize the three special types of control PDBs for HLAPI/UNIX (**SECURITY_ID**, **PASSWORD**, and **DATABASE_PROFILE**) and to insert the PDBs into the chain of control PDBs for each HL01 transaction.
- Create one or more database profiles for use with your program's transaction sequences.
- Create any system profiles needed for requesters.
- Review the error handling sections of the program to determine whether changes are needed to process HLAPI/UNIX error conditions.
- When starting the compiler and the link editor, compile with appropriate options for locating the HLAPI/UNIX header files and for linking dynamically to the HLAPI/UNIX shared runtime library.

Using the REXX HLAPI/AIX Interface

The REXX HLAPI/AIX interface enables you to access HLAPI/AIX transactions from AIX REXX/6000 programs similar to the manner in which HLAPI/REXX on MVS enables you to access HLAPI transactions from MVS REXX programs. You should be familiar with HLAPI/REXX which is described in the *Tivoli Information Management for z/OS Application Program Interface Guide* before you attempt to use the REXX HLAPI/AIX interface.

REXX HLAPI/AIX allows you to write a REXX program on AIX that sets REXX variables with control and input information, then links to the HLAPI/AIX through the REXX HLAPI/AIX interface to process that information. On return, REXX HLAPI/AIX uses data returned by the HLAPI/AIX to set various REXX output variables in your program. The particular transactions that the REXX HLAPI/AIX interface supports are the same as those supported by the HLAPI/AIX. For a list of those transactions, see Table 4 on page 284.

The use of shared REXX variables for specifying control and input data to Tivoli Information Management for z/OS and returning output data from Tivoli Information Management for z/OS is equivalent to how this is implemented for HLAPI/REXX on MVS. Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* for information on how to define REXX variables in your program and for a list of reserved REXX variables that the REXX HLAPI interfaces (HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, and REXX HLAPI/USS) use.

The remainder of this section discusses the operating differences between HLAPI/REXX and REXX HLAPI/AIX.

REXX HLAPI/AIX Installation and Setup

The REXX HLAPI/AIX interface is a REXX external function package that you call from your REXX program. The REXX HLAPI/AIX is named blmyrxm and is distributed with the client interface option of the HLAPI/AIX. After installation of the HLAPI/AIX client interface, blmyrxm is located in /usr/lpp/idbhlapi/lib. A sample REXX program is also distributed with the client interface and is located in /usr/lpp/idbhlapi/examples. It is named blmyrxsa. “Installing and Setting Up HLAPI/UNIX” on page 237 contains information on HLAPI/AIX installation.

When you run the REXX HLAPI/AIX interface, you must make sure that blmyrxm is located in any directory in your LIBPATH, so you may need to copy blmyrxm to a directory in your LIBPATH or include /usr/lpp/idbhlapi/lib in your LIBPATH. You should also make certain that the REXX shared library librexx.a is located in any directory in your LIBPATH.

REXX HLAPI/AIX Software Requirement

AIX REXX/6000 (commonly referred to as REXX) must be installed on the same AIX system as the REXX HLAPI/AIX.

Invoking REXX HLAPI/AIX

REXX HLAPI/AIX is an external function package that you invoke from your AIX REXX/6000 program. Because the REXX HLAPI/AIX is an external function package, your program must first register it with REXX before you call it.

You use the SysAddFuncPkg system-defined function in your REXX program to register the REXX HLAPI/AIX. The syntax is as follows:

```
call SysAddFuncPkg 'blmyrxm'
```

After calling SysAddFuncPkg to register the REXX HLAPI/AIX, you can test the REXX variable **RESULT** for a value of 0 to make sure that the code was registered successfully.

You deregister the REXX HLAPI/AIX at the end of your REXX program using **SysDropFuncPkg** as follows:

```
call SysDropFuncPkg 'blmyrxm'
```

In addition, before you run your REXX program, you must ensure that blmyrxm and the REXX shared library (librexx.a) are located in any directory in your LIBPATH.

After the REXX HLAPI/AIX is registered, your REXX program can call it. The syntax of the REXX HLAPI/AIX call is:

```
call blmyrxm transaction-name,{control},{input},{output}
```

The following example illustrates a call to the REXX HLAPI/AIX for a create transaction with an input stem name of **INPUT**, a control stem name of **CONTROL** and an output stem name of **OUTPUT**.

```
call blmyrxm 'CREATE','CONTROL','INPUT','OUTPUT'
```

REXX HLAPI/AIX Transaction Names

The following REXX HLAPI/AIX transaction names are supported. The list matches the HLAPI/AIX transaction subset which is specified in Table 1 on page 3.

Table 4. REXX HLAPI/AIX Transaction Names

NAME	FUNCTION
INIT	Initialize Tivoli Information Management for z/OS
TERM	Terminate Tivoli Information Management for z/OS
GETID	Obtain External Record ID
CHECKOUT	Check Out Record
CHECKIN	Check In Record
RETRIEVE	Retrieve Record
CREATE	Create Record
UPDATE	Update Record
CHANGE_APPROVAL	Change Record Approval
SEARCH	Record Inquiry
ADD_REL	Add Record Relations
DELETE	Delete Record
USERTSP	Start User TSP
GETDATAMODEL	Get Data Model

Running Your AIX REXX/6000 Program

To run your REXX program, you can use either of the following methods:

- At your shell command prompt, type **rexx name** where **name** is the name of your REXX program. This method explicitly starts the REXX/AIX language interpreter and identifies the REXX program to be run.
  OR

- If the first line of your REXX program contains a “magic number” and identifies the directory where the REXX/AIX interpreter resides, your REXX program can be run by simply typing its name at the shell command prompt. For example, if the REXX/AIX interpreter file is in the /usr/bin directory include the following as the very first line of your REXX program starting in column 1:

```
#! /usr/bin/rexx
```

  When you have finished creating your REXX program source file and wish to run it for the first time, make it executable using the **chmod +x name** command. Then run the REXX program by typing its filename at your shell command prompt.

REXX Reserved Variables

The REXX HLAPI/AIX interface uses the same REXX reserved variables as those described for HLAPI/REXX in the *Tivoli Information Management for z/OS Application Program Interface Guide* with the following additions and deletions:

Table 5. REXX reserved variables added for REXX HLAPI/AIX

BLG_HLAPIAIX_RC	Return code passed back from HLAPI/AIX. The return codes are documented in “HLAPI Service Call Return Codes” on page 367.
BLG_REXXVAR_POOL_RC	Set to return code from REXX variable pool service on failures.

Table 6. HLAPI/REXX reserved variables not used by REXX HLAPI/AIX

BLG_R15	replaced by BLG_HLAPIAIX_RC
BLG_IRXEXCOM_RC	replaced by BLG_REXXVAR_POOL_RC
RC	replaced by RESULT

Other Considerations

- REXX HLAPI/AIX does not support asynchronous processing; all transactions are processed synchronously. Synchronous processing forces the REXX program’s current process to wait for a Tivoli Information Management for z/OS transaction to finish before it can perform any other work.
- REXX HLAPI/AIX requires three additional control variables on the **INIT** transaction. The REXX variable names are:
  - SECURITY_ID
  - PASSWORD
  - DATABASE_PROFILE

  Use **SECURITY_ID** to specify the MVS user ID. Use **PASSWORD** to specify the MVS password for the user ID. Use **DATABASE_PROFILE** to specify the name of the database profile.
- Neither HLAPI/AIX nor REXX HLAPI/AIX supports text data sets.
  - For a **RETRIEVE** transaction, the REXX variable **TEXT_MEDIUM** only supports storage medium type **B**.
  - For a **CREATE** or **UPDATE** transaction, **text-name.?width** for text data must be nonzero or not be specified.

Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* for information on how to define REXX variables in your program and for a list of reserved REXX variables that the REXX HLAPI interfaces use.

REXX HLAPI/AIX Sample REXX Program

A sample REXX program named blmyrxsa is distributed with the client interface option of the HLAPI/AIX. After installation, blmyrxsa is located in /usr/lpp/idbhlapi/examples. This sample REXX program illustrates how to:

- register the REXX HLAPI/AIX interface with REXX
- setup REXX variables
- make REXX HLAPI/AIX transaction calls
- retrieve output data
- deregister the REXX HLAPI/AIX interface

"Sample Program BLMYRXSA" also shows the **blmyrxsa** sample REXX program. The following steps are performed:

1. Register the REXX HLAPI/AIX interface using the **SysAddFuncPkg** system-defined function.
2. Setup REXX variables for an **INIT** transaction.
3. Call REXX HLAPI/AIX to perform the **INIT** transaction.
4. Record REXX HLAPI/AIX output to an AIX file named **blmyrxsa.out**.
5. Setup REXX variables to **CREATE** a record with record id **SAMP1**.
6. Call REXX HLAPI/AIX to perform the **CREATE** transaction.
7. Record REXX HLAPI/AIX output to the AIX file named **blmyrxsa.out**.
8. Setup REXX variables to retrieve the record just created.
9. Call REXX HLAPI/AIX to perform the **RETRIEVE** transaction.
10. Record REXX HLAPI/AIX output to the AIX file named **blmyrxsa.out**.
11. Setup REXX variables to delete the record just created.
12. Call REXX HLAPI/AIX to perform the **DELETE** transaction.
13. Record REXX HLAPI/AIX output to the AIX file named **blmyrxsa.out**.
14. Call REXX HLAPI/AIX to perform the **TERM** transaction.
15. Deregister the REXX HLAPI/AIX interface using the **SysDropFuncPkg** system-defined function.

Sample Program BLMYRXSA

The sample program **blmyrxsa** demonstrates calls to the REXX HLAPI/AIX interface.

```
#! /usr/bin/rexx
/*                                                                  */
/*------------------------------------------------------------------*/
/*                                                                  */
/*              Licensed Materials - Property of IBM.               */
/*                                                                  */
/*                                                                  */
/*                            5697-SD9                              */
/*                                                                  */
```

```
/*               (C) Copyright IBM Corp. 1981, 2001                  */
/*                                                                   */
/*                    See Copyright Instructions                     */
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*-------------------------------------------------------------------*/
/*                                                                   */
/* This sample REXX program demonstrates calls to the REXX HLAPI/AIX */
/* interface.  The program shows how:                                */
/*                                                                   */
/*  - SysAddFuncPkg() is used to register the REXX HLAPI/AIX         */
/*                                                                   */
/*  - REXX HLAPI/AIX variables are set for transactions              */
/*                                                                   */
/*  - INIT, CREATE, RETRIEVE, DELETE and TERM REXX HLAPI/AIX         */
/*    transactions are issued                                        */
/*                                                                   */
/*  - SysDropFuncPkg() is used to deregister the REXX HLAPI/AIX      */
/*                                                                   */
/*  - a log of transaction output is kept in file BLMYRXSA.OUT       */
/*                                                                   */
/*-------------------------------------------------------------------*/


'rm blmyrxsa.out'

/********************************************/
/* call SysAddFuncPkg to register blmyrxm   */
/********************************************/
call SysAddFuncPkg 'blmyrxm'
if RESULT=0 then
 do
  /*******************************************/
  /* set the CONTROL data for the INIT       */
  /*******************************************/
  CONTROL.0 = 10
  CONTROL.1 = 'database_profile'
  CONTROL.2 = 'privilege_class'
  CONTROL.3 = 'session_member'
  CONTROL.4 = 'application_id'
  CONTROL.5 = 'class_count'
  CONTROL.6 = 'hlimsg_option'
  CONTROL.7 = 'security_id'
  CONTROL.8 = 'password'
  CONTROL.9 = 'spool_interval'
  CONTROL.10 = 'apimsg_option'
  privilege_class = 'MASTER'
  session_member  = 'BLGSES00'
  application_id  = 'SAMPID'
  class_count     = 1
  spool_interval  = 200
  hlimsg_option   = 'C'
  apimsg_option   = 'C'
  security_id     = 'SAMPID'
  password        = 'PASSWORD'
  database_profile = 'database.pro'
  /*******************************************/
  /* call REXX HLAPI/AIX to perform the      */
  /* INIT transaction                        */
  /*******************************************/
  Call blmyrxm 'INIT','CONTROL'

  /*******************************************/
  /* record REXX HLAPI/AIX output in file    */
  /* blmyrxsa.out                            */
  /*******************************************/
```

```
say 'Returned from INIT with Result = ' Result
rc = lineout('blmyrxsa.out','INIT Transaction results:')
rc = lineout('blmyrxsa.out','Result='  Result)
rc = lineout('blmyrxsa.out','BLG_RC='  BLG_RC)
rc = lineout('blmyrxsa.out','BLG_REAS='  BLG_REAS)
rc = lineout('blmyrxsa.out','BLG_VARNAME='  BLG_VARNAME)
rc = lineout('blmyrxsa.out','BLG_HLAPIAIX_RC=' BLG_HLAPIAIX_RC)
rc = lineout('blmyrxsa.out',' ')
say '

if Result \= 0 then
  do
   call blmyrxm 'TERM'
   call SysDropFuncPkg 'blmyrxm'
   exit
  end

/*****************************************/
/* set the CONTROL and INPUT data for     */
/* the CREATE                             */
/*****************************************/
CONTROL = '
CONTROL.0 =  1
CONTROL.1 = 'PIDT_NAME'
PIDT_NAME = 'BLGYPRC'

INPUT = '
INPUT.0 = 6
INPUT.1.?NAME = 'S0BEE'
INPUT.1.?PROC = 'V'
INPUT.2.?NAME = 'S0B59'
INPUT.3.?NAME = 'S0CA9'
INPUT.4.?NAME = 'S0E0F'
INPUT.5.?NAME = 'S0CCF'
INPUT.6.?NAME = 'S0E01.'
S0E01.?WIDTH = 20
S0BEE = 'INITIAL'
S0B59 = 'DOE/JOHN'
S0CA9 = 'LPT1'
S0E0F = 'PROBLEM RECORD CREATE BY REXX HLAPI/AIX'
S0CCF = 'SAMP1'
S0E01.0  = 2
S0E01.1  = 'Sample1 first line'
S0E01.2  = 'Sample1 second line'
OUTPUT.0 = 1
OUTPUT.1.?TYPE = ' '

/*****************************************/
/* call REXX HLAPI/AIX to perform the     */
/* CREATE transaction                     */
/*****************************************/
Call blmyrxm 'CREATE','CONTROL','INPUT','OUTPUT'

/*****************************************/
/* record REXX HLAPI/AIX output in file   */
/* blmyrxsa.out                           */
/*****************************************/
say 'Returned from CREATE with Result = ' Result
rc = lineout('blmyrxsa.out','CREATE Transaction results:')
rc = lineout('blmyrxsa.out','Result='  Result)
rc = lineout('blmyrxsa.out','BLG_RC='  BLG_RC)
rc = lineout('blmyrxsa.out','BLG_REAS='  BLG_REAS)
rc = lineout('blmyrxsa.out','BLG_VARNAME='  BLG_VARNAME)
rc = lineout('blmyrxsa.out','BLG_HLAPIAIX_RC=' BLG_HLAPIAIX_RC)
if BLG_ERRCODE.0 \= 0 then
  do
    rc = lineout('blmyrxsa.out','REXX Error Variables')
```

```
    do i = 1 to BLG_ERRCODE.0
     rc = lineout('blmyrxsa.out','Error Name ' i ' = ' BLG_ERRCODE.i.?NAME)
     rc = lineout('blmyrxsa.out','Error Code ' i ' = ' BLG_ERRCODE.i.?CODE)
    end
  end
if BLG_MSGS.0 \= 0 then
  do
    rc = lineout('blmyrxsa.out','Messages')
    do i = 1 to BLG_MSGS.0
     rc = lineout('blmyrxsa.out','Message ' i ' = ' BLG_MSGS.i.?NAME)
    end
  end
rc = lineout('blmyrxsa.out',' ')
say '
if Result \= 0 then
  do
   call blmyrxm 'TERM'
   call SysDropFuncPkg 'blmyrxm'
   exit
  end

/*******************************************/
/* set the CONTROL data for the RETRIEVE  */
/*******************************************/
CONTROL = '
CONTROL.0 =  3
CONTROL.1 = 'PIDT_NAME'
CONTROL.2 = 'TEXT_OPTION'
CONTROL.3 = 'RNID_SYMBOL'

PIDT_NAME = 'BLGYPRR'
TEXT_OPTION = 'YES'
RNID_SYMBOL = 'SAMP1'

/*******************************************/
/* call REXX HLAPI/AIX to perform the     */
/* RETRIEVE transaction                   */
/*******************************************/
call blmyrxm 'RETRIEVE','CONTROL',,'OUTPUT'
/*******************************************/
/* record REXX HLAPI/AIX output in file   */
/* blmyrxsa.out                           */
/*******************************************/
say 'Returned from RETRIEVE with Result = ' Result
rc = lineout('blmyrxsa.out','RETRIEVE Transaction results:')
rc = lineout('blmyrxsa.out','Result='  Result)
rc = lineout('blmyrxsa.out','BLG_RC='  BLG_RC)
rc = lineout('blmyrxsa.out','BLG_REAS='  BLG_REAS)
rc = lineout('blmyrxsa.out','BLG_VARNAME='  BLG_VARNAME)
rc = lineout('blmyrxsa.out','BLG_HLAPIAIX_RC=' BLG_HLAPIAIX_RC)
rc = lineout('blmyrxsa.out','REXX Output Variables')
do i = 1 to OUTPUT.0
 otype = OUTPUT.i.?TYPE
 tname = OUTPUT.i.?NAME
 rc = lineout('blmyrxsa.out','Name- ' tname '  Type- ' otype)
 /*******************************************/
 /* if the output is freeform text, put   */
 /* out all the lines                     */
 /*******************************************/
 if otype = 'X' then
  do j= 1 to OUTPUT.tname.0
   rc = lineout('blmyrxsa.out','Data- ' OUTPUT.tname.j)
  end
 else
  rc = lineout('blmyrxsa.out','Data- ' OUTPUT.tname)
end
if BLG_MSGS.0 \= 0 then
```

30. HLAPI/UNIX Transactions

```
      do
        rc = lineout('blmyrxsa.out','Messages')
        do i = 1 to BLG_MSGS.0
         rc = lineout('blmyrxsa.out','Message ' i ' = ' BLG_MSGS.i.?NAME)
        end
      end
    rc = lineout('blmyrxsa.out',' ')
    say '

    if Result \= 0 then
      do
       call blmyrxm 'TERM'
       call SysDropFuncPkg 'blmyrxm'
       exit
      end

    /*******************************************/
    /* set the CONTROL data for the DELETE     */
    /*******************************************/
    CONTROL = '
    CONTROL.0 =  1
    CONTROL.1 = 'RNID_SYMBOL'

    RNID_SYMBOL = 'SAMP1'

    /*******************************************/
    /* call REXX HLAPI/AIX to perform the      */
    /* DELETE transaction                      */
    /*******************************************/
    Call blmyrxm 'DELETE','CONTROL'
    /*******************************************/
    /* record REXX HLAPI/AIX output in file    */
    /* blmyrxsa.out                            */
    /*******************************************/
    say 'Returned from DELETE with Result = ' Result
    rc = lineout('blmyrxsa.out','DELETE Transaction results:')
    rc = lineout('blmyrxsa.out','Result='  Result)
    rc = lineout('blmyrxsa.out','BLG_RC='  BLG_RC)
    rc = lineout('blmyrxsa.out','BLG_REAS='  BLG_REAS)
    rc = lineout('blmyrxsa.out','BLG_VARNAME='  BLG_VARNAME)
    rc = lineout('blmyrxsa.out','BLG_HLAPIAIX_RC=' BLG_HLAPIAIX_RC)
    say '

    /*******************************************/
    /* call REXX HLAPI/AIX to perform the      */
    /* TERM transaction                        */
    /*******************************************/
    Call blmyrxm 'TERM'

    /*********************************************/
    /* call SysDropFuncPkg to deregister blmyrxm*/
    /*********************************************/
    call SysDropFuncPkg 'blmyrxm'
   end

  Exit
```

31

Introduction to HLAPI/USS

Tivoli Information Management for z/OS supports local and remote access from an application program that runs on OS/390 UNIX System Services. It does this through the High-Level Application Program Interface (HLAPI) and the Tivoli Information Management for z/OS HLAPI client for OS/390 UNIX System Services (HLAPI/USS).

The HLAPI/USS provides local and remote access to Tivoli Information Management for z/OS data and data manipulation services. It consists of three parts:

- A Tivoli Information Management for z/OS MRES with TCP/IP *server*, an MVS-based transaction program that resides on the MVS host system. It provides the link between Tivoli Information Management for z/OS and the HLAPI/USS system.
- The Tivoli Information Management for z/OS HLAPI/USS *requester*, a USS-based transaction program that provides access to the HLAPI through a Tivoli Information Management for z/OS MRES with TCP/IP server.
- The Tivoli Information Management for z/OS HLAPI/USS *client interface*, a USS-based shared library and bindings for the C language.

HLAPI/USS also provides a REXX HLAPI/USS feature that provides access to HLAPI/USS from REXX programs.

Like the HLAPI, the HLAPI/USS is a transaction-based application programming interface. User application programs interact with Tivoli Information Management for z/OS from the OS/390 UNIX System Services environment in basically the same way as they do from MVS using the HLAPI. User application programs can be local, running under OS/390 UNIX System Services on the same MVS host as Tivoli Information Management for z/OS, or they can be on a remote MVS host. These local or remote environment user application programs can be thought of as the *clients* to the Tivoli Information Management for z/OS *server*. The OS/390 UNIX System Services environment offers a subset of HLAPI transactions, which are listed in Table 1 on page 3 and described in the *Tivoli Information Management for z/OS Application Program Interface Guide*.

The HLAPI/USS enables application programmers to write applications for use in their specific work environment.

Server Overview

A Tivoli Information Management for z/OS server is an MVS/ESA transaction program that handles communication between a HLAPI/USS requester and a Tivoli Information Management for z/OS database that resides on the MVS system where the server is installed. An OS/390 UNIX System Services client application program must use the HLAPI/USS

client interface to access a server through a HLAPI/USS requester. The HLAPI/USS requester only supports TCP/IP communication protocol, so it can only access the MRES with the TCP/IP server.

An MRES with TCP/IP server must be installed and available on every MVS/ESA machine with a Tivoli Information Management for z/OS database that HLAPI/USS needs to access. See "Configuring and Running a Multiclient Remote Environment Server (MRES) with TCP/IP" on page 53 for information about installing the server.

Requester Overview

The requester receives information from the client application program through the client interface and transfers the information to the appropriate server. It receives information from the server and transfers the information back to the client application program through the client interface. A requester can communicate with multiple servers and multiple client interfaces.

A HLAPI/USS requester includes the following components:

- A daemon program that runs on a requester host to serve as a communication link between a server on MVS and a client interface on OS/390 UNIX System Services. The communication link from the HLAPI/USS requester to the server uses TCP/IP communication protocol; therefore, the HLAPI/USS requester has access to only MRES with TCP/IP servers.

  A requester and a client interface communicate using TCP/IP sockets. Each requester can communicate with multiple client interfaces on multiple hosts, and each client interface can communicate with multiple requesters on multiple requester hosts. The requester need not be located on the same requester host as a client interface that uses the requester. The requester must be running before a client application program accesses it.
- An optional system profile that specifies parameters affecting the requester's function.

You can run one or more requesters on a single OS/390 UNIX System Services host, and you can run requesters on different OS/390 UNIX System Services hosts. The requesters can access the same MRES with TCP/IP or different MRESs with TCP/IP. A single requester can also access different MRESs with TCP/IP.

Client Interface Overview

The HLAPI/USS client interface transfers information from the client application program to the requester. It also transfers information it receives from the requester back to the client application program.

The HLAPI/USS client interface uses C programming language bindings. The client interface includes the following components:

- A C-language header file **blmh.h** that must be included by applications that use HLAPI/USS services. This file contains the declarations for the function and data structures that application programs must use to communicate with Tivoli Information Management for z/OS through HLAPI/USS.
- An optional C-language header file **blmech.h** that contains named constants representing HLAPI, LLAPI, and HLAPI/USS return and reason codes used by HLAPI/USS.

- A shared runtime library **blmhlapi** that application programs using HLAPI/USS services link to dynamically. This library contains the entry point and executable code for the client interface function.
- A definition side-deck **blmhlapi.x** that contains the import function for the client interface and must be specified when you compile and link your client application.
- One or more database profiles, each of which specifies parameters that apply to entire sequences of transactions.

The client interface communicates with a requester using TCP/IP protocol. A client interface can communicate with multiple requesters. The requester must be running before the client interface can access it.

HLAPI/USS also provides a REXX HLAPI/USS interface. The REXX HLAPI/USS interface enables you to access HLAPI/USS functions from REXX programs. See “Using the REXX HLAPI/USS Interface” on page 329 for more information about REXX HLAPI/USS.

Communication Overview

The HLAPI/USS requester and a Tivoli Information Management for z/OS server communicate using TCP/IP protocol, so a HLAPI/USS requester can only use an MRES with TCP/IP. Each requester can communicate with multiple MRES with TCP/IP servers on multiple MVS hosts, and each MRES with TCP/IP server can communicate with multiple requesters on multiple requester hosts.

A requester and a client interface communicate using sockets. Each requester can communicate with multiple client interfaces on multiple hosts, and each client interface can communicate with multiple requesters on multiple hosts.

A requester need not be located on the same host as a client interface using that requester.

Basic Transaction Flow

A ***transaction sequence*** is a series of HLAPI/USS transactions that begins with an initialize Tivoli Information Management for z/OS transaction (HL01), followed by other supported transactions in any order and ends with a terminate Tivoli Information Management for z/OS transaction (HL02). Client application programs submit transactions in a transaction sequence, which is referred to as a logical session.

Each HLAPI/USS transaction request travels from a client application program on OS/390 UNIX System Services on MVS to Tivoli Information Management for z/OS on the same or a different host along the route shown in Figure 27 on page 294. This example illustrates HLAPI/USS installed on one MVS host and Tivoli Information Management for z/OS installed on a different host. The requester uses TCP/IP to establish a connection to the remote Tivoli Information Management for z/OS database.

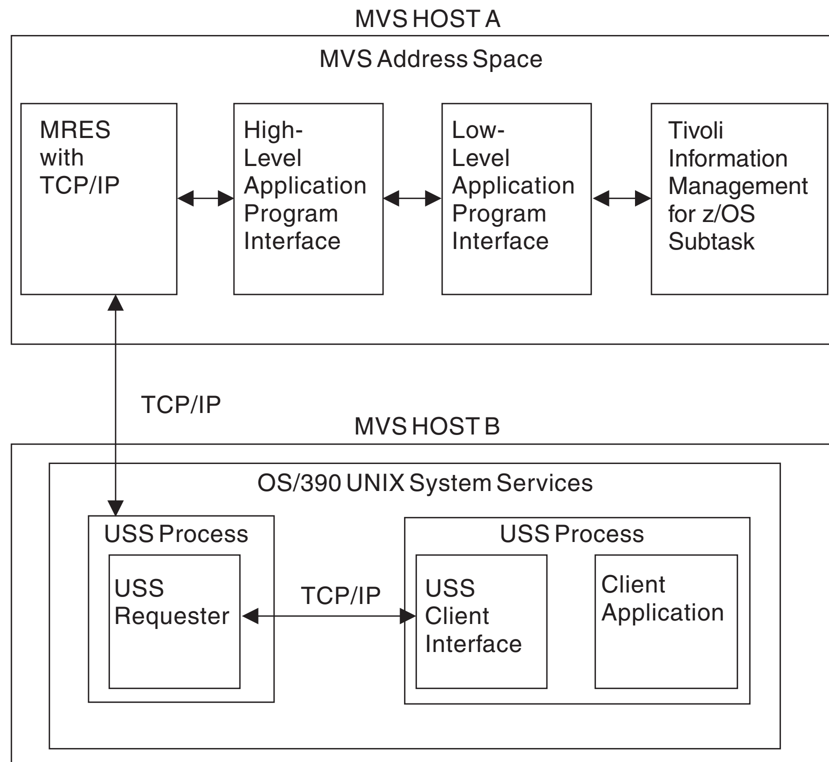


Figure 27. HLAPI/USS Transaction Flow

The transaction reply travels from Tivoli Information Management for z/OS to the client application along the same route in reverse.

The following steps describe the events that occur when a HLAPI/USS transaction is processed. For simplicity, assume that the transaction is neither the HL01 transaction nor the HL02 transaction of the transaction sequence.

On the USS host running the client application program

A client application program initiates a transaction by calling the HLAPI/USS function **IDBTransactionSubmit()**. The argument passed to this function is a HICA structure and its control PDBs and input PDBs. Together, these structures represent a ***transaction request***. The HLAPI/USS client interface frees any output, error, and message PDBs it allocated for the client application program during the previous transaction in the sequence. The client interface converts the data in the HICA and PDBs from data–structure format to byte–stream format, then transmits the data via TCP/IP sockets to the requester identified in the database profile specified in a control PDB.

On the USS host running the requester

The HLAPI/USS requester forwards the transaction request to the MRES with TCP/IP server over the communication link established during the HL01 transaction that initiated the transaction sequence. For more information about communication links between requesters and servers, see “Defining the Client Interface to Requester Communication Link” on page 300.

On the MVS host running the MRES with TCP/IP server and Tivoli Information Management for z/OS

When the transaction request arrives at the server, the server submits the request to the Tivoli Information Management for z/OS HLAPI. After the requested HLAPI transaction finishes, the server transmits the HICA, the output PDB, error PDB, message PDB, and the **PDBCODE** field of the input PDBs to the requester.

On the USS host running the requester

The requester forwards the transaction reply to the HLAPI/USS client interface that submitted the transaction.

On the USS host running the client application program

The HLAPI/USS client interface updates the HICA and PDBs of the client application program with data received from the transaction. The order of the PDBs in each chain is maintained from the server to the client interface. The transaction is complete, and the client application program returns from the call to **IDBTransactionSubmit()**.

HLAPI/USS Configuration Considerations

The following information is helpful in configuring HLAPI/USS.

A UNIX System Services client application program can manage multiple concurrent transaction sequences through HLAPI/USS. Each HICA is associated with a specific transaction sequence. Each transaction sequence is associated with a Tivoli Information Management for z/OS logical session on MVS. Therefore, there is a one-to-one correspondence between HICAs and Tivoli Information Management for z/OS logical sessions.

A client application program specifies the parameters listed below when it submits an HL01 transaction. Each of these parameters remains in effect for the duration of a transaction sequence. If conversation sharing (described in “IDBSHARECMS” on page 305) is enabled, the requester can assign multiple transaction sequences to the communication connection if each transaction sequence uses the same values for the following parameters:

- *Requester host* (**IDBREQUESTERHOST**) *and service name* (**IDBREQUESTERSERVICE**)

  The *requester host* and the *requester service name* are parameters in the database profile. Together, they identify the requester host and TCP/IP service name or alias, which the client interface uses to establish communication with the requester.

- MRES with TCP/IP *server host* (**IDBSERVERHOST**) and *service name* (**IDBSERVERSERVICE**).

  The **IDBSERVERHOST** and **IDBSERVERSERVICE** keywords are specified in the database profile. Together, they identify the MVS host and the MRES with TCP/IP service name that the requester uses to establish a TCP/IP conversation with the server on MVS.

- *User ID* for the MVS logon

  The MVS *user ID* is specified in the **SECURITY_ID** control PDB for the HL01 transaction.

- *User password* for the MVS logon

The MVS *password* is specified on the **PASSWORD** control PDB for the HL01 transaction.

The requester assigns a new transaction sequence to an existing conversation if the following conditions apply when the HL01 transaction is processed:

- Conversation sharing is enabled.
- The parameters of the conversation match the parameters specified by the client application program for the transaction sequence (as just identified).
- Fewer than 10 transaction sequences are assigned to the conversation.

A server can perform transactions against multiple Tivoli Information Management for z/OS databases within the same BLX-SP on the server's MVS host. However, a logical session can perform transactions against only one Tivoli Information Management for z/OS database. When a server receives a valid HL01 transaction request, the server establishes a Tivoli Information Management for z/OS logical session with the database specified in a control PDB. The association between the logical session and the database lasts until the transaction sequence ends.

Multiple transaction sequences can be routed through a single MRES with TCP/IP conversation and its associated Tivoli Information Management for z/OS logical session. Each session processes transactions serially on a first-in, first-out (FIFO) basis. A transaction sequence cannot have more than one transaction pending. All previously submitted transactions must be complete before a client application program can submit another transaction in the same transaction sequence.

Resources Needed for the Client Interface

When you write a client application program, consider that the HLAPI/USS client interface uses the following types of resources:

- Processes

  A client interface requires that the *process ID* and the effective *user ID* of a calling process remain constant from call to call for a transaction sequence. This allows HLAPI/USS to reaccess and release resources without compromising resources or data on the OS/390 UNIX System Services system.

- Sockets

  A client interface uses sockets from the **AF_INET** address family. The client interface uses, at most, two AF_INET sockets.

  The descriptor table of any process calling HLAPI/USS services must allow enough entries for the sockets and files used by the client interface.

- Memory

  The amount of memory that a client interface uses to process a transaction is difficult to predict. The type of transactions that the client application calls and the database contents determine the memory requirements of the client interface. The client application program is responsible for allocating and freeing memory used for the HICA, and the input and control PDBs. The client interface allocates and frees memory used for output, error, and message PDBs, and other HLAPI/USS structures. In general, search transactions are more memory-intensive than other transactions.

- Files

A client interface accesses the following OS/390 UNIX System Services system files during transaction processing:
- **/etc/services**
- **/etc/hosts**
- **/etc/utmpx**

A client interface also accesses the following HLAPI/USS files during transaction processing. The client interface does not delete these files except when replacing an old archived file with a new archived file:
- Database profile
- Active trace log file
- Archived trace log file
- Probe log file (**blmprobe.log**)

Additional files created during installation of a client interface are listed in "Components of HLAPI/USS" on page 352. Most of these files, with the exception of the files in the directory ***/examples**, are used during transaction processing (where * is the directory where the HLAPI/USS is installed).

The descriptor table of any process calling HLAPI/USS services must allow enough entries for the sockets and files used by the client interface.

Resources Needed for the Requester

The HLAPI/USS requester uses the following types of resources:

- Files
  - /etc/services
  - /etc/hosts
- Semaphores
  - The requester uses semaphores to serialize access to conversations. The number of semaphores the requester requires is equal to the maximum number of conversations it can create plus one. Therefore, your OS/390 UNIX System Services maximum semaphore limit must be greater than the maximum number of conversations that the requester can create. Refer to "IDBMAXCMS" on page 305 for information on how the requester determines the maximum number of conversations that it can create.

A requester also accesses a system profile, if one is specified, and the files listed for the requester in "Components of HLAPI/USS" on page 352.

Hardware and Software Requirements

The HLAPI/USS clients consist of two parts:
- Requester
- Client interface

Both the OS/390 UNIX System Services requester and client interface can be run on the same machine or on different machines. When you use SMP/E to install HLAPI/USS, both the requester and client interface are installed.

Hardware for HLAPI/USS

- TCP/IP connectivity is required between the MVS system running the requester and the MRES with TCP/IP servers.
- The MVS systems running the client interface and the requester.

Software for HLAPI/USS

- TCP/IP 3.2 (5655–HAL) with PTF UN98840, or equivalent
- OS/390 UNIX System Services must be configured to start up in full-function mode. Refer to *OS/390 OpenEdition Planning*.

32

Installing and Setting Up HLAPI/USS

Installing HLAPI/USS involves the following steps:

- Plan your HLAPI/USS configuration
- Install HLAPI/USS using SMP/E
- Configure HLAPI/USS and associated software

Planning a HLAPI/USS Configuration

Decide which OS/390 UNIX System Services host will run your HLAPI/USS client application programs or HLAPI/USS requesters. You need to install the HLAPI/USS feature using **SMP/E** on each of these OS/390 UNIX System Services hosts. Refer to the Program Directory for installation instructions.

Note: Upgrades or patches that can be downloaded from a Tivoli Web site may be available for HLAPI/USS. Visit the Tivoli Information Management for z/OS Web site for more information:

http://www.tivoli.com/infoman

Configuring HLAPI/USS and Associated Software

After you install HLAPI/USS on your OS/390 UNIX System Services hosts, you must configure the communication link between the requester host and an MRES with TCP/IP server. You must also enable communication between the host that will run your client application programs and the requester hosts. Then you must define database profiles to enable your client application programs to use HLAPI/USS services. You may also need to create system profiles for your requester hosts. The following sections tell you how to configure your communications software and update various files. "HLAPI/USS Profiles, Environment Variables, and Data Logging" on page 303 provides information about defining database and system profiles.

Configuring HLAPI/USS for TCP/IP

Update either the file /etc/services or else the data set hlq.ETC.SERVICES, whichever you use, on a requester host to associate a service name or alias with a TCP/IP port number of an MRES with TCP/IP server. You must specify a **service name** and **port number** of each server host the requester needs to be able to connect to. The port numbers must match those on the MVS host designated for the Tivoli Information Management for z/OS MRES with TCP/IP servers. The general format of an entry in /etc/services or hlq.ETC.SERVICES is:

```
<service>     <port>/tcp   <alias_list>       #<comment>
```

<service> The server service name

<port> The server port number

<alias_list> Alias definitions for the service

<comment> Comment text that describes the service.

For example, to associate the default server service name (**infoman**) with server port number **1451**, you must place the following line in the file **/etc/services** or the data set **hlq.ETC.SERVICES**, whichever you use, before you run the HLAPI/USS:

```
infoman    1451/tcp      #default MRES server
```

The default service name and port number 1451 are reserved for Tivoli Information Management for z/OS. You can use them to designate your server service. If a client does not specify a server service (*IDBSERVERSERVICE*) in the database profile specified on the HL01 transaction, *infoman* will be assumed. Therefore, be sure to include it in the **/etc/services** file or the data set **hlq.ETC.SERVICES**.

Be sure that your client application programs are aware of the service names that you define. If you access an MRES for TCP/IP that uses a port number other than the default (*infoman/1451*), you must specify the server service name in the **IDBSERVERSERVICE** keyword in your database profile.

If your MRES with TCP/IP server and requester run on the same MVS host, you may not use the same service name and port number to identify them. Each requester and each MRES with TCP/IP server on the same MRES host system must be identified with a unique service name and port number.

If your client applications use host names to identify the server hosts, the requester must be able to resolve the host names.

Defining the Client Interface to Requester Communication Link

To define the communication link between a requester and a client interface, you must update the **/etc/services** file or the data set **hlq.ETC.SERVICES**, whichever you use, on all hosts where HLAPI/USS is installed. Host names must also be resolvable.

Note: Service names, host names, and aliases are case-sensitive.

Updating /etc/services on a Requester Host

Update the file **/etc/services** or the data set **hlq.ETC.SERVICES**, whichever you use, on a requester host to associate the requester service name or alias with the TCP/IP port number of the requester. The general format of a requester entry in **/etc/services** or the data set **hlq.ETC.SERVICES** is:

```
<service>     <port>/tcp   <alias_list>       #<comment>
```

<service> The requester service name

<port> The requester port number

<alias_list> Alias definitions for the service

<comment> Comment text that describes the service.

For example, to associate the default requester service name (*infoman*) with requester port number 1451, you must place the following line in the file **/etc/services** or the data set **hlq.ETC.SERVICES** before you run the HLAPI/USS:

```
infoman    1451/tcp      #default HLAPI/USS requester
```

The default service name and default port number are reserved for Tivoli Information Management for z/OS. You can use a different combination of service name (or alias) and port number for a requester. For each combination, you must specify a corresponding line in the **/etc/services** file or the data set **hlq.ETC.SERVICES** to associate the port number with the service name or alias. Port numbers greater than 6000 are user-definable.

Only one service can use a given port number and a given service name or alias in an **/etc/services** file or the data set **hlq.ETC.SERVICES** on an OS/390 UNIX System Services host. However, you can use the same port number or the same service name (or alias) on different OS/390 UNIX System Services hosts.

If your MRES with TCP/IP server and requester run on the same MVS host, you may not use the same service name and port number to identify them. Each requester and each MRES with TCP/IP server on the same MVS host system must be identified with a unique service name and port number.

You can run multiple requesters on a single OS/390 UNIX System Services host. Use a different service name for each requester and associate the service name with a port number in the requester's **/etc/services** file or the data set. You also need to create a system profile for each requester and specify the appropriate service name in each profile.

Updating /etc/services on a Client Host

You must associate the client interface with each local or remote requester that the client interface communicates with. To do this, update the file **/etc/services** or the data set **hlq.ETC.SERVICES**, whichever you use, on the client interface host with the requester service names and their corresponding TCP/IP port numbers. If you use host names to identify your requester hosts, the host names must be resolvable.

The general format of a requester entry in **/etc/services** or the data set **hlq.ETC.SERVICES** is:

```
<service>     <port>/tcp   <alias_list>       #<comment>
```

<service> The requester service name

<port> The requester port number

<alias_list> Alias definitions for the service

<comment> Comment text describing the service.

For example, to associate the default requester service name (*infoman*) with requester port number 1451, put the following line in the **/etc/services** file or the data set **hlq.ETC.SERVICES** before you attempt to run the HLAPI/USS:

```
infoman    1451/tcp      #default HLAPI/USS Requester
```

The default service name and default port number are reserved for the exclusive use of Tivoli Information Management for z/OS. However, you may use a different combination of service name (or alias) and port number for a requester. For each combination you specify for your requesters, put a corresponding line in the client's **/etc/services** file or the data set

hlq.ETC.SERVICES to associate the port number with the service name or alias. Port numbers greater than 6000 are user-definable; they should not conflict with reserved ports.

In a **/etc/services** file or the data set **hlq.ETC.SERVICES** on an OS/390 UNIX System Services host, only one service can use a given port number and a given service name or alias. However, different **/etc/services** files or **hlq.ETC.SERVICES** data sets on different OS/390 UNIX System Services hosts may use the same port number or the same service name (or alias). This is because each OS/390 UNIX System Services host manages its own set of ports and service names.

33

HLAPI/USS Profiles, Environment Variables, and Data Logging

Certain aspects of HLAPI/USS can be customized to meet the requirements of your application. You do this by specifying profile keywords and values in two types of HLAPI/USS profiles. The profiles are OS/390 UNIX System Services text files; so you can use any text editor to create and update them.

The first type of profile is a *system profile*, which is associated with a requester. You can specify the name of a system profile as an optional argument of a command that starts a HLAPI/USS requester. The parameters in a system profile control aspects of HLAPI/USS operation. If no system profile is specified, the requester uses default values for all system profile parameters.

The second type of profile is a *database profile*, which is associated with a transaction sequence. The name of a database profile must be specified by a control PDB passed to HLAPI/USS as part of an HL01 transaction. The parameters in a database profile control aspects of HLAPI/USS operation.

Profile Syntax

Profile parameter entries are specified in the form

```
<keyword>=<data_value>
```

<keyword> Represents one of the keywords defined by HLAPI/USS.

= Is a literal character.

<data_value> Represents a data value to be associated with the keyword.

Whitespace characters (blanks or tabs) can precede or follow the value for *keyword* or *data value*. The *data value* includes all characters following the **=** to the end of the line. Each profile parameter must be specified entirely on a single line. For example,

```
IDBTIMEOUT = 60
```

Profile comments are specified in the form

```
REM <comment_text>
```

REM A literal keyword

<comment_text> Any sequence of characters up to the end of the line.

Whitespace characters (blanks or tabs) can precede the **REM** keyword; at least one whitespace character must immediately follow the **REM** keyword. Each profile comment must be specified entirely on a single line. For example,

```
REM    This is an example of a comment in a profile.
```

Each line of a profile must contain exactly one of the following:
- A profile parameter entry
- A profile comment
- Whitespace only

Keywords cannot be duplicated in profiles. If duplicate keywords are detected, processing stops and an error is returned to the client application program.

Profile keywords and data values are case-sensitive. Profile keywords must be entered with uppercase characters only. Profile data values must match their definitions in your **/etc/services** or other system configuration file.

When specifying a numeric value in a profile, use decimal digits to represent the value. Do not place any delimiter characters, such as commas or periods, among the digits of the value.

System Profile

You can specify the name of a single system profile as an optional parameter of a command that starts a HLAPI/USS requester. If a system profile is not specified when a requester is started or if some parameters are unspecified in the system profile, the requester uses the default values for the parameters. A requester reads a system profile only once when the requester is started. The developer of the client application program or OS/390 UNIX System Services system administrator or superuser creates system profiles. Multiple requesters can use the same system profile.

You can specify either the file name only or you can fully qualify the name with its path and drive. If you specify only a file name, the file is obtained from the current working directory. If the file is not found, the qualifying path name is obtained from the value of the OS/390 UNIX System Services environment variable **BLMSMPATH**. See “BLMSMPATH” on page 311 for additional information.

The valid keywords for system profiles follow.

IDBINBOUNDBUFSIZE

This value specifies the number of bytes to allocate for the communication buffers that the requester uses for receiving data from the server for transmittal to the client interface.

The buffer size can affect communication performance. For example, you might wish to specify a large buffer size, such as 28672, if the requester will be handling database search transactions that generate large amounts of reply data. For related information, see “IDBOUTBOUNDBUFSIZE” on page 305.

Valid values: Any integer from 1 to 32767, inclusive.

Values are rounded up to the nearest multiple of 4096. If the value after rounding is 32768, it is adjusted to 32767.

Default value:
4096

IDBMAXCMS

This value specifies the maximum number of child processes the requester can create to manage conversations between the requester and servers. The conversations are in a one-to-one correspondence with the requester processes that manage the conversations. The actual number of conversations might be further limited by constraints on OS/390 UNIX System Services system resources. One example is the OS/390 UNIX System Services system variable **_SC_CHILD_MAX**, which limits the number of processes for each user.

Valid values: Any integer from 1 to 65534, inclusive.

Default value:
65534

IDBOUTBOUNDBUFSIZE

This value specifies the number of bytes to allocate for the communication buffers that the requester uses to transmit transaction data received from the client interface to the server.

The buffer size can affect communication performance. For example, you might wish to specify a large buffer size, such as 8192, if the requester will be handling record creation transactions that involve large amounts of record data. For related information, see “IDBINBOUNDBUFSIZE” on page 304.

Valid values: Any integer from 1 to 32767, inclusive.

Values are rounded up to the nearest multiple of 4096. If the value after rounding is 32768, it is adjusted to 32767.

Default value:
4096

IDBSERVICENAME

This value specifies the service name for the requester that is listed in the OS/390 UNIX System Services file **/etc/services** or the **hlq.ETC.SERVICES** data set, whichever you use, on the requester host.

Valid values: Any valid service name or alias. Service names and aliases are case-sensitive.

Default value:
infoman

This is the default requester service name suggested during HLAPI/USS installation. If you specified a different service name in your **/etc/services** file or the **hlq.ETC.SERVICES** data set during installation in “Defining the Client Interface to Requester Communication Link” on page 300, you must specify that service name or its alias on this parameter in a system profile.

Note: If you run an MRES with TCP/IP on the same MVS host as your requester, you must ensure that the port number corresponding to the service name for the requester is different from the port number that your MRES with TCP/IP uses. Refer to “Configuring HLAPI/USS for TCP/IP” on page 299 and “Defining the Client Interface to Requester Communication Link” on page 300 for additional information.

IDBSHARECMS

This keyword determines whether the requester should enable or disable conversation sharing. When conversation sharing is enabled, the requester assigns new client applications

to an existing conversation if criteria such as same server and same security ID are met. When conversation sharing is disabled (the default), each client application is assigned its own dedicated conversation. A conversation is terminated when the last client assigned to it submits an HL02.

Note: If you choose to use conversation sharing, you must be aware that there is a potential for a delay because transactions are handled synchronously. Thus, if Client A and Client B share a conversation, and Client A submits a long search and Client B submits an update, Client B will wait for Client A's search to complete before its transaction can be processed.

Note: If you are using pre-started API sessions (described in "MRES with Pre-started API Sessions Considerations" on page 18), you must disable conversation sharing.

The number of conversations that the requester can start up is based on requester and operating system limitations. When conversation sharing is disabled, you may be more likely to reach these limits if your client applications hold on to conversations for an extended period. HLAPI/USS provides the ability to start multiple concurrent requesters as long as each requester is assigned a unique port number identified by the **IDBSERVICENAME** in the system profile. You may wish to route client applications that require dedicated conversation to requesters that you have started with conversation sharing disabled, and route your other client applications to other requesters with conversation sharing enabled.

Valid entries: **0** (conversation sharing disabled) or **1** (conversation sharing enabled).

Default value: **0** (conversation sharing disabled).

IDBTIMEOUT

This value specifies the maximum number of minutes that can elapse between transactions in a single transaction sequence. This is called *idle time*, which is defined as the time interval between the completion of one transaction and the start of the next transaction in the same transaction sequence. The start time is determined by when the transaction becomes the active transaction in the requester-server conversation. The end time is determined by when the transaction stops being the active transaction in the conversation.

If the idle time for a transaction sequence exceeds the smaller of this value or the value of **IDBIDLECLIENTTIMEOUT** in the database profile for the transaction sequence, the requester refuses to process additional transactions in the sequence (including any transaction pending on the conversation queue). This allows the requester to end logical sessions and conversations if a client application program does not submit an HL02 transaction for a transaction sequence.

Valid values: Any integer from 0 to 35791394, inclusive.

The value is specified in minutes, except that a value of 0 corresponds to an infinite interval.

Default value:
35791394

System Profile Example

```
REM*****************************************************************
REM
REM          SAMPLE HLAPI/USS System Profile
REM
REM*****************************************************************

IDBINBOUNDBUFSIZE       =     28672

IDBMAXCMS               =        25

IDBOUTBOUNDBUFSIZE      =      8192

IDBSERVICENAME          = hlapiuss1

IDBSHARECMS             =         0

IDBTIMEOUT              =        60
```

Database Profile

Your client application must specify the name of a single database profile in a **DATABASE_PROFILE** control PDB on the HL01 transaction of each transaction sequence. If a **DATABASE_PROFILE PDB** is not specified or if multiple **DATABASE_PROFILE PDB**s are specified for an HL01 transaction, an error is returned to the client application program. If a **DATABASE_PROFILE PDB** is specified on a transaction other than HL01, the PDB is ignored. A client interface reads the database profile only once per transaction sequence, and that is during the HL01 transaction. The OS/390 UNIX System Services system administrator or superuser, client application developer, or client application user creates the database profile for a transaction sequence. Multiple client application programs can use the same database profile.

The specified database profile must contain at least an entry for the **IDBSERVERHOST** keyword. This information identifies to which MRES with TCP/IP server the requester establishes a communication link for the client application. See “IDBSERVERHOST” on page 309 for information on this keyword.

You can specify either the database profile file name only or you can fully qualify the name with its path and drive. If you specify only a file name, the file is obtained from the current working directory. If the file is not found, the qualifying path name is obtained from the value of the OS/390 UNIX System Services environment variable **BLMDBPATH**. See “BLMDBPATH” on page 311 for more information.

The valid keywords for database profiles follow.

IDBDATALOGLEVEL

This keyword determines whether the client interface logs transaction data. The value specified with this keyword can be overridden by the OS/390 UNIX System Services environment variable **IDBDATALOGLEVEL**.

Valid values: 0 (logging disabled) 1 (logging enabled)

Default value:
0 (logging disabled).

The default value is applied only if neither the OS/390 UNIX System Services environment variable **IDBDATALOGLEVEL** nor the database profile keyword **IDBDATALOGLEVEL** specifies a value.

IDBIDLECLIENTTIMEOUT

This keyword specifies the maximum number of minutes that can elapse between transactions in a single transaction sequence. This is called ***idle time***, which is defined as the time interval between the completion of one transaction and the start of the next transaction in the same transaction sequence. The *start time* is determined by when the transaction becomes the active transaction in the requester-server conversation. The *end time* is determined by when the transaction stops being the active transaction in the conversation.

If the idle time for a transaction sequence exceeds the smaller of this value or the value of **IDBTIMEOUT** in the system profile associated with the requester, the requester refuses to process additional transactions in the sequence (including any transaction pending on the conversation queue). This allows the requester to end logical sessions and conversations if a client application program does not submit an HL02 transaction for a transaction sequence.

Valid values: Any integer from 0 to 35 791 394, inclusive. The value is specified in minutes, except that a value of 0 corresponds to an infinite interval.

Default value:
60

IDBLOGFILENAMEACTIVE

This keyword specifies the name of the active log file for the transaction sequence. Multiple transaction sequences can use the same log file. However, it is recommended that you limit the number of simultaneous transaction sequences per log file to reduce contention for write access to the file and to prevent the file from being archived frequently.

Valid values: Any valid file name on your OS/390 UNIX System Services system. File names and path names are case-sensitive.

Default value:
./IDB_LOG.ACT

IDBLOGFILENAMEOLD

This keyword specifies the name to give an active log file when it is archived. Active log files are archived when they reach their maximum size as specified on the keyword **IDBLOGFILESIZE**. See “IDBLOGFILESIZE”.

Valid values: Any valid file name on your OS/390 UNIX System Services system. File names and path names are case-sensitive.

Default value:
./IDB_LOG.OLD

IDBLOGFILESIZE

This keyword specifies the approximate maximum size, in bytes, of the active log file. If logging a transaction causes the active log file to exceed this size, the active log file is archived. Archiving involves two steps:
1. Closing and renaming the active log file
2. Opening a new active log file

Valid values: Any integer from 0 to 10 485 760, inclusive. (The upper limit equals 10 megabytes.)

If a value between 1 and 4 095 is specified, then 4 096 is substituted. If the value 0 is specified, HLAPI/USS does not restrict the size of the log file.

Default value:
262 144 (Equal to 256 kilobytes.)

IDBREQUESTERHOST

This keyword identifies the OS/390 UNIX System Services host that is running the requester so that communication can be established between the client interface and the requester.

Valid values: Any valid IP address in dotted-decimal format, or any valid host name, such as **MVSHOSTX**. If you specify a host name, the host name must be resolvable on the client host.

Default value:
The OS/390 UNIX System Services host running the client interface that is handling the transaction sequence.

IDBREQUESTERSERVICE

This keyword specifies the service name of the requester you want to establish communication with. The service name must be listed in the **/etc/services** file or the **hlq.ETC.SERVICES** data set, whichever you use, on the client host.

Valid values: Any valid service name or alias. Service names and aliases are case-sensitive.

Default value:
infoman

This is the default requester service name suggested during installation. If you specified a different service name in your **/etc/services** file or the **hlq.ETC.SERVICES** data set during installation in "Defining the Client Interface to Requester Communication Link" on page 300, you must specify that service name or its alias on this parameter in a database profile.

Note: If you run an MRES with TCP/IP on the same MVS host as your requester, you must ensure that the port number corresponding to the service name for your requester is different from the port number that your MRES with TCP/IP uses. Refer to "Configuring HLAPI/USS and Associated Software" on page 299 and "Defining the Client Interface to Requester Communication Link" on page 300 for additional information.

IDBSERVERHOST

This keyword identifies the MVS host that is running the MRES with TCP/IP server you want the requester to establish communication with for your client application.

Valid values: Any valid IP address in dotted-decimal format, or any valid host name, such as **mvshost**. If you specify a host name, the host name must be resolvable on the requester host.

Default value:
None.

This keyword is required.

IDBSERVERSERVICE

This keyword specifies the service name of the MRES with TCP/IP you want to establish communication with. The service name must be listed in the OS/390 UNIX System Services **/etc/services** file or the **hlq.ETC.SERVICES** data set, whichever you use, on the requester host.

Valid values: Any valid service name or alias. Service names and aliases are case-sensitive.

Default value:
infoman

IDBSERVERSERVICE is an optional keyword. If you do not specify it when you specify **IDBSERVERHOST**, the default is assumed.

Note: If your MRES with TCP/IP and requester run on the same MVS host, you must ensure that the port number that your MRES with TCP/IP use is different from the port number corresponding to the service name for your requester. Refer to “Configuring HLAPI/USS for TCP/IP” on page 299 or “Defining the Client Interface to Requester Communication Link” on page 300 for additional information.

Database Profile Example

```
REM**********************************************************************
REM
REM                  SAMPLE HLAPI/USS Database Profile
REM
REM
REM**********************************************************************

IDBREQUESTERHOST     =          zeus
IDBREQUESTERSERVICE  =        hlapiuss9
IDBIDLECLIENTTIMEOUT =            30

IDBSERVERHOST        =       hera
IDBSERVERSERVICE     =       infosrv2

IDBDATALOGLEVEL      =             1
IDBLOGFILENAMEACTIVE =  ./helpdesk5.log
IDBLOGFILENAMEOLD    =  ./helpdesk5.old
IDBLOGFILESIZE       =        262144
```

Environment Variables

HLAPI/USS recognizes some OS/390 UNIX System Services environment variables for which you can define values. If you are using a shell with case-sensitive environment variables, you must enter the names of HLAPI/USS environment variables using only uppercase characters.

You can use the HLAPI/USS environment variables described in the following sections to override the corresponding database profile parameters and to qualify the path names for the profiles.

IDBDATALOGLEVEL

If this environment variable is set to a valid value when a client interface reads the database profile, the value in the environment variable overrides any value given for the database

profile keyword **IDBDATALOGLEVEL**. Valid values are 0 (transaction data logging disabled) and 1 (transaction data logging enabled). See "IDBDATALOGLEVEL" on page 307 for additional information.

IDBREQUESTERHOST

If this environment variable is set to a valid value when a client interface reads the database profile, the value in the environment variable overrides any value set for the database profile keyword **IDBREQUESTERHOST**. You can specify any valid IP address in dotted-decimal format, or any resolvable host name. See "IDBREQUESTERHOST" on page 309 for additional information.

IDBREQUESTERSERVICE

If this environment variable is set to a valid value when a client interface reads a database profile, the value in the environment variable overrides any value set for the database profile keyword **IDBREQUESTERSERVICE**. You can specify any valid service name or alias. See "IDBREQUESTERSERVICE" on page 309 for additional information.

BLMDBPATH

This environment variable specifies a search path for locating a database profile when both of the following conditions are true:

1. The database profile file name does not explicitly specify a complete path from the root directory to the database profile file. This is called a ***relative path***.
2. The database profile is not in the current working directory of the calling process.

You can specify multiple paths on **BLMDBPATH**. For example, if the file name of a database profile is specified using a relative path, then the following assignment:

```
BLMDBPATH=/:/usr/profiles/:
```

causes a client interface to seek the database profile first in the current working directory, then in the root directory, then in the directory **/usr/profiles**.

The colon **:** following the final path specification is optional. The trailing slash **/** of the second path specification is also optional. For example, the following two assignments are equivalent:

```
BLMDBPATH=/:/usr/profiles/:
```

```
BLMDBPATH=/:/usr/profiles
```

BLMSMPATH

This environment variable specifies a search path for locating a system profile when both of the following conditions are true:

1. The system profile file name does not explicitly specify a complete path from the root directory to the system profile file. This is called a ***relative path***.
2. The system profile is not in the initial working directory of the requester process. The initial working directory of the requester process is the current working directory at the time the requester is started.

You can specify multiple paths on **BLMSMPATH**. For example, if the file name of a system profile is specified using a relative path, the following assignment:

```
BLMSMPATH=/:/usr/profiles/:
```

causes a requester to seek the system profile first in the initial working directory of the requester process, then in the root directory, then in the directory **/usr/profiles**.

The colon **:** following the final path specification is optional. The trailing slash **/** of the second path specification is also optional. For example, the following two assignments are equivalent:

```
BLMSMPATH=/:/usr/profiles/:

BLMSMPATH=/:/usr/profiles
```

Transaction Logging

HLAPI/USS transactions can be logged by the server as well as by the client interface. When both server logging and client interface logging are active, you might notice differences between entries in the server log and the corresponding entries in the client interface log. For example, the client interface logs PDBs with a data length of zero but does not send those PDBs to the server. Therefore, the server log records no zero-length PDBs.

Transaction Logging by a Client Interface

Every HLAPI/USS transaction sequence has an associated log file in which the client interface can record transactions. The value specified for the HLAPI/USS environment variable **IDBDATALOGLEVEL** or the HLAPI/USS database profile keyword **IDBDATALOGLEVEL** determines whether transactions in a transaction sequence are logged by the client interface.

The client interface creates a log file automatically if it does not already exist. Log entries are appended to the end of a log file. In order to prevent the log file from growing indefinitely, HLAPI/USS provides an archiving mechanism. The client interface records individual transactions in the log file specified by the database profile keyword **IDBLOGFILENAMEACTIVE** until the file reaches or exceeds the maximum file size specified by the database profile keyword **IDBLOGFILESIZE**. The active log file is then renamed to the file name specified by the database profile keyword **IDBLOGFILENAMEOLD**. If a previously archived log file of the same name exists, it is deleted before the active log file is archived. Finally, a new active log file is created with the name specified by the database profile keyword **IDBLOGFILENAMEACTIVE**.

Multiple transaction sequences can use the same log file. However, if multiple transaction sequences attempt to record transactions in the same file, there will probably be contention for access to the file. To log a transaction, a client interface opens a log file, records the transaction, and closes the file. While the file is open, the process has exclusive write access to the file. If a process attempts to open the log file while another process has exclusive write access to it, the attempt to open the file fails. When this happens, the process that failed to open the file repeatedly attempts to open the file until the process opens the file successfully or reaches an internal HLAPI/USS retry limit. If the client interface fails to open the file because it reaches the retry limit, a return code and reason code are returned in the HICA to reflect the logging failure.

For more information about transaction logging by the client interface, see “Database Profile” on page 307 and “Environment Variables” on page 310.

Error Probe Logging by a Requester or Client Interface

A HLAPI/USS requester or client interface might encounter an error condition that cannot be explained with available information. When this happens, the requester or client interface records an entry in the HLAPI/USS probe log file **/tmp/blmprobe.log** on the local OS/390 UNIX System Services host. If a probe log file does not exist, one is created.

HLAPI/USS imposes no limit on the maximum size of a probe log file. You can delete or rename a probe log file at any time. However, avoid using a probe log file in any way that would prevent HLAPI/USS from opening the file for exclusive write access.

When diagnosing a problem associated with a HLAPI/USS client application program, you might find entries in HLAPI/USS probe log files to be useful supplements to any transaction reply data that HLAPI/USS returns to the program.

34

The HLAPI/USS Requester

The HLAPI/USS requester is a daemon task. You can start it manually or start it automatically from the shell or by JCL. Messages from the requester are written in the file **blmprobe.log** on the requester host. Two methods that you can use to start the requester are illustrated in this chapter. You should also consult *OS/390 OpenEdition Planning* for additional information on how to start daemons. The HLAPI/USS requester should be started by a superuser, but it does not require daemon authority.

Regardless of how you start a requester, the OS/390 UNIX System Services file **/etc/services** or **hlq.ETC.SERVICES** data set, whichever you use, on both the requester host and the client interface host must contain an entry associating the service name of the requester with the TCP/IP port for the requester. See “Configuring HLAPI/USS and Associated Software” on page 299 for information on updating the **/etc/services** file and **hlq.ETC.SERVICES** data set.

Starting the Requester from the Shell

The syntax of the command to start a HLAPI/USS requester is:

```
<Requester_executable> [ -p|-P <System_Profile> ]
```

<Requester_executable> Specifies the name of the requester executable file. The requester executable program name is **blmreq**. It is located in **/usr/lpp/InfoMan/bin**.

-p <System_Profile> Specifies a system profile to use when starting the requester. The value for **System_Profile** must be preceded by either *-p* or *-P*. At least one blank must separate the *-p* from the value for **System_Profile**. This parameter and value are optional. If they are not specified, defaults for the system profile keywords are used.

For example, if you start a requester with the following command:

```
/usr/lpp/InfoMan/bin/blmreq -p blmsys.pro
```

The requester will search for the system profile **blmsys.pro** in the current working directory first. If the profile is not there, it will search in any directories in the search path specified by the environment variable **BLMSMPATH**.

You can place an entry in the **/etc/rc** file to direct OS/390 UNIX System Services to automatically start a HLAPI/USS requester during OS/390 UNIX System Services initialization. Refer to *OS/390 OpenEdition Planning* for additional information.

Stopping a Requester from the Shell

Once started, a requester runs indefinitely even if all of its conversations have ended. You can use the shell **kill** command to stop a requester, but first you must determine the **process ID (pid)** of the requester's highest-level process. Use the following command to do that:

```
ps -ef | grep blmreq
```

Once you know the **process ID**, issue the following command to stop the requester:

```
kill <pid>
```

If the requester does not stop, you can use the **kill** command with the *–9* parameter, but you should consult the *OS/390 OpenEdition Command Reference* for possible consequences of using this command:

```
kill -9 <pid>
```

Starting a Requester by JCL

Another method of starting the HLAPI/USS requester is with a catalogued procedure that uses **BPXBATCH** to invoke the requester located in the HFS:

```
//BLMREQ PROC
//BLMREQ EXEC PGM=BPXBATCH,REGION=30M,TIME=NOLIMIT,
//     PARM='PGM  /usr/lpp/InfoMan/bin/blmreq
//            -p /usr/lpp/InfoMan/examples/blmsys.pro'
```

An operator can use the **start** command to start the catalogued procedure for the requester and can use the **cancel** command to stop the requester.

Diagnosing Some Common Problems

Symptom: Changing the setting of a parameter in a database profile or a system profile has no impact on HLAPI/USS operation.

Possible Causes:

1. HLAPI/USS failed to locate your profile because it is not in the profile search path.

   Action: Specify the complete path for the profile or define the profile search path by setting the value of the appropriate HLAPI/USS environment variable. Make sure the value of the environment variable applies to the calling process. For information about environment variables, see "Environment Variables" on page 310.

2. HLAPI/USS used another profile that was ahead of your profile in the search path.

   Action: Delete one of the profiles, change the search path by changing the value of the appropriate HLAPI/USS environment variable, or specify the complete path

name for the profile in the **DATABASE_PROFILE** control PDB.

3. A HLAPI/USS environment variable is overriding the value for the parameter.

   Action: Redefine the HLAPI/USS environment variable.

Symptom: Your client application process takes much longer than usual to run or it is rejected by the requester.

Possible causes:

1. If the conversation limit on the host is set too low, applications trying to start will have to wait until an HL01 is honored. This wait time may be significant, and nothing is returned to the waiting application indicating that the conversation limit was reached. When one of the applications already running finishes, the waiting application begins processing.

   Action: Ensure that the conversation limit is set high enough to run your applications.

2. The time between the client application's last transaction and the current transaction may have exceeded the allowable wait time specified by the database or the system profile. The client application is forced off the requester when the wait time is exceeded.

   Action: Ensure that the wait time set in your system profile is set high enough to run your applications.

35

HLAPI/USS Transactions

The work done by HLAPI/USS takes place through the use of HLAPI transaction sequences. Each transaction sequence begins with an HL01 transaction, optionally followed by any of the transactions listed in Table 1 on page 3, and ended by an HL02 transaction. Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* for information on the HLAPI transactions. This chapter explains aspects of transaction processing that are specific to the HLAPI/USS.

Validation of the Calling Process

When a client application program submits an HL01 transaction to begin a transaction sequence, HLAPI/USS records the process ID and the effective user ID of the process submitting the transaction. On all subsequent HLAPI/USS function calls of the transaction sequence, the process ID and effective user ID of the calling process must match those recorded for the transaction sequence. If they do not match, the HLAPI/USS returns an error to the calling process. This ensures that HLAPI/USS has the permissions necessary to access and free any HLAPI/USS resources that persist between calls to HLAPI/USS services.

CAUTION:

Do not modify a HICA or PDB associated with a transaction submitted to the HLAPI/USS until the transaction ends successfully or unsuccessfully. Any changes to a HICA or its associated PDBs during transaction processing causes unpredictable results.

Transaction Processing Modes

HLAPI/USS supports synchronous transaction processing only. Synchronous processing forces the current process in the application program to wait for a Tivoli Information Management for z/OS transaction to end before the process returns from a HLAPI/USS function call. The process cannot do any other work until the transaction is complete. Transaction completion includes both successful and unsuccessful outcomes.

Transaction Concurrency Limitations

A client application program can use multiple HICA structures, each corresponding to a different transaction sequence. At any time, any of the transaction sequences can have a transaction in process.

For any single requester-server conversation, transactions are processed serially on a first-come, first-served basis.

Developing HLAPI/USS Client Applications

To use HLAPI/USS data structures and function calls, your application source files must include the C language-based header file **blmh.h**. You can optionally include **blmech.h** to declare named constants for HLAPI/USS return and reason codes. When you compile and link your application, you must specify that you are using a dll and you must specify the definition side-deck *blmhlapi.x*. The runtime library is named **blmhlapi**, which provides the client interface service **IDBTransactionSubmit()**. The path and file name for the runtime library is

```
/usr/lpp/InfoMan/lib/blmhlapi
```

and the path and file name for the definition side-deck is

```
/usr/lpp/InfoMan/lib/blmhlapi.x
```

To use this function, your application program must do the following:

1. Allocate memory for HICA and PDB structures using the data types declared in the header file **blmh.h**.
2. Assign valid values to the fields of the structures.
3. Pass the structures and other arguments on calls to the HLAPI/USS functions.

Note: HLAPI/USS also provides a REXX HLAPI/USS interface which allows you to access HLAPI/USS functions from REXX programs in the same manner as HLAPI/USS on MVS allows you to access the HLAPI from MVS REXX programs. See “Using the REXX HLAPI/USS Interface” on page 329 for more information about REXX HLAPI/USS.

There are differences between the HICA and PDB structure definitions for the HLAPI and the corresponding structure definitions for HLAPI/USS. Corresponding field names are generally similar. Some field names have been modified for HLAPI/USS to conform to standard C-language naming conventions.

Including the HLAPI/USS Header File blmh.h

You must include the HLAPI/USS header file **blmh.h** in application source code that references HLAPI/USS function and data types. This header file declares the HLAPI/USS data types and function prototype your program needs to communicate with HLAPI/USS runtime services. Ensure that each source file’s references to HLAPI/USS function and data types fall within the scope of the include file’s declarations. To include the header file **blmh.h** in a source file, put the following line into the source file:

```
#include <blmh.h>    /* HLAPI/USS header file */
```

The default path and file names for these include files are:

```
/usr/lpp/InfoMan/include/blmh.h
```

Installation of HLAPI/USS creates the symbolic link

```
/usr/include/blmh.h
```

to reference the include file.

The compiler’s default search path for header files is generally sufficient to access the HLAPI/USS include file. If the header file resides in a location other than the default or if the specified symbolic link is absent, then you may need to explicitly specify a search path to enable the compiler to locate the header file.

Refer to the file **blmappl.c** in the HLAPI/USS examples subdirectory (listed below) for an example of how to include the HLAPI/USS header files:

```
/usr/lpp/InfoMan/examples
```

Including the HLAPI/USS Header File blmech.h

You can include the HLAPI/USS header file **blmech.h** in application source files to declare named constants defined for HLAPI/USS return and reason codes. This header file declares the named constants for most LLAPI, HLAPI, and HLAPI/USS return and reason codes used by HLAPI/USS. Ensure that each source file's references to the named constants fall within the scope of the include file's declarations. To include the header file **blmech.h** in a source file, put the following line into the source file:

```
#include  <blmech.h>    /*  HLAPI/USS header file  */
```

The default path and file names for these include files are:

```
/usr/lpp/InfoMan/include/blmech.h
```

Installation of HLAPI/USS creates the symbolic link

```
/usr/include/blmech.h
```

to reference the include file.

The compiler's default search path for header files is generally sufficient to access the HLAPI/USS include file. If the header file resides in a location other than the default or if the specified symbolic link is absent, then you may need to explicitly specify a search path to enable the compiler to locate the header file.

Refer to the file **blmappl.c** in the HLAPI/USS examples subdirectory (listed below) for an example of how to include the HLAPI/USS header files.

```
/usr/lpp/InfoMan/examples
```

Overview of HICA and PDB Data Structures

The primary data structures that your application uses to communicate with HLAPI/USS are HICA structures and PDB structures. The argument of the HLAPI/USS function **IDBTransactionSubmit()** is the address of a HICA that represents a transaction sequence. The fields of the HICA include pointers to PDB structures. Each non-null PDB pointer of the HICA is a pointer to the first element of a linked list of PDBs of a particular type. There are five types of PDBs, hence five PDB pointers in a HICA:

- Control PDB
- Input PDB
- Output PDB
- Error PDB
- Message PDB

Each type of PDB can have subtypes. For example, some of the subtypes of control PDBs are **SPOOL_INTERVAL**, **HLIMSG_OPTION**, **HLAPILOG_ID**, **DATABASE_PROFILE**, **SECURITY_ID**, and **PASSWORD**.

To submit a transaction request to HLAPI/USS, your application program calls **IDBTransactionSubmit()** and passes the address of a HICA with associated control PDB and input PDB structures. The HL01 transaction must include the three required control PDBs, **SECURITY_ID**, **PASSWORD**, and **DATABASE_PROFILE**, which are described in "Initialize Tivoli Information Management for z/OS (HL01)" on page 325. The HLAPI/USS communicates the transaction reply to your application program by associating linked lists of

output PDBs, error PDBs, and message PDBs with the HICA. In addition, HLAPI/USS updates the list of input PDBs for some transaction replies.

The values you store in the control PDBs and input PDBs depend on the specific HLAPI/USS transaction you want to use. For example, if a control PDB specifies that the transaction type is for record creation, then input PDBs may specify the data values for individual fields of the record to be created. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information about HICAs and PDBs.

Allocating and Initializing a HICA structure

Your application program must allocate and initialize one HICA structure for each HLAPI/USS transaction sequence. The HICA is used throughout the transaction sequence. Therefore, your application program must ensure that the storage allocated for the HICA persists for the duration of the transaction sequence. Your application program can allocate any class of storage for a HICA, such as shared memory, automatic storage, and static storage. This example shows an application fragment that illustrates the allocation and initialization of a HICA. Note that the **ENVP** field of a HICA must be set to null before each HL01 transaction and must not be changed by the application program during the remainder of the transaction sequence.

```
/****************************************************************/
/* allocate a HICA for a transaction sequence                  */
/****************************************************************/
   static IDB_HICA    MyHICA;     /* allocate HICA              */
/****************************************************************/
/* initialize a HICA before an HL01 transaction                */
/****************************************************************/

   memset( &MyHICA,               /* fill HICA with nulls       */
           '\0',
           sizeof(MyHICA)       );

   memcpy( MyHICA.ACRO,           /* initialize HICA eyecatcher */
           HICAACRO_TEXT,
           sizeof(MyHICA.ACRO)  );

   MyHICA.LENG = sizeof(MyHICA);  /* set HICA length            */

   MyHICA.ENVP = (void *)0;       /* initialize ENVP field --   */
                                  /* ONLY FOR HL01 TRANSACTION  */

⋮
                       /* associate Control PDBs     */

⋮
                       /* and Input PDBs with HICA   */
```

Allocating and Initializing a PDB Structure

Your application program is responsible for allocating, initializing, and freeing control PDBs and input PDBs for each transaction. HLAPI/USS is responsible for allocating, initializing, and freeing output PDBs, error PDBs, and message PDBs for each transaction.

Your application must allocate and initialize any required control PDBs and input PDBs before it submits a transaction request. Your application program should examine any output PDBs, error PDBs, and message PDBs that are returned with a transaction reply. In addition, your application program should examine the input PDBs for updates made by HLAPI/USS

as part of certain transaction replies. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information.

The last field of a PDB structure is the data field. This field is declared to be a one-element array of unsigned characters, but it is actually a variable-length field. The actual length of the field is determined by the number of extra bytes allocated for the PDB structure. Your application program should use the data field to record the actual number of bytes of data in the data field. This arrangement allows your application program to conserve memory by allocating each PDB with only as much storage as it needs. The total size of a PDB is the sum of **PDBFIX_SIZE** (a constant defined in **blmh.h**) and the number of bytes for the data field.

This example shows an application fragment that illustrates allocation and partial initialization of an input PDB. Note the calculation of PDB size.

```
int      PDBsize;
PDB     *pPDB;                /* pointer to PDB           */

PDBsize = PDBFIX_SIZE + strlen("DOE JOHN");
pPDB = malloc( PDBsize );     /* allocate memory for PDB   */

if (pPDB) {

    memset( pPDB,             /* fill PDB with nulls      */
            '\0',
            PDBsize         );
    memcpy( pPDB->Acro,       /* initialize PDB eyecatcher  */
            PDBACRO_TEXT,
            sizeof(pPDB->Acro));

    memset( pPDB->Name,       /* initialize with blanks     */
            ' ',
            sizeof(pPDB->Name));
    memcpy( pPDB->Name,       /* record field value          */
            "REPORTER_NAME",
            strlen("REPORTER_NAME"));
    pPDB->Datl = strlen("DOE JOHN");  /* initialize length     */
    memcpy( pPDB->Data,             /* initialize value      */
            "DOE JOHN",
            pPDB->DATL            );


    pPDB->Proc = 'V';         /* request validation          */
    pPDB->Code = ' ';         /* initialize data error code  */

⋮
                  /* initialize other fields */

    pPDB->Next = MyHICA.INPP;   /* insert PDB into linked list  */
    pPDB->Prev = NULL;          /*   of Input PDBs              */
    MyHICA.INPP->Prev = pPDB;
    MyHICA.INPP = pPDB;

    }
⋮
```

35. HLAPI/USS Transactions

HLAPI/USS Function

One C-language HLAPI/USS function is available to client application programs:

- **IDBTransactionSubmit()** for submitting a transaction request

The HLAPI/USS header file **blmh.h** contains the prototype for the function and declarations of the data types and values associated with the function.

IDBTransactionSubmit()

A client application program calls the function **IDBTransactionSubmit()** to submit a transaction request to HLAPI/USS. In a C-language program, the call to **IDBTransactionSubmit()** looks like

```
rc = IDBTransactionSubmit(pHICA);
```

Your application must provide the variable *pHICA* as a pointer to a structure of the type HICA that contains the HICA that you want to submit to HLAPI/USS.

HLAPI/USS returns a value from this function call that you should examine before looking at the HICA return and reason codes. This return code (rc) is a variable of type **IDBRC_TYPE**. The values that can be returned for it are listed in “HLAPI Service Call Return Codes” on page 367.

```
#include <blmh.h>
IDBRC_TYPE     rc;
HICA           MyHICA;
⋮
rc = IDBTransactionSubmit( &MyHICA );
⋮
```

Usage Notes

After calling **IDBTransactionSubmit()** to submit a transaction, a client application program should not alter the HICA, PDBs, or other associated structures until the transaction is complete.

After submitting the HL01 transaction to initiate a transaction sequence, a client application need only change the control PDB and input PDB chains to prepare the HICA for a subsequent transaction. The HICA pointers to the output PDB, message PDB, and error PDB chains may be set optionally to NULL to prepare for a subsequent transaction.

Additional information about error conditions for a particular transaction may be obtained from the following:

- The values of the HICA fields **RETC** and **REAS**, if **IDBTransactionSubmit()** returned the value **IDBRC_XERR**
- Any error PDBs and message PDBs chained to the HICA
- Any input PDBs updated by HLAPI/USS
- The HLAPI/USS file blmprobe.log
- The client interface log files, if transaction logging is enabled for the transaction sequence.

- "Transaction Logging by a RES and by an MRES Without Pre-started API Sessions" on page 20 .

Using the HLAPI/USS Function in a Transaction Sequence

All HLAPI/USS function calls for a given transaction sequence must be made by the same process. In addition, the effective user ID of the process must not vary from one HLAPI/USS function call to the next. HLAPI/USS returns an error to the calling process if the process ID or effective user ID does not match the value recorded during the call to **IDBTransactionSubmit()** for the HL01 transaction of the transaction sequence.

The HLAPI transactions supported by HLAPI/USS are listed in Table 1 on page 3. For a description of the transactions supported by HLAPI/USS, refer to the *Tivoli Information Management for z/OS Application Program Interface Guide*. The remainder of this section discusses aspects of each transaction that are specific to HLAPI/USS.

Initialize Tivoli Information Management for z/OS (HL01)

The HLAPI/USS transaction HL01 requests a connection to a Tivoli Information Management for z/OS database on a specific Tivoli Information Management for z/OS server. The HL01 transaction initiates a HLAPI/USS transaction sequence. This section describes the steps performed for an HL01 transaction.

1. The user creates a HLAPI/USS database profile for the database connection. A text editor can be used to create or update a database profile. Different database connections can share the same database profile, but it is generally advisable to create a different database profile for each database connection. The database profile specifies the server (host and service names for a server that supports TCP/IP), the requester host and service names, and other parameters governing the transaction sequence for the database connection. See "HLAPI/USS Profiles, Environment Variables, and Data Logging" on page 303 for a description of database profile contents.
2. The client application program allocates and initializes a HICA structure for the transaction sequence. The client application program allocates and initializes three required control PDBs:
   - The **DATABASE_PROFILE** PDB, specifying the name of the database profile for the transaction sequence
   - The **SECURITY_ID** PDB, specifying the MVS user ID
   - The **PASSWORD** PDB, specifying the MVS user password

   The client application places these PDBs on the HICA's chain of control PDBs. The **DATABASE_PROFILE** PDB can go anywhere on the chain of control PDBs. If the **DATABASE_PROFILE** PDB uses a relative path for the database profile, the value of the HLAPI/USS environment variable **BLMDBPATH** is used to qualify the path name and locate the profile. A relative path does not explicitly specify a complete path from the root directory to the file.
3. The client application program calls **IDBTransactionSubmit()**, passing the address of the HICA as a parameter. During the call to **IDBTransactionSubmit()**, HLAPI/USS reads the database profile and records its parameters for use during the entire transaction sequence. The client interface then uses the requester host and service names to establish communication with the requester.
4. At this time, the specified requester must be available to the client interface.

5. The client interface transmits the MVS user ID, MVS password, and server host and server service name to the requester. If sharing is enabled, the requester examines these values to determine whether they match the corresponding values for any existing conversation between the requester and a server. If a matching conversation is identified and the new transaction sequence can be assigned to the conversation without exceeding the implicit HLAPI/USS limit on the number of transaction sequences per conversation (10), then the requester assigns the new transaction sequence to the conversation. Otherwise, a new conversation is started and the new transaction sequence is assigned to it.
6. The **TIMEOUT_INTERVAL** PDB applies to the HLAPI that is running on Tivoli Information Management for z/OS. If you specify a timeout interval, it determines the maximum time that may elapse during HLAPI processing of a single transaction. If the HLAPI processing time exceeds the timeout interval, the HLAPI ends the transaction. Note that the HLAPI processing time for a transaction does not include time spent by the server or components of HLAPI/USS to process the transaction. Therefore, a transaction submitted from HLAPI/USS may appear to require more processing time than the timeout interval indicates.
7. For information about transaction logging, see “Transaction Logging” on page 312.
8. The client application program returns from the call to **IDBTransactionSubmit()**. The client application program can submit other transactions of the transaction sequence.

Any process of a client application program can have multiple, concurrent transaction sequences. The transactions of a transaction sequence can be submitted independently of the transactions of all other transaction sequences. However, a different HICA must be associated with each transaction sequence.

Terminate Tivoli Information Management for z/OS (HL02)

The HLAPI/USS transaction HL02 allows a client application program to close a database connection on a specific server. The HL02 transaction completes a normal HLAPI/USS transaction sequence. This section describes the steps performed for an HL02 transaction.

1. The client application program allocates and initializes the normal control PDBs requesting a disconnect from the database, associates the PDBs with the HICA for the transaction sequence, and calls **IDBTransactionSubmit()** to submit the HL02 transaction to HLAPI/USS.
2. The client interface communicates the disconnect request through the requester to the server. If this is the last transaction sequence assigned to the conversation between the requester and the server, the conversation with the server is ended. If this is not the last transaction sequence assigned to the conversation, the conversation is not ended.
3. The client application program returns from the call to **IDBTransactionSubmit()**. For an HL02 transaction, the transaction reply does not include any output PDBs, message PDBs, or error PDBs. However, the transaction reply does include the return and reason codes returned in the HICA fields **RETC** and **REAS**.

Retrieve Record (HL06)

The optional **TEXT_MEDIUM** control PDB can specify the type of storage medium for the HLAPI. However, the HLAPI/USS only supports storage medium type **B**. The **TEXT_MEDIUM** PDB is optional for the HLAPI/USS HL06 transactions. However, HLAPI/USS overrides any specified value with the value for storage medium type **B**.

If you want to retrieve freeform text as a continuous stream of data with carriage return / line feed characters (ASCII X'0D0A') after each text line, set the optional control PDB **TEXT_STREAM** to **YES**. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information about the **TEXT_STREAM** PDB.

Create Record (HL08)

HLAPI/USS does not support text data sets. Always specify a non-zero value for the **PDB_DATW** field of the input PDBs for text data.

If you are creating a record that contains freeform text, and the input text contains either line feed characters (ASCII X'0A') or carriage return / line feed characters (ASCII X'0D0A'), set the optional control PDB **TEXT_STREAM** to **YES**. This will ensure that text formatting information is stored in the record. When the text is retrieved, it will be formatted exactly as it was entered. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information about the **TEXT_STREAM** PDB.

Update Record (HL09)

HLAPI/USS does not support text data sets. Always specify a non-zero value for the **PDB_DATW** field of the input PDBs for text data.

If you are updating a record that contains freeform text, and the input text contains either line feed characters (ASCII X'0A') or carriage return / line feed characters (ASCII X'0D0A'), set the optional control PDB **TEXT_STREAM** to **YES**. This will ensure that text formatting information is stored in the record. When the text is retrieved, it will be formatted exactly as it was entered. The *Tivoli Information Management for z/OS Application Program Interface Guide* contains additional information about the **TEXT_STREAM** PDB.

Compiling and Linking Your Application to HLAPI/USS Services

The HLAPI/USS dll contains the client interface entry point and executable code accessed by calling the HLAPI/USS function **IDBTransactionSubmit()**. Before using the HLAPI/USS dll, you must compile and link your application. You must specify that you are using a dll and you must specify the HLAPI/USS definition side-deck. The default path and filename for the definition side-deck is

```
/usr/lpp/InfoMan/lib/blmhlapi.x
```

The default path and file names for the HLAPI/USS dll is:

```
/usr/lpp/InfoMan/lib/blmhlapi
```

However, installation of HLAPI/USS creates a symbolic link in the **/usr/lib** directory to reference the dll.

To compile and link a C program named blmappl.c using c89:

```
c89 -o blmappl -W c,dll blmappl.c blmhlapi.x -lc
```

assuming the C program is in the same path as **blmhlapi.x**.

Refer to the file blmappl.mak in the **HLAPI/USS** examples subdirectory (listed below) for an example of how to link your application.

```
/usr/lpp/InfoMan/examples
```

Planning Your HLAPI/USS Application

This section lists some questions you need to answer when you plan and design your application that uses HLAPI/USS. Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* for more information on this subject.

- Which Tivoli Information Management for z/OS transactions (for example, *create* or *update*) do you use?
- Which record types (for example, *problem* or *change*) do you use?
- Which fields (for example, *problem status* or *assignee name*) do you use?
- Do you need to connect to more than one Tivoli Information Management for z/OS database?
- On which MVS systems are your Tivoli Information Management for z/OS databases located?
- Does your application require security?
- Do you have port numbers assigned for the sockets any servers that support TCP/IP will need?
- On which OS/390 UNIX System Services systems will your client application programs run?
- On which OS/390 UNIX System Services systems will your requesters be located?
- Is there TCP/IP connectivity between the MVS systems running your MRES with TCP/IP servers and the MVS systems running your requesters and between the requester hosts and client interface hosts.
- How should your OS/390 UNIX System Services system resource limits be configured to support requester and client interface requirements?
- How are the processes of your application related to one another?
- How much storage do you need in a server to support sessions, given your transaction mix?
- Do you want to enable transaction logging by the client interface or by the server?
- Do you want to perform data validation? Data length validation is always performed for a transaction, but other validation can be controlled as described in the *Tivoli Information Management for z/OS Application Program Interface Guide*.

Converting HLAPI Programs to HLAPI/USS Programs

If you want to convert an existing C-language program that uses HLAPI to a C-language program that uses HLAPI/USS, here are some tips on how to do that:

- Make general modifications that are required to make the program run on OS/390 UNIX System Services:
  - Change HICA and PDB field names to their HLAPI/USS counterparts.
- Update references to included header files:
  - Include the HLAPI/USS header file **blmh.h**.
  - Include the HLAPI/USS header file **blmech.h** if you want to use the named constants for HLAPI/USS return and reason codes.

- Do not include the HLAPI header file **spc.h**.

- Convert any HL06, HL08, and HL09 transactions that use the data set method of freeform text processing to the buffer method of freeform text processing.
- Convert HLAPI function calls to the HLAPI/USS function call **IDBTransactionSubmit()**. Do not define variables to reference the **BLGYHLPI** module, and do not fetch the **BLGYHLPI** module.
- Add processing to allocate and initialize the three special types of control PDBs for HLAPI/USS (**SECURITY_ID**, **PASSWORD**, and **DATABASE_PROFILE**) and to insert the PDBs into the chain of control PDBs for each HL01 transaction.
- Create one or more database profiles for use with your program's transaction sequences.
- Create any system profiles needed for requesters.
- Review the error handling sections of the program to determine whether changes are needed to process HLAPI/USS error conditions.
- When starting the compiler and the linkage editor, compile with appropriate options for locating the HLAPI/USS header files and for linking with the HLAPI/USS definition side-deck.
- Remove asynchronous processing, because HLAPI/USS supports only synchronous processing.

Using the REXX HLAPI/USS Interface

The REXX HLAPI/USS interface enables you to access HLAPI/USS transactions from MVS REXX programs similar to the manner in which HLAPI/REXX enables you to access HLAPI transactions from MVS REXX programs. You should be familiar with HLAPI/REXX, described in the *Tivoli Information Management for z/OS Application Program Interface Guide*, before you attempt to use the REXX HLAPI/USS interface.

REXX HLAPI/USS allows you to write a REXX program that sets REXX variables with control and input information, then links to the HLAPI/USS through the REXX HLAPI/USS interface to process that information. On return, REXX HLAPI/USS uses data returned by the HLAPI/USS to set various REXX output variables in your program. The particular transactions that the REXX HLAPI/USS interface supports are the same as those supported by the HLAPI/USS. For a list of those transactions, see Table 7 on page 331.

The use of shared REXX variables for specifying control and input data to Tivoli Information Management for z/OS and returning output data from Tivoli Information Management for z/OS is equivalent to how this is implemented for HLAPI/REXX on MVS. Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* for information on how to define REXX variables in your program and for a list of reserved REXX variables that the REXX HLAPI interfaces (HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, and REXX HLAPI/USS) use.

Although HLAPI/REXX and REXX HLAPI/USS are quite similar and both run on an MVS host, their access to Tivoli Information Management for z/OS databases are significantly different. HLAPI/REXX uses the HLAPI to access the database and therefore it must run on the same MVS host as Tivoli Information Management for z/OS. REXX HLAPI/USS, however, uses HLAPI/USS which establishes a TCP/IP connection to an MRES with TCP/IP server and the server provides access to the database. This is a significant difference,

because it allows REXX HLAPI/USS to be installed and run on an MVS system that is remote from the MVS system where the Tivoli Information Management for z/OS database is located.

The remainder of this section discusses the operating differences between HLAPI/REXX and REXX HLAPI/USS.

REXX HLAPI/USS Installation and Setup

The REXX HLAPI/USS interface is an MVS load module that you call from your REXX program. The REXX HLAPI/USS is named **BLMYRXM** and is distributed with HLAPI/USS. After you use SMP/E to install the HLAPI/USS feature, BLMYRXM is located in the **SBLMMOD1** data set. A sample REXX program name **blmyrxsa** is also distributed with HLAPI/USS and is located in the HFS file **/usr/lpp/InfoMan/examples**. “Installing and Setting Up HLAPI/USS” on page 299 contains information on HLAPI/USS installation.

Invoking REXX HLAPI/USS

REXX HLAPI/USS is an MVS load module located in **SBLMMOD1**. The syntax of the REXX HLAPI/USS call is:

```
ADDRESS LINKMVS 'BLMYRXM stem'
```

where `stem` is the name of a high-level compound variable whose elements must contain the transaction name and the stem names of the control, input, and output compound variables. Note that the high-level stem name is passed as the only parameter to BLMYRXM; this differs from REXX HLAPI/2 and REXX HLAPI/AIX, where the transaction name and the control, input, and output stem names are all passed as parameters to blmyrxm.

The general form of the high-level compound variable is:

stem.1 = transaction name
stem.2 = control stem name
stem.3 = input stem name
stem.4 = output stem name
stem.0 = the number of the largest element of the compound variable

The following example illustrates a call to the REXX HLAPI/USS for a create transaction with an input stem name of INPUT, a control stem name of CONTROL, an output stem name of OUTPUT, and a high-level stem name of PARM.

```
PARM.0=4
PARM.1='CREATE'
PARM.2='CONTROL'
PARM.3='INPUT'
PARM.4='OUTPUT'
ADDRESS LINKMVS 'BLMYRXM PARM'
```

If the transaction you want to run does not require control data, input data, or output data, set the corresponding element value of the high level compound variable to null. For example, if you have control data and want to specify an output stem name, but you do not have input data, specify the high level stem element values as follows:

stem.0=*4*
stem.1=*transaction name*
stem.2=*control stem name*
stem.3=’’ (with no blanks between the quote marks to indicate a null value for the *input stem* name)
stem.4=*output stem name*

If the transaction returns output data, you do not have to specify an output stem name. REXX HLAPI/USS defaults the output stem name to **BLG_OUT**. If you want to use the default output stem name, specify the high level stem element values as follows:

stem.0=*3*
stem.1=*transaction name*
stem.2=*control stem name*
stem.3=*input stem name*

Because no output stem name was specified, the output stem name will take the default name of **BLG_OUT**.

REXX HLAPI/USS Transaction Names

The following REXX HLAPI/USS transaction names are supported. The list matches the list of HLAPI transactions and functions listed in Table 1 on page 3.

Table 7. REXX HLAPI/USS Transaction Names

NAME	FUNCTION
INIT	Initialize Tivoli Information Management for z/OS
TERM	Terminate Tivoli Information Management for z/OS
GETID	Obtain External Record ID
CHECKOUT	Check Out Record
CHECKIN	Check In Record
RETRIEVE	Retrieve Record
CREATE	Create Record
UPDATE	Update Record
CHANGE_APPROVAL	Change Record Approval
SEARCH	Record Inquiry
ADD_REL	Add Record Relations
DELETE	Delete Record
USERTSP	Start User TSP
GETDATAMODEL	Get Data Model

Running REXX HLAPI/USS from OS/390 UNIX System Services

Before you can run your REXX program, your OS/390 UNIX System Services environment needs to have several environment variables set:

- Your REXX programs need to reside in a directory specified in the **PATH** environment variable
- Environment variable **_BPX_SHAREAS** must be set to **YES**
- The **STEPLIB** environment variable must be set to point to the **SBLMMOD1** data set so that your REXX program can locate the **BLMYRXM** load module.

When you have finished creating your REXX program source file and wish to run it for the first time, make it executable using the `chmod +x` command

```
chmod +x name
```

Then run the REXX program by typing its filename at your shell command prompt.

Running REXX HLAPI/USS from MVS

In addition to being run from a REXX program in OS/390 UNIX System Services, as was described in "Running REXX HLAPI/USS from OS/390 UNIX System Services", a REXX

exec that resides in an MVS partitioned data set can be used to invoke the REXX HLAPI/USS interface. When you run your program from MVS, the **SBLMMOD1** data set needs to be in your logon procedure or available through either the linklist or through the LPA.

A REXX program that you run from MVS to call the REXX HLAPI/USS interface is similar to the sample illustrated in "Sample Program blmyrxsa" on page 333. However, the following change is required:

- Near the beginning of the listing, immediately before the
```
address syscall
```
  add the statement
```
Call syscalls('ON')
```
  This statement allows the REXX syscall commands used in the REXX exec to run.

REXX Reserved Variables

The REXX HLAPI/USS interface uses the same REXX reserved variables as those described for HLAPI/REXX in the *Tivoli Information Management for z/OS Application Program Interface Guide* with these additions and deletions.

REXX reserved variables added for REXX HLAPI/USS

BLG_HLAPIUSS_RC	Return code passed back from HLAPI/USS. The return codes are documented in "HLAPI Service Call Return Codes" on page 367.

HLAPI/REXX reserved variables not used by REXX HLAPI/USS

BLG_R15	replaced by BLG_HLAPIUSS_RC

BLG_RC and BLG_REAS

BLG_RC is set to the value of the **RETC** field of the HICA. BLG_REAS is set to the value of the **REAS** field. If BLG_RC is set to 16 (indicating an abend), BLG_REAS is set to `xxsssuuu`, where `sss` is an abend code, `uuu` is a reason code, and `xx` are do-not-care positions.

Other Considerations

- Neither HLAPI/USS nor REXX HLAPI/USS supports asynchronous processing; all transactions are processed synchronously. Synchronous processing forces the REXX program's current process to wait for a Tivoli Information Management for z/OS transaction to finish before it can perform any other work.
- REXX HLAPI/USS requires three additional control variables on the INIT transaction. The REXX variable names are:
  - **SECURITY_ID** to specify the MVS user ID
  - **PASSWORD** to specify the MVS user password
  - **DATABASE_PROFILE** to specify the name of the database profile
- Neither HLAPI/USS nor REXX HLAPI/USS supports text data sets.

- For a **RETRIEVE** transaction, the REXX variable **TEXT_MEDIUM** only supports storage medium type **B**.
- For a **CREATE** or **UPDATE** transaction, **text-name.?width** for text data must be nonzero or not be specified.

Refer to the *Tivoli Information Management for z/OS Application Program Interface Guide* for information on how to define REXX variables in your program and for a list of reserved REXX variables that the REXX HLAPI interfaces use.

REXX HLAPI/USS Sample REXX Program

A sample REXX program named **blmyrxsa** is distributed with HLAPI/USS. After installation, **blmyrxsa** is located in **/usr/lpp/InfoMan/examples**. This sample REXX program illustrates how to:

- Setup REXX variables
- Make REXX HLAPI/USS transaction calls
- Retrieve output data

"Sample Program blmyrxsa" also shows the **blmyrxsa** sample REXX program. The following steps are performed:

1. Open the HFS file named **/usr/lpp/InfoMan/examples/blmyrxsa.out**; this is where REXX HLAPI/USS output will be written.
2. Setup REXX variables for an **INIT** transaction.
3. Call REXX HLAPI/USS to perform the **INIT** transaction.
4. Record REXX HLAPI/USS output to the HFS file opened in step 1.
5. Setup REXX variables to **CREATE** a record with record id **SAMP1**.
6. Call REXX HLAPI/USS to perform the **CREATE** transaction.
7. Record REXX HLAPI/USS output to the HFS file opened in step 1.
8. Setup REXX variables to retrieve the record just created.
9. Call REXX HLAPI/USS to perform the **RETRIEVE** transaction.
10. Record REXX HLAPI/USS output to the HFS file opened in step 1.
11. Setup REXX variables to delete the record just created.
12. Call REXX HLAPI/USS to perform the **DELETE** transaction.
13. Record REXX HLAPI/USS output to the HFS file opened in step 1.
14. Call REXX HLAPI/USS to perform the **TERM** transaction.
15. Close the HFS file opened in step 1.

Note: The **lineout** function that is used to write data to an HFS file is not a built-in REXX function. A **lineout** subroutine is provided at the end of the **blmyrxsa** sample to show how data can be written to an HFS file. The HFS file must be opened before **lineout** is used to write to it, and it must be closed before exiting the REXX program.

Sample Program blmyrxsa

The sample program **blmyrxsa** demonstrates calls to the REXX HLAPI/USS interface. In this example, **blmyrxsa** is intended to run from OS/390 UNIX System Services. Your REXX

program can also be run from TSO. Refer to “Running REXX HLAPI/USS from MVS” on page 331 for information on how this sample can be changed to enable it to run from TSO.

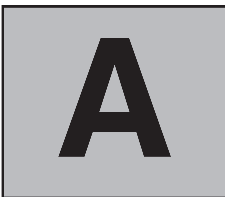
```
/* rexx */
/*------------------------------------------------------------------*/
/*                                                                  */
/* This sample REXX program demonstrates calls to the REXX HLAPI/USS */
/* interface.  The program shows how:                               */
/*                                                                  */
/*  - REXX HLAPI/USS variables are set for transactions             */
/*                                                                  */
/*  - INIT, CREATE, RETRIEVE, DELETE and TERM REXX HLAPI/USS        */
/*    transactions are issued                                       */
/*                                                                  */
/*  - a log of transaction output is kept in file:                  */
/*      /usr/lpp/InfoMan/examples/blmyrxsa.out                    */
/*                                                                  */
/*------------------------------------------------------------------*/

address syscall

"open /usr/lpp/InfoMan/examples/blmyrxsa.out"  ,
       o_trunc+o_creat+o_wronly 755
file_id = retval

  /******************************************/
  /* set the CONTROL data for the INIT      */
  /******************************************/

  CONTROL.0 = 10
  CONTROL.1 = 'database_profile'
  CONTROL.2 = 'class_count'
  CONTROL.3 = 'hlimsg_option'
  CONTROL.4 = 'security_id'
  CONTROL.5 = 'password'
  CONTROL.6 = 'spool_interval'
  CONTROL.7 = 'apimsg_option'
  CONTROL.8 = 'application_id'
  CONTROL.9 = 'privilege_class'
  CONTROL.10= 'session_member'

  PRIVILEGE_CLASS = 'MASTER'
  SESSION_MEMBER  = 'BLGSES00'
  APPLICATION_ID  = 'SAMPID'
  CLASS_COUNT     = 1
  SPOOL_INTERVAL  = 200
  HLIMSG_OPTION   = 'B'
  APIMSG_OPTION   = 'B'
  SECURITY_ID     = 'SAMPID'
  PASSWORD        = 'PASSWORD'
  DATABASE_PROFILE = '/usr/lpp/InfoMan/examples/blmdb.pro'

  /******************************************/
  /* call REXX HLAPI/USS to perform the     */
  /* INIT transaction                       */
  /******************************************/
  PARM.0=2
  PARM.1='INIT'
  PARM.2='CONTROL'
  ADDRESS   LINKMVS  'BLMYRXM PARM'
  rexx_rc = RC

  /******************************************/
  /* record REXX HLAPI/USS output in file   */
  /* blmyrxsa.out                           */
  /******************************************/
  say 'Returned from INIT with RC = ' rexx_rc
```

```
rc = lineout('INIT Transaction results:')
rc = lineout('rexx_rc=' rexx_rc)
rc = lineout('BLG_RC=' BLG_RC)
rc = lineout('BLG_REAS=' BLG_REAS)
rc = lineout('BLG_VARNAME=' BLG_VARNAME)
rc = lineout('BLG_HLAPIUSS_RC=' BLG_HLAPIUSS_RC)
rc = lineout(' ')
say ''

if rexx_rc \= 0 then
  do
   PARM.0=1
   PARM.1='TERM'
   ADDRESS   LINKMVS  'BLMYRXM parm'
   exit
  end

/*******************************************/
/* set the CONTROL and INPUT data for     */
/* the CREATE                             */
/*******************************************/
CONTROL = ''
CONTROL.0 =  1
CONTROL.1 = 'PIDT_NAME'
PIDT_NAME = 'BLGYPRC'

INPUT = ''
INPUT.0 = 6
INPUT.1.?NAME = 'S0BEE'
INPUT.1.?PROC = 'V'
INPUT.2.?NAME = 'S0B59'
INPUT.3.?NAME = 'S0CA9'
INPUT.4.?NAME = 'S0E0F'
INPUT.5.?NAME = 'S0CCF'
INPUT.6.?NAME = 'S0E01.'
S0E01.?WIDTH = 20
S0BEE = 'INITIAL'
S0B59 = 'DOE/JOHN'
S0CA9 = 'LPT1'
S0E0F = 'PROBLEM RECORD CREATE BY REXX HLAPI/USS'
S0CCF = 'SAMP1'
S0E01.0  = 2
S0E01.1  = 'Sample1 first line'
S0E01.2  = 'Sample1 second line'
OUTPUT.0 = 1
OUTPUT.1.?TYPE = ' '

/*******************************************/
/* call REXX HLAPI/USS to perform the     */
/* CREATE transaction                     */
/*******************************************/
PARM.0=4
PARM.1='CREATE'
PARM.2='CONTROL'
PARM.3='INPUT'
PARM.4='OUTPUT'
ADDRESS   LINKMVS  'BLMYRXM PARM'
rexx_rc = RC

/*******************************************/
/* record REXX HLAPI/USS output in file   */
/* blmyrxsa.out                           */
/*******************************************/
say 'Returned from CREATE with RC = ' RC
rc = lineout('CREATE Transaction results:')
rc = lineout('rexx_rc=' rexx_rc)
rc = lineout('BLG_RC=' BLG_RC)
```

```
rc = lineout('BLG_REAS=' BLG_REAS)
rc = lineout('BLG_VARNAME=' BLG_VARNAME)
rc = lineout('BLG_HLAPIUSS_RC=' BLG_HLAPIUSS_RC)
if BLG_ERRCODE.0 \= 0 then
  do
    rc = lineout('REXX Error Variables')
    do i = 1 to BLG_ERRCODE.0
     rc = lineout('Error Name ' i ' = ' BLG_ERRCODE.i.?NAME)
     rc = lineout('Error Code ' i ' = ' BLG_ERRCODE.i.?CODE)
    end
  end
if BLG_MSGS.0 \= 0 then
  do
    rc = lineout('Messages')
    do i = 1 to BLG_MSGS.0
     rc = lineout('Message ' i ' = ' BLG_MSGS.i.?NAME)
    end
  end
rc = lineout(' ')
say ''

if rexx_rc \= 0 then
  do
   PARM.0=1
   PARM.1='TERM'
   ADDRESS   LINKMVS  'BLMYRXM parm'
   exit
  end

/*******************************************/
/* set the CONTROL data for the RETRIEVE  */
/*******************************************/
CONTROL = ''
CONTROL.0 =  3
CONTROL.1 = 'PIDT_NAME'
CONTROL.2 = 'TEXT_OPTION'
CONTROL.3 = 'RNID_SYMBOL'

PIDT_NAME = 'BLGYPRR'
TEXT_OPTION = 'YES'
RNID_SYMBOL = 'SAMP1'

/*******************************************/
/* call REXX HLAPI/USS to perform the     */
/* RETRIEVE transaction                   */
/*******************************************/
PARM.0=4
PARM.1='RETRIEVE'
PARM.2='CONTROL'
PARM.3=''
PARM.4='OUTPUT'
ADDRESS   LINKMVS  'BLMYRXM PARM'
rexx_rc = RC

/*******************************************/
/* record REXX HLAPI/USS output in file   */
/* blmyrxsa.out                           */
/*******************************************/
say 'Returned from RETRIEVE with RC = ' RC
rc = lineout('RETRIEVE Transaction results:')
rc = lineout('rexx_rc='                     rexx_rc)
rc = lineout('BLG_RC=' BLG_RC)
rc = lineout('BLG_REAS=' BLG_REAS)
rc = lineout('BLG_VARNAME=' BLG_VARNAME)
rc = lineout('BLG_HLAPIUSS_RC=' BLG_HLAPIUSS_RC)
rc = lineout('REXX Output Variables')
 do i = 1 to OUTPUT.0
```

```
 otype = OUTPUT.i.?TYPE
 tname = OUTPUT.i.?NAME
 rc = lineout('Name- ' tname '  Type- ' otype)
 /******************************************/
 /* if the output is freeform text, put   */
 /* out all the lines                     */
 /******************************************/
 if otype = 'X' then
  do j= 1 to OUTPUT.tname.0
   rc = lineout('Data- ' OUTPUT.tname.j)
  end
 else
  rc = lineout('Data- ' OUTPUT.tname)
end
if BLG_MSGS.0 \= 0 then
  do
    rc = lineout('Messages')
    do i = 1 to BLG_MSGS.0
     rc = lineout('Message ' i ' = ' BLG_MSGS.i.?NAME)
    end
  end
rc = lineout(' ')
say ''

if rexx_rc \= 0 then
  do
   PARM.0=1
   PARM.1='TERM'
   ADDRESS   LINKMVS  'BLMYRXM parm'
   exit
  end

/*******************************************/
/* set the CONTROL data for the DELETE    */
/*******************************************/
CONTROL = ''
CONTROL.0 =  1
CONTROL.1 = 'RNID_SYMBOL'

RNID_SYMBOL = 'SAMP1'

/*******************************************/
/* call REXX HLAPI/USS to perform the     */
/* DELETE transaction                     */
/*******************************************/
PARM.0=2
PARM.1='DELETE'
PARM.2='CONTROL'
ADDRESS   LINKMVS  'BLMYRXM PARM'
rexx_rc = RC

/*******************************************/
/* record REXX HLAPI/USS output in file   */
/* blmyrxsa.out                           */
/*******************************************/
say 'Returned from DELETE with RC = ' RC
rc = lineout('DELETE Transaction results:')
rc = lineout('rexx_rc=' rexx_rc)
rc = lineout('BLG_RC=' BLG_RC)
rc = lineout('BLG_REAS=' BLG_REAS)
rc = lineout('BLG_VARNAME=' BLG_VARNAME)
rc = lineout('BLG_HLAPIUSS_RC=' BLG_HLAPIUSS_RC)
say ''

/*******************************************/
/* call REXX HLAPI/USS to perform the     */
/* TERM transaction                       */
```

```
  /*******************************************/
     RC = 4
     PARM.0=1
     PARM.1='TERM'
     ADDRESS   LINKMVS  'BLMYRXM parm'

 "close " file_id
Exit

lineout:
 parse arg string;
 printit = string || esc_n
 "write " file_id "printit"
return(0)
```



Components of Tivoli Information Management for z/OS Clients

Components of HLAPI/2

HLAPI/2 files that you install on the workstation reside in the default directory C:\INFOAPI, unless you changed this default during installation. The HLAPI/2 files that are installed on a LAN Server can be found in the directory C:\INFOAPIS.

Files on the Workstation

Installation and Maintenance Component

- Root directory
  - CONFIG.ADD
- OS/2 System Directory
  - EPFIS.INI
  - EPFICAT.PKG
  - EPFIHCNF.CNF
- INFOAPI directory
  - EPFINSTS.EXE
  - EPFIPRCS.EXE
  - EPFIPII.DLL
  - EPFIEXTS.DLL
  - EPFIRSBK.DLL
  - EPFIHPLB.HLP
  - EPFIMSG.MSG
  - EPFIDLDS.EXE
  - EPFIUPK2.EXE
  - EPFIICIS.ICO
  - EPFIHELP.INF
  - EPFISINC.PKG
  - READ.ME

Toolkit Component

- INFOAPI directory
  - BLMIPKG.PKG
  - BLMICF.ICF
  - \H
    - IDBHLAPI.LIB
    - IDBECH.H
    - IDBH.H

- \SAMPLE\C
  - BLM2SAM1.C
  - BLM2SAMI.CMD
- \SAMPLE\CPPWRAP
  - BLMYCWHI.CPP
  - BLMYCWHI.HPP
  - BLMYCWPD.CPP
  - BLMYCWPD.HPP
  - BLMYCWIS.CPP
  - BLMYCWIS.HPP
  - BLMYCWC.HPP
  - BLMYCWS1.CPP
  - BLMYCWRP.MAK
  - BLMYCWS1.MAK
  - BLMYCWRP.DEF
  - BLMYCWRP.CMD
  - BLMYCWS1.CMD
- \SAMPLE\REXX
  - BLMYRXSA.CMD

Run-Time Component

- INFOAPI directory
  - BLMIPKG.PKG
  - IDBREQ.EXE
  - IDBREQB.EXE
  - IDBREQA.EXE
  - IDBREQT.EXE
  - IDBMSG.MSG
  - BLMICF.ICF
  - BLMREQI.CMD
- \DLL
  - IDBHLAPI.DLL
  - BLMYRXM.DLL
- \SAMPLE
  - DATABASE.PRO
  - SYSTEM.PRO

Files on the LAN Server

Installation and Maintenance

INFOAPIS directory

- BLMIPDL.PKG
- INSTALL.EXE
- EPFIUPK2.EXE
- EPFINSTS.EXE
- EPFIPRCS.EXE
- EPFIPII.DLL
- EPFIEXTS.DLL
- EPFIRSBK.DLL
- EPFIHPLB.HLP
- EPFIMSG.MSG

- EPFIDLDS.EXE
- EPFIICIS.ICO
- EPFIHELP.INF
- INSTALL.IN_
- BLMIPKG.PKG
- EPFISINC.PKG
- BLMIDDL.DSC
- BLMICF.ICF
- READ.ME

HLAPI/2 Base Files

INFOAPIS directory

- BLMITKPK.PA_
- BLMIRTPK.PA_

Components of HLAPI/CICS

HLAPI/CICS client has load modules and a sample program library. The sample program library has both source and compiled versions.

| Members in the CICS/client system load library (BLMCICS.VxRxMx.SBLMMOD1):

- BLMYKINF
- BLMYKCOM
- BLMYKTRM
- BLMYKMNU
- BLMYKCTL
- BLMYKCRE
- BLMYKRTV
- BLMYKDEL
- BLMMAPS

| Members in the sample source (VS COBOL II) library (BLMCICS.VxRxMx.SBLMSAMP):

- BLMYKMNU
- BLMYKCTL
- BLMYKCRE
- BLMYKRTV
- BLMYKDEL
- BLMYKMAP

| Members in the sample compiled library - Objects (BLMCICS.VxRxMx.SBLMTXT1):

- BLMYKMNU
- BLMYKCTL
- BLMYKCRE
- BLMYKRTV
- BLMYKDEL
- BLMYKMAP

| Members in the message library - Objects (BLMCICS.VxRxMx.SBLMSRC1):

- BLMYKMSG

Components of HLAPI/NT

HLAPI/NT files that you install on the workstation reside in the default directory C:\INFOAPI, unless you changed this default during installation. The HLAPI/NT files that are installed on a network drive can be found in the directory C:\INFOAPI.

Files on the Workstation

Installation Files

- INFOAPI directory
  - DeIsL1.isu
  - DeIsL2.isu
  - DeIsL3.isu
  - UNINSTAL.EXE
  - UNINST.EXE

Base Files

- INFOAPI directory
  - README.TXT

Toolkit Component

- INFOAPI directory
  - \H
    - IDBECH.H
    - IDBH.H
    - IDBHLAPI.LIB
  - \JAVA
    - Blmyjwc.class
    - Blmyjws1.java
    - Hicao.class
    - NameNotFoundException.class
    - Pdbo.class
  - \SAMPLE\C
    - BLM2SAMI.BAT
    - BLM2SAM1.C
    - BLM2SAMM.BAT
  - \SAMPLE\CPPWRAP
    - BLMYCWC.HPP
    - BLMYCWHI.CPP
    - BLMYCWHI.HPP
    - BLMYCWIS.CPP
    - BLMYCWIS.HPP
    - BLMYCWPD.CPP
    - BLMYCWPD.HPP
    - BLMYCWRP.BAT
    - BLMYCWRP.MAK
    - BLMYCWS1.BAT
    - BLMYCWS1.CPP
  - \LOCALE\ICONV
    - ICONV.LST
    - UCSTBL.DLL

  - UTF-8.DLL
- \LOCALE\UCONVTAB
  - IBM-037
  - IBM-1004
  - IBM-1006
  - IBM-1008
  - IBM-1009
  - IBM-1010
  - IBM-1011
  - IBM-1012
  - IBM-1013
  - IBM-1014
  - IBM-1015
  - IBM-1016
  - IBM-1017
  - IBM-1018
  - IBM-1019
  - IBM-1025
  - IBM-1026
  - IBM-1027
  - IBM-1028
  - IBM-1038
  - IBM-1040
  - IBM-1041
  - IBM-1042
  - IBM-1043
  - IBM-1046
  - IBM-1047
  - IBM-1050
  - IBM-1051
  - IBM-1088
  - IBM-1089
  - IBM-1092
  - IBM-1097
  - IBM-1098
  - IBM-1112
  - IBM-1114
  - IBM-1115
  - IBM-1116
  - IBM-1117
  - IBM-1118
  - IBM-1119
  - IBM-1122
  - IBM-1123
  - IBM-1124
  - IBM-1250
  - IBM-1251
  - IBM-1252
  - IBM-1253
  - IBM-1254
  - IBM-1255
  - IBM-1256

- IBM-1257
- IBM-1275
- IBM-1276
- IBM-1277
- IBM-1350
- IBM-1380
- IBM-1381
- IBM-1382
- IBM-1383
- IBM-256
- IBM-259
- IBM-273
- IBM-274
- IBM-277
- IBM-278
- IBM-280
- IBM-282
- IBM-284
- IBM-285
- IBM-287
- IBM-290
- IBM-293
- IBM-297
- IBM-300
- IBM-301
- IBM-361
- IBM-363
- IBM-367
- IBM-382
- IBM-383
- IBM-385
- IBM-386
- IBM-387
- IBM-388
- IBM-389
- IBM-391
- IBM-392
- IBM-393
- IBM-394
- IBM-395
- IBM-420
- IBM-423
- IBM-424
- IBM-437
- IBM-4948
- IBM-4951
- IBM-4952
- IBM-4960
- IBM-500
- IBM-5037
- IBM-5039
- IBM-5048

- IBM-5049
- IBM-5067
- IBM-5142
- IBM-5478
- IBM-813
- IBM-819
- IBM-829
- IBM-833
- IBM-834
- IBM-835
- IBM-836
- IBM-837
- IBM-838
- IBM-850
- IBM-851
- IBM-852
- IBM-855
- IBM-856
- IBM-857
- IBM-860
- IBM-861
- IBM-8612
- IBM-862
- IBM-863
- IBM-864
- IBM-865
- IBM-866
- IBM-868
- IBM-869
- IBM-870
- IBM-871
- IBM-874
- IBM-875
- IBM-880
- IBM-891
- IBM-895
- IBM-896
- IBM-897
- IBM-903
- IBM-9030
- IBM-904
- IBM-905
- IBM-9056
- IBM-9066
- IBM-907
- IBM-909
- IBM-910
- IBM-912
- IBM-913
- IBM-914
- IBM-9145
- IBM-915

- IBM-916
- IBM-918
- IBM-919
- IBM-920
- IBM-921
- IBM-922
- IBM-927
- IBM-930
- IBM-933
- IBM-935
- IBM-937
- IBM-939
- IBM-941
- IBM-942
- IBM-943
- IBM-946
- IBM-947
- IBM-948
- IBM-949
- IBM-950
- IBM-951
- IBM-952
- IBM-955
- IBM-960
- IBM-961
- IBM-963
- IBM-964
- IBM-970
- IBM-971
- IBMSBDCN
- IBMSBDTW
- IBMUDCCN
- IBMUDCJP
- IBMUDCTW
- X2081983

Requester Component

- INFOAPI directory
  - IDBREQ.EXE (TCP/IP only)
    or
  - IDBREQB.EXE (TCP/IP and APPC)
  - \DLL
    - BLMYJWRP.DLL
    - IDBDLL.DLL
    - IDBHLAPI.DLL
  - \SAMPLE
    - DATABASE.PRO
    - SYSTEM.PRO
  - \LOCALE\ICONV

  - ICONV.LST
  - UCSTBL.DLL
  - UTF-8.DLL
- \LOCALE\UCONVTAB
  - IBM-037
  - IBM-850

Files on the Network Server

Installation Files

■ INFOAPI directory

- DeIsL1.isu
- DeIsL2.isu
- DeIsL3.isu
- DeIsL4.isu
- \DISK1
  - _INST32I.EX_
  - _ISDEL.EXE
  - _SETUP.DLL
  - _SETUP.LIB
  - DISK1.ID
  - SETUP.BMP
  - SETUP.EXE
  - SETUP.INI
  - SETUP.INS
  - SETUP.ISS
  - SETUP.PKG
  - UNINST.EXE
  - UNINSTAL.EXE
- \DISK2
  - DISK2.ID
- \DISK3
  - DISK3.ID
- \DISK4
  - DISK4.ID

Base Files

■ INFOAPI directory

- \DISK1
  - README.TXT

Toolkit Component

■ INFOAPI directory

- \DISK2
  - INFOT.1
- \DISK3
  - INFOT.2

Requester Component

- INFOAPI directory
  - \DISK4
    - INFOR.Z
    - INFORB.Z

Components of HLAPI/AIX

The following files, directories, and symbolic links are created during the installation of the requester option and client interface option of HLAPI/AIX.

Requester Option

Directories

- /usr/lpp/idbhlapi
- /usr/lpp/idbhlapi/bin
- /usr/lpp/idbhlapi/examples
- /usr/lpp/idbhlapi/nls
- /usr/lpp/idbhlapi/nls/msg
- /usr/lpp/idbhlapi/nls/msg/En_US

Files

- /usr/lpp/idbhlapi/idbinsta
- /usr/lpp/idbhlapi/bin/idbreq
- /usr/lpp/idbhlapi/bin/idbreqt
- /usr/lpp/idbhlapi/examples/idbsys.pro
- /usr/lpp/idbhlapi/nls/msg/En_US/idbreq.cat

Symbolic links

- /usr/lib/nls/msg/En_US/idbreq.cat -> /usr/lpp/idbhlapi/nls/msg/En_US/idbreq.cat
- /usr/lib/nls/msg/prime/idbreq.cat -> /usr/lpp/idbhlapi/nls/msg/En_US/idbreq.cat

Client Interface Option

Directories

- /usr/lpp/idbhlapi
- /usr/lpp/idbhlapi/examples
- /usr/lpp/idbhlapi/include
- /usr/lpp/idbhlapi/lib
- /usr/lpp/idblhapi/java
- /usr/lpp/idbhlapi/nls
- /usr/lpp/idbhlapi/nls/msg
- /usr/lpp/idbhlapi/nls/msg/En_US

Files

- /usr/lpp/idbhlapi/examples/blmyrxsa
- /usr/lpp/idbhlapi/examples/idbappl.c
- /usr/lpp/idbhlapi/examples/idbappl.mak
- /usr/lpp/idbhlapi/examples/idbdb.pro
- /usr/lpp/idbhlapi/include/idbech.h
- /usr/lpp/idbhlapi/include/idbh.h
- /usr/lpp/idbhlapi/java/Blmyjwc.class

- /usr/lpp/idbhlapi/java/Blmyjws1.java
- /usr/lpp/idbhlapi/java/Hicao.class
- /usr/lpp/idbhlapi/java/NameNotFoundException.class
- /usr/lpp/idbhlapi/java/Pdbo.class
- /usr/lpp/idbhlapi/lib/blmyrxm
- /usr/lpp/idbhlapi/lib/libidb.a
- /usr/lpp/idbhlapi/lib/libblmyjwrp.so
- /usr/lpp/idbhlapi/nls/msg/En_US/idbcli.cat

Symbolic links

- /usr/include/idbech.h -> /usr/lpp/idbhlapi/include/idbech.h
- /usr/include/idbh.h -> /usr/lpp/idbhlapi/include/idbh.h
- /usr/lib/libidb.a -> /usr/lpp/idbhlapi/lib/libidb.a
- /usr/lib/libblmyjwrp.so -> /usr/lpp/idbhlapi/lib/libblmyjwrp.so
- /usr/lib/nls/msg/En_US/idbcli.cat -> /usr/lpp/idbhlapi/nls/msg/En_US/idbcli.cat
- /usr/lib/nls/msg/prime/idbcli.cat -> /usr/lpp/idbhlapi/nls/msg/En_US/idbcli.cat

Components of HLAPI/HP

The following files, directories, and symbolic links are created during the installation of the requester option and client interface option of HLAPI/HP.

Requester Option

Directories

- */idbhlapi
- */idbhlapi/bin
- */idbhlapi/examples
- */idbhlapi/nls
- */idbhlapi/nls/C

Files

- */idbhlapi/idbinstl
- */idbhlapi/bin/idbreq
- */idbhlapi/examples/idbsys.pro
- */idbhlapi/nls/C/idbreq.cat

Symbolic links

- /usr/lib/nls/C/idbreq.cat -> */idbhlapi/nls/C/idbreq.cat

* is the directory where the HLAPI/UNIX is installed.

Client Interface Option

Directories

- */idbhlapi
- */idbhlapi/examples
- */idbhlapi/include
- */idbhlapi/java
- */idbhlapi/lib
- */idbhlapi/nls
- */idbhlapi/nls/C

Files

- */idbhlapi/idbinstl
- */idbhlapi/examples/idbappl.c
- */idbhlapi/examples/idbappl.mak
- */idbhlapi/examples/idbdb.pro
- */idbhlapi/include/idbech.h
- */idbhlapi/include/idbh.h
- */idbhlapi/java/Blmyjwc.class
- */idbhlapi/java/Blmyjws1.java
- */idbhlapi/java/Hicao.class
- */idbhlapi/java/NameNotFoundException.class
- */idbhlapi/java/Pdbo.class
- */idbhlapi/lib/libidb.sl
- */idbhlapi/lib/libblmyjwrp.sl
- */idbhlapi/nls/C/idbcli.cat

Symbolic links

- /usr/include/idbech.h -> */idbhlapi/include/idbech.h
- /usr/include/idbh.h -> */idbhlapi/include/idbh.h
- /usr/lib/libidb.sl -> */idbhlapi/lib/libidb.sl
- /usr/lib/libblmyjwrp.sl -> */idbhlapi/lib/libblmyjwrp.sl
- /usr/lib/nls/C/idbcli.cat -> */idbhlapi/nls/C/idbcli.cat

Other Files

When you installed the HLAPI/HP, if you did not choose the option to remove the `tar` files after installation, you also need to delete these files:

- */idbhlapi/idbcli.tar
- */idbhlapi/idbreq.tar
- */idbhlapi/idball.tar
- */idbhlapi.tar

Components of HLAPI/Solaris

The following files, directories, and symbolic links are created during the installation of the requester option and client interface option of HLAPI/Solaris.

Requester Option

Directories

- */idbhlapi
- */idbhlapi/bin
- */idbhlapi/examples
- */idbhlapi/locale
- */idbhlapi/locale/C
- */idbhlapi/locale/C/LC_MESSAGES

Files

- */idbhlapi/idbinstl
- */idbhlapi/bin/idbreq
- */idbhlapi/examples/idbsys.pro
- */idbhlapi/locale/C/LC_MESSAGES/idbreq.cat

Symbolic links

- /usr/lib/locale/C/LC_MESSAGES/idbreq.cat -> */idbhlapi/locale/C/LC_MESSAGES/idbreq.cat

* is the directory where the HLAPI/UNIX is installed.

Client Interface Option

Directories

- */idbhlapi
- */idbhlapi/examples
- */idbhlapi/include
- */idbhlapi/java
- */idbhlapi/lib
- */idbhlapi/locale
- */idbhlapi/locale/C
- */idbhlapi/locale/C/LC_MESSAGES

Files

- */idbhlapi/idbinstl
- */idbhlapi/examples/idbappl.c
- */idbhlapi/examples/idbappl.mak
- */idbhlapi/examples/idbdb.pro
- */idbhlapi/include/idbech.h
- */idbhlapi/include/idbh.h
- */idbhlapi/java/Blmyjwc.class
- */idbhlapi/java/Blmyjws1.java
- */idbhlapi/java/Hicao.class
- */idbhlapi/java/NameNotFoundException.class
- */idbhlapi/java/Pdbo.class
- */idbhlapi/lib/libidb.so
- */idbhlapi/lib/libblmyjwrp.so
- */idbhlapi/locale/C/LC_MESSAGES/idbcli.cat

Symbolic links

- /usr/include/idbech.h -> */idbhlapi/include/idbech.h
- /usr/include/idbh.h -> */idbhlapi/include/idbh.h
- /usr/lib/libidb.so -> */idbhlapi/lib/libidb.so
- /usr/lib/libblmyjwrp.so -> */idbhlapi/lib/libblmyjwrp.so
- /usr/lib/locale/C/LC_MESSAGES/idbcli.cat -> */idbhlapi/locale/C/LC_MESSAGES/idbcli.cat

Other Files

When you installed the HLAPI/Solaris, if you did not choose the option to remove the `tar` files after installation, you also need to delete these files:

- */idbhlapi/idbcli.tar
- */idbhlapi/idbreq.tar
- */idbhlapi/idball.tar
- */idbhlapi.tar

Components of HLAPI/USS

The following files, directories, and symbolic links are created during HLAPI/USS installation.

Directories

- /usr/lpp/InfoMan
- /usr/lpp/InfoMan/bin
- /usr/lpp/InfoMan/examples
- /usr/lpp/InfoMan/include
- /usr/lpp/InfoMan/lib
- /usr/lpp/InfoMan/nls
- /usr/lpp/InfoMan/nls/msg
- /usr/lpp/InfoMan/nls/msg/C

Files

- /usr/lpp/InfoMan/bin/blmreq
- /usr/lpp/InfoMan/examples/blmappl.c
- /usr/lpp/InfoMan/examples/blmappl.mak
- /usr/lpp/InfoMan/examples/blmdb.pro
- /usr/lpp/InfoMan/examples/blmsys.pro
- /usr/lpp/InfoMan/examples/blmyrxsa
- /usr/lpp/InfoMan/include/blmech.h
- /usr/lpp/InfoMan/include/blmh.h
- /usr/lpp/InfoMan/lib/blmhlapi
- /usr/lpp/InfoMan/lib/blmhlapi.x
- /usr/lpp/InfoMan/nls/msg/C/blmcli.cat
- /usr/lpp/InfoMan/nls/msg/C/blmreqoe.cat

Symbolic links

- /usr/include/blmech.h -> /usr/lpp/InfoMan/include/blmech.h
- /usr/include/blmh.h -> /usr/lpp/InfoMan/include/blmh.h
- /usr/lib/blmhlapi -> /usr/lpp/InfoMan/lib/blmhlapi
- /usr/lib/blmhlapi.x -> /usr/lpp/InfoMan/lib/blmhlapi.x
- /usr/lib/nls/msg/C/blmcli.cat -> /usr/lpp/InfoMan/nls/msg/C/blmcli.cat
- /usr/lib/nls/msg/C/blmreqoe.cat -> /usr/lpp/InfoMan/nls/msg/C/blmreqoe.cat

B

Tivoli Information Management for z/OS Java Wrappers (HLAPI for Java)

The following classes were written to allow programmers to more easily write HLAPI applications using Java. They are based on the existing HLAPI programming model, but simplify that model by providing some of the common programming functions. This section describes the classes and their methods.

The following software must be installed to use the Tivoli Information Management for z/OS Java wrapper classes:

- Tivoli Information Management for z/OS HLAPI Client

In order to use them, you must have the Java Runtime Environment (JRE) Release 1.1 or later. If you are developing Java applications, you must also have the Java Development Kit (JDK) Release 1.1 or later.

To use the Tivoli Information Management for z/OS Java wrapper classes, you must make them accessible to the Java environment. To do this, add the directory in which they are located to the CLASSPATH environment variable, or copy the class files into a directory that is already listed in the CLASSPATH environment variable.

Hicao.class

This class contains methods and data needed to manipulate HICA field data and collections of PDBs. The application writer uses the Hicao in their Java application to replace code for common programming functions they would normally have to write. Functions include the following:

- Obtaining storage and freeing storage for the HICA and the PDB data structures
- Initializing fields and clearing fields in the HICA and the PDB structures
- Chaining PDBs to the HICA and removing PDBs from the HICA
- Pointer (index) movement within the PDB chains
- Retrieving and setting HICA and PDB field information
- Copying HICA and PDB structures
- Accessing information about the HICA and PDB, such as the current count of control PDBs
- Submitting the transaction to HLAPI

Refer to "Class Hicao" for a detailed description of the Hicao methods.

Pdbo.class
This class contains methods for storing data in and retrieving data from the PDB data structure. Functions include the following:

- Setting, retrieving, and initializing PDB fields
- Copying a PDB

Refer to "Class Pdbo" on page 361 for a detailed description of the Pdbo methods.

Blmyjwc.class
This class contains constants used for control PDB names. "Class Blmyjwc" on page 363 contains a list of the constants.

NameNotFoundException.class
Exception class for cases where a PDB object cannot be found whose name matches a specified value.

Blmyjws1.java
Java source code for sample program

Class Hicao

The public class Hicao is a class file used to implement HICA data structure. The function of the Hicao class is to provide methods for implementing PDB chains and methods for submitting the HLAPI transactions.

Hicao()
Constructor for the Hicao object.

addAsLastControl(String,String,char)
Builds a new PDB object (Pdbo) and adds it to the control PDB vector and sets the internal cursor (index) to point to the new PDB object.

```
public void addAsLastControl(String name,String data,char proc)
```

Parameters:
- **name** PDB symbolic name
- **data** PDB data
- **proc** PDB processing flag

addAsLastInput(Pdbo)
Adds an existing PDB object (Pdbo) to the input PDB chain and sets the internal cursor (index) to point to the new PDB object in the chain.

```
public void addAsLastInput(Pdbo PdbO)
```

addAsLastInput(String,String)
Builds a new PDB object (Pdbo) and adds it to the input PDB chain and sets the internal cursor (index) to point to the new PDB object in the chain. Because this method does not call a PDB processing flag, the flag defaults to a blank value.

```
public void addAsLastInput(String name,String data)
```

addAsLastInput(String,String,char)
Builds a new PDB object (Pdbo) and adds it to the input PDB chain and sets the internal cursor (index) to point to the new PDB object in the chain.

```
public void addAsLastInput(String name,String data,char proc)
```

or

```
public void addAsLastInput(String,String)
```

or

```
public void addAsLastInput(Pdbo PdbO)
```

Parameters:

name PDB symbolic name
data PDB data
char PDB processing flag

PdbO PDB object (Pdbo) to add to the list

clear()

Resets the Hicao object to the initial state: Clears all of the Pdbo vectors and invalidates the indices for those vectors.

```
public Hicao clear()
```

Returns:
Hicao Object

copyOutput(Hicao)

Copy all of the output Pdbo objects from the current Hicao object to the passed Hicao object and set the output Pdbo index in the passed object to point to the first output Pdbo.

```
public void copyOutput(Hicao targetHica)
```

Parameters:
targetHica - HICA Object to contain output PDB list

freeAllControl()

Removes all of the objects from the control Pdbo vector and invalidates the index.

```
public Hicao freeAllControl()
```

Returns:
Hicao Object

freeAllInput()

Removes all of the objects from the input Pdbo vector and invalidates the index.

```
public Hicao freeAllInput()
```

Returns:
Hicao Object

getCntrlPdbo(int)

Retrieves a Pdbo object (Pdbo) from the control PDB list.

```
public Pdbo getCntrlPdbo(int index) throws NullPointerException
```

Parameters:
index - Control PDB list index

Returns:
PDB Control Object

Throws:
NullPointerException (Occurs when the control PDB list is empty.)

getErrorPdbo(int)

Retrieves a Pdbo object (Pdbo) from the error PDB list.

```
public Pdbo getErrorPdbo(int index)
throws NullPointerException
```

Parameters:
index - Error PDB list index

Returns:
PDB Error Object

Throws:
NullPointerException (Occurs when the error PDB list is empty.)

getInputPdbo(int)
Retrieves a Pdbo object (Pdbo) from the input PDB list.

```
public Pdbo getInputPdbo(int index)
throws NullPointerException
```

Parameters:
index - Input PDB list index

Returns:
PDB Input Object

Throws:
NullPointerException (Occurs when the input PDB list is empty.)

getMessagePdbo(int)
Retrieves a Pdbo object (Pdbo) from the message PDB list.

```
public Pdbo getMessagePdbo(int index)
throws NullPointerException
```

Parameters:
index - Message PDB list index

Returns:
PDB Message Object

Throws:
NullPointerException (Occurs when the message PDB list is empty.)

getOutputPdbo(int)
Retrieves a Pdbo object (Pdbo) from the output PDB list.

```
public Pdbo getOutputPdbo(int index)
throws NullPointerException
```

Parameters:
index - Output PDB list index

Returns:
PDB Output Object

Throws:
NullPointerException (Occurs when the output PDB list is empty.)

HLAPIReasonCode()
Returns the Tivoli Information Management for z/OS remote HLAPI reason code.

```
public long HLAPIReasonCode()
```

Returns:
Remote API reason code

HLAPIReturnCode()

Returns the Tivoli Information Management for z/OS remote HLAPI return code.

```
public long HLAPIReturnCode()
```

Returns:

Remote API return code

HLAPITrans()

Locates the first control Pdbo element that has a key of TRANSACTION_ID and returns the data associated with that element.

```
public String HLAPITrans()
throws NullPointerException, NameNotFoundException
```

Returns:

PDB object data

Throws:

NullPointerException (Occurs when the control PDB list is empty.)

NameNotFoundException (Occurs when there is no control PDB object found whose name matches the value for the TRANSACTION_ID Pdbo).

isSessionActive()

Return true if this Hicao currently has a session active

```
public boolean isSessionActive()
```

Returns:

True or False

locateFirstControlWithName(String)

Locate the first control PDB object (Pdbo) whose name matches the passed name.

```
public Pdbo locateFirstControlWithName(String name)
throws NullPointerException, NameNotFoundException
```

Parameters:

name--PDB symbolic name

Returns:

Pdbo object

Throws:

NullPointerException (Occurs when the control PDB list is empty.)

NameNotFoundException (Occurs when there is no control PDB object found whose name matches the passed name.)

locateFirstInputWithName(String)

Locate the first input PDB object (Pdbo) whose name matches the passed name.

```
public Pdbo locateFirstInputWithName(String name)
throws NullPointerException, NameNotFoundException
```

Parameters:

name -- PDB symbolic name

Returns:

Pdbo object

Throws:

NullPointerException (Occurs when the input PDB list is empty.)

NameNotFoundException (Occurs when there is no input PDB object found whose name matches the passed name.)

locateFirstOutputWithName(String)

Locate the first output PDB object (Pdbo) whose name matches the passed name.

```
public Pdbo locateFirstOutputWithName(String name)
throws NullPointerException, NameNotFoundException
```

Parameters:

name--PDB symbolic name

Returns:

Pdbo object

Throws:

NullPointerException (Occurs when the output PDB list is empty.)

NameNotFoundException (Occurs when there is no output PDB object found whose name matches the passed name.)

numberOfControlElements()

Returns a count of the number of elements in the control Pdbo vector.

```
public long numberOfControlElements()
```

Returns:

Number of control objects

numberOfErrorElements()

Returns a count of the number of elements in the error Pdbo vector.

```
public long numberOfErrorElements()
```

Returns:

Number of error objects

numberOfInputElements()

Returns a count of the number of elements in the input Pdbo vector.

```
public long numberOfInputElements()
```

Returns:

Number of input objects

numberOfMessageElements()

Returns a count of the number of elements in the message Pdbo vector.

```
public long numberOfMessageElements()
```

Returns:

Number of message objects

numberOfOutputElements()

Returns a count of the number of elements in the output Pdbo vector.

```
public long numberOfOutputElements()
```

Returns:

Number of output objects

remoteAPIReturnCode()

Returns the Tivoli Information Management for z/OS remotes HLAPI language binding return code.

```
public long remoteAPIReturnCode()
```

Returns:
Remote API language binding return code

removeInput()
If index for input PDB vector is valid, remove the element at that position, set the internal input index to the first input Pdbo object; otherwise, throw an exception for the Hicao object.

```
public void removeInput()
throws NullPointerException
```

Throws:
NullPointerException (Occurs when input PDB list is empty or the input PDB index is invalid.)

setcIndex(int)
Set control PDB list index

```
public void setcIndex(int index)
```

Parameters:
index - PDB list index

setControl(String,String,char)
If index for control PDB vector is valid, update the current Pdbo object with the passed values; otherwise, throw an exception for the Hicao object.

```
public Pdbo setControl(String name,String data,char proc)
throws NullPointerException
```

Parameters:

name PDB symbolic name

data PDB data

proc PDB processing flag

Returns:
PDB Control Object

Throws:
NullPointerException (Occurs when control PDB list is empty or the control PDB index is invalid.)

setControlWithName(String,String,char)
Look for a control Pdbo object whose name matches the passed name, and if found, updates the Pdbo object.

```
public Pdbo setControlWithName(String name,String data,char proc)
throws NullPointerException,NameNotFoundException
```

Parameters:

name PDB symbolic name

data PDB data

proc PDB processing flag

Returns:
PDB Control Object

Throws:
NullPointerException (Occurs when the control PDB list is empty.)

NameNotFoundException (Occurs when there is no control PDB object found whose name matches the passed name.)

seteIndex(int)

Set error PDB list index

```
public void seteIndex(int index)
```

Parameters:

index - PDB list index

setiIndex(int)

Set input PDB list index

```
public void setiIndex(int index)
```

Parameters:

index - PDB list index

setInput(String,String,char,char,long)

If index for input PDB vector is valid, update the current Pdbo object with the passed values; otherwise, throw an exception for the Hicao object.

```
public Pdbo setInput(String name,String data,char proc,char type,long datw)
throws NullPointerException
```

Parameters:

name PDB symbolic name

data PDB data

proc PDB processing flag

type PDB data type

datw PDB parameter data width

Returns:

PDB Control Object

Throws:

NullPointerException (Occurs when input PDB list is empty or the input PDB index is invalid.)

setmIndex(int)

Set message PDB list index

```
public void setmIndex(int index)
```

Parameters:

index - PDB list index

setoIndex(int)

Set output PDB list index

```
public void setoIndex(int index)
```

Parameters:

index - PDB list index

submit()

Call the Tivoli Information Management for z/OS remote HLAPI passing the current HICA object (Hicao).

```
public Hicao submit()
```

Returns:
Hicao Object

summary()
Output a summary of the Hicao object to the standard output stream -- the transaction (if there is one) obtained from the control Pdbo vector, and the return and reason codes.

```
public void summary()
throws NullPointerException, NameNotFoundException
```

Throws:
NullPointerException (Occurs when the control PDB list is empty.)

NameNotFoundException (Occurs when there is no control PDB object found whose name matches the passed name.)

Class Pdbo

The public class Pdbo is a class file used to implement the PDB data structure. The Pdbo class contains information that is stored into and retrieved from the Tivoli Information Management for z/OS HLAPI PDB control block by the associated methods.

Pdbo()
Constructor for the Pdbo object. The Pdbo data values are set to a default of either blank or 0, depending on the data type.

```
public Pdbo()
```

Pdbo(String,String,char,char,char,long)
Constructor for the Pdbo object. The Pdbo data is initialized with the passed data values.

```
public Pdbo(String name,String data,char proc,char type,char code,long dataw)
```

Parameters

name symbolic name (PDBNAME)
data parameter data (PDBDATA)
proc processing flag value (PDBPROC)
type data type value (PDBTYPE)
code data error code value (PDBCODE)
dataw data unit width value (PDBDATW)

copyTo(Pdbo)
Assigns values from passed Pdbo object to the current Pdbo object.

```
public void copyTo(Pdbo pdb)
```

Parameters:
pdb - Pdbo object

equalTo(Pdbo)
Determines the equality of Pdb objects based on their symbolic names.

```
public boolean equalTo(Pdbo pdb)
```

Parameters:
pdb - Pdbo object

getAppl()
Retrieves Pdb object (Pdbo) application use field.

```
public int getAppl()
```

Returns:
PDB application use field

getCode()
Retrieves Pdb object (Pdbo) parameter data error code.

```
public byte getCode()
```

Returns:
PDB parameter data error code

getData()
Retrieves Pdb object (Pdbo) parameter data.

```
public String getData()
```

Returns:
PDB parameter data

getDatw()
Retrieves Pdb object (Pdbo) parameter data unit width.

```
public int getDatw()
```

Returns:
PDB parameter data unit width

getFlag()
Retrieves Pdb object (Pdbo) binary flag.

```
public boolean getFlag()
```

Returns:
PDB binary data flag

getName()
Retrieves name of current Pdb object.

```
public String getName()
```

Returns:
PDB symbolic name

getProc()
Retrieves Pdb object (Pdbo) processing flag.

```
public byte getProc()
```

Returns:
PDB processing flag

getType()
Retrieves Pdb object (Pdbo) data type.

```
public byte getType()
```

Returns:
PDB data type

setAppl(long)
Sets Pdb object (Pdbo) application use field.

```
public void setAppl(long appl)
```

Parameters:
appl - PDB application use field

setCode(char)
Sets Pdb object (Pdbo) data code.

```
public void setCode(char code)
```

Parameters:
code - PDB data code

setData(String)
Sets Pdb object (Pdbo) parameter data.

```
public void setData(String data)
```

Parameters:
data - PDB parameter data

setData(String,int)
Sets Pdb object (Pdbo) parameter data.

```
public void setData(String data,int datalen)
```

Parameters:
data PDB parameter data
datalen
Length of parameter data

setDatw(long)
Sets Pdb object (Pdbo) data unit width.

```
public void setDatw(long datw)
```

Parameters:
datw - PDB data unit width

setName(String)
Sets name of current Pdb object (Pdbo).

```
public void setName(String name)
```

Parameters:
name - PDB symbolic name

setProc(char)
Sets Pdb object (Pdbo) processing flag.

```
public void setProc(char proc)
```

Parameters:
proc - PDB processing flag

setType(char)
Sets Pdb object (Pdbo) data type.

```
public void setType(char type)
```

Parameters:
type - PDB data type

Class Blmyjwc

ALIAS_TABLE
public final static String ALIAS_TABLE

APIMSG_OPTION
public final static String APIMSG_OPTION

APPLICATION_ID
public final static String APPLICATION_ID

APPROVAL_STATUS
public final static String APPROVAL_STATUS

ASSOCIATED_DATA
public final static String ASSOCIATED_DATA

BEGINNING_HIT_NUMBER
public final static String BEGINNING_HIT_NUMBER

BYPASS_PANEL_PROCESSING
public final static String BYPASS_PANEL_PROCESSING

CLASS_COUNT
public final static String CLASS_COUNT

DATA_VIEW_NAME
public final static String DATA_VIEW_NAME

DATABASE_ID
public final static String DATABASE_ID

DATABASE_PROFILE
public final static String DATABASE_PROFILE

DATE_FORMAT
public final static String DATA_FORMAT

DEFAULT_DATA_STORAGE_SIZE
public final static String DEFAULT_DATA_STORAGE_SIZE

DEFAULT_OPTION
public final static String DEFAULT_OPTION

DELETE_HISTORY
public final static String DELETE_HISTORY

EQUAL_SIGN_PROCESSING
public final static String EQUAL_SIGN_PROCESSING

HISTORY_DATA
public final static String HISTORY_DATA

HLAPILOG_ID
public final static String HLAPILOG_ID

HLIMSG_OPTION
public final static String HLIMSG_OPTION

INQUIRY_RESULT
public final static String INQUIRY_RESULT

LIST_MODE
public final static String LIST_MODE

MESSAGE_DATA
public final static String MESSAGE_DATA

MULTIPLE_RESPONSE_FORMAT
public final static String MULTIPLE_RESPONSE_FORMAT

NUM_RESERVED_CPDBNAME
public final static long NUM_RESERVED_CPDBNAME

NUMBER_OF_HITS
public final static String NUMBER_OF_HITS

PASSWORD
public final static String PASSWORD

PIDT_NAME
public final static String PIDT_NAME

PRIVILEGE_CLASS
public final static String PRIVILEGE_CLASS

REPLACE_TEXT_DATA
public final static String REPLACE_TEXT_DATA

RETRIEVE_ITEM
public final static String RETRIEVE_ITEM

RETURN_VALIDATION_DATA
public final static String RETURN_VALIDATION_DATA

RNID_SYMBOL
public final static String RNID_SYMBOL

SEARCH_ID
public final static String SEARCH_ID

SEARCH_TYPE
public final static String SEARCH_TYPE

SECURITY_ID
public final static String SECURITY_ID

SEPARATOR_CHARACTER
public final static String SEPARATOR_CHARACTER

SESSION_MEMBER
public final static String SESSION_MEMBER

SPOOL_INTERVAL
public final static String SPOOL_INTERVAL

TABLE_COUNT
public final static String TABLE_COUNT

TEXT_AREA
public final static String TEXT_AREA

TEXT_AUDIT_OPTION
public final static String TEXT_AUDIT_OPTION

TEXT_DDNAME
public final static String TEXT_DDNAME

TEXT_MEDIUM
public final static String TEXT_MEDIUM

TEXT_OPTION
public final static String TEXT_OPTION

TEXT_STREAM
public final static String TEXT_STREAM

TEXT_UNITS
public final static String TEXT_UNITS

TEXT_WIDTH
public final static String TEXT_WIDTH

TIMEOUT_INTERVAL
public final static String TIMEOUT_INTERVAL

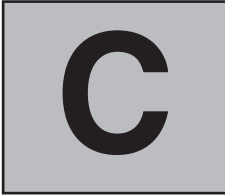
TRANSACTION_ID
public final static String TRANSACTION_ID

TSP_NAME
public final static String TSP_NAME

USE_AS_IS_ARGUMENT
public final static String USE_AS_IS_ARGUMENT

USER_PARAMETER
public final static String USER_PARAMETER

USER_PARAMETER_DATA
public final static String USER_PARAMETER_DATA



HLAPI Service Call Return Codes

Some codes are returned from the HLAPI client and are not returned within the HICA. They are presented as a simple return code in the normal C-language style. Those return codes are listed in this section.

Note: Return and reason codes for the HLAPI, LLAPI, HLAPI remote environment servers, and HLAPI remote clients are contained in the *Tivoli Information Management for z/OS Application Program Interface Guide*. That document also contains the return codes for the HLAPI/REXX, REXX HLAPI/2, REXX HLAPI/AIX, and REXX HLAPI/USS.

Return Codes

IDBRC_NOERR (0)

Explanation: The service call was successful. This does not imply any status about the transaction submitted. It simply means that the service call finished without error.

IDBRC_XERR (1)

Explanation: Extended error. The service call finished, but a nonzero return code exists in the HICARETC field in the specified HICA.

IDBRC_BADHICA (2)

Explanation: The specified HICA is not valid. Three possible reasons for this are:

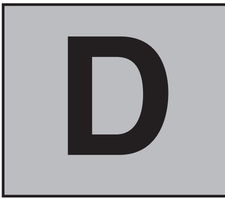
1. The acronym *HICA* does not exist in the HICA field labeled ACRO.
2. A length that is not valid has been given for the HICA.
3. The acronym *PDB* does not exist in one or more of the PDBs associated with the HICA.

IDBRC_BADPARM (3)

Explanation: An incorrect parameter was passed in the service call. For example, a null pointer was passed for a HICA address, or a value that is not valid was specified for StatusMode or SubmitMode.

IDBRC_SYSERROR (4)

Explanation: This error is returned if there is an unknown error that cannot be returned in the HICA.



Relating Publications to Specific Tasks

Your data processing organization can have many different users performing many different tasks. The books in the Tivoli Information Management for z/OS library contain task-oriented scenarios to teach users how to perform the duties specific to their jobs.

The following table describes the typical tasks in a data processing organization and identifies the Tivoli Information Management for z/OS publication that supports those tasks. See "The Tivoli Information Management for z/OS Library" on page 375 for more information about each book.

Typical Tasks

Table 8. Relating Publications to Specific Tasks

If You Are:	And You Do This:	Read This:
Planning to Use Tivoli Information Management for z/OS	Identify the hardware and software requirements of Tivoli Information Management for z/OS. Identify the prerequisite and corequisite products. Plan and implement a test system.	*Tivoli Information Management for z/OS Planning and Installation Guide and Reference*
Installing Tivoli Information Management for z/OS	Install Tivoli Information Management for z/OS. Define and initialize data sets. Create session-parameters members.	*Tivoli Information Management for z/OS Planning and Installation Guide and Reference* *Tivoli Information Management for z/OS Integration Facility Guide*
	Define and create multiple Tivoli Information Management for z/OS BLX-SPs.	*Tivoli Information Management for z/OS Planning and Installation Guide and Reference*
	Define and create APPC transaction programs for clients.	*Tivoli Information Management for z/OS Client Installation and User's Guide*
	Define coupling facility structures for sysplex data sharing.	*Tivoli Information Management for z/OS Planning and Installation Guide and Reference*
Diagnosing problems	Diagnose problems encountered while using Tivoli Information Management for z/OS	*Tivoli Information Management for z/OS Diagnosis Guide*

Table 8. Relating Publications to Specific Tasks (continued)

If You Are:	And You Do This:	Read This:
Administering Tivoli Information Management for z/OS	Manage user profiles and passwords. Define and maintain privilege class records. Define and maintain rules records.	*Tivoli Information Management for z/OS Program Administration Guide and Reference* *Tivoli Information Management for z/OS Integration Facility Guide*
	Define and maintain USERS record. Define and maintain ALIAS record. Implement GUI interface. Define and maintain command aliases and authorizations.	*Tivoli Information Management for z/OS Program Administration Guide and Reference*
	Implement and administer Notification Management. Create user-defined line commands. Define logical database partitioning.	*Tivoli Information Management for z/OS Program Administration Guide and Reference*
	Create or modify GUI workstation applications that can interact with Tivoli Information Management for z/OS. Install the Tivoli Information Management for z/OS Desktop on user workstations.	*Tivoli Information Management for z/OS Desktop User's Guide*
Maintaining Tivoli Information Management for z/OS	Set up access to the data sets. Maintain the databases. Define and maintain privilege class records.	*Tivoli Information Management for z/OS Planning and Installation Guide and Reference* *Tivoli Information Management for z/OS Program Administration Guide and Reference*
	Define and maintain the BLX-SP. Run the utility programs.	*Tivoli Information Management for z/OS Operation and Maintenance Reference*
Programming applications	Use the application program interfaces.	*Tivoli Information Management for z/OS Application Program Interface Guide*
	Use the application program interfaces for Tivoli Information Management for z/OS clients.	*Tivoli Information Management for z/OS Client Installation and User's Guide*
	Create Web applications using or accessing Tivoli Information Management for z/OS data.	*Tivoli Information Management for z/OS World Wide Web Interface Guide*

Table 8. Relating Publications to Specific Tasks (continued)

If You Are:	And You Do This:	Read This:
Customizing Tivoli Information Management for z/OS	Design and implement a Change Management system. Design and implement a Configuration Management system. Design and implement a Problem Management system.	*Tivoli Information Management for z/OS Problem, Change, and Configuration Management*
	Design, create, and test terminal simulator panels or terminal simulator EXECs. Customize panels and panel flow.	*Tivoli Information Management for z/OS Terminal Simulator Guide and Reference* *Tivoli Information Management for z/OS Panel Modification Facility Guide*
	Design, create, and test Tivoli Information Management for z/OS formatted reports.	*Tivoli Information Management for z/OS Data Reporting User's Guide*
	Create a bridge between NetView® and Tivoli Information Management for z/OS applications. Integrate Tivoli Information Management for z/OS with Tivoli distributed products.	*Tivoli Information Management for z/OS Guide to Integrating with Tivoli Applications*
Assisting Users	Create, search, update, and close change, configuration, or problem records. Browse or print Change, Configuration, or Problem Management reports.	*Tivoli Information Management for z/OS Problem, Change, and Configuration Management*
	Use the Tivoli Information Management for z/OS Integration Facility.	*Tivoli Information Management for z/OS Integration Facility Guide*
Using Tivoli Information Management for z/OS	Learn about the Tivoli Information Management for z/OS panel types, record types, and commands. Change a user profile.	*Tivoli Information Management for z/OS User's Guide*
	Learn about Problem, Change, and Configuration Management records.	*Tivoli Information Management for z/OS Problem, Change, and Configuration Management*
	Receive and respond to Tivoli Information Management for z/OS messages.	*Tivoli Information Management for z/OS Messages and Codes*
	Design and create reports.	*Tivoli Information Management for z/OS Data Reporting User's Guide*

Tivoli Information Management for z/OS Courses

Education Offerings

Tivoli Information Management for z/OS classes are available in the United States and in the United Kingdom. For information about classes outside the U.S. and U.K., contact your local IBM representative or visit **http://www.training.ibm.com** on the World Wide Web.

United States

IBM Education classes can help your users and administrators learn how to get the most out of Tivoli Information Management for z/OS. IBM Education classes are offered in many locations in the United States and at your own company location.

For a current schedule of available classes or to enroll, call 1-800-IBM TEACh (1-800-426-8322). On the World Wide Web, visit:

http://www.training.ibm.com

to see the latest course offerings.

United Kingdom

In Europe, the following public courses are held in IBM's central London education centre at the South Bank at regular intervals. On-site courses can also be arranged.

For course schedules and to enroll, call Enrollments Administration on 0345 581329, or send an e-mail note to:

contact_educ_uk@vnet.ibm.com

On the World Wide Web, visit:

http://www.europe.ibm.com/education-uk

to see the latest course offerings.

Where to Find More Information

The Tivoli Information Management for z/OS library is an integral part of Tivoli Information Management for z/OS. The books are written with particular audiences in mind. Each book covers specific tasks.

The Tivoli Information Management for z/OS Library

The publications shipped automatically with each Tivoli Information Management for z/OS Version 7.1 licensed program are:

- *Tivoli Information Management for z/OS Application Program Interface Guide*
- *Tivoli Information Management for z/OS Client Installation and User's Guide* *
- *Tivoli Information Management for z/OS Data Reporting User's Guide* *
- *Tivoli Information Management for z/OS Desktop User's Guide*
- *Tivoli Information Management for z/OS Diagnosis Guide* *
- *Tivoli Information Management for z/OS Guide to Integrating with Tivoli Applications* *
- *Tivoli Information Management for z/OS Integration Facility Guide* *
- *Tivoli Information Management for z/OS Licensed Program Specification*
- *Tivoli Information Management for z/OS Master Index, Glossary, and Bibliography*
- *Tivoli Information Management for z/OS Messages and Codes*
- *Tivoli Information Management for z/OS Operation and Maintenance Reference*
- *Tivoli Information Management for z/OS Panel Modification Facility Guide*
- *Tivoli Information Management for z/OS Planning and Installation Guide and Reference*
- *Tivoli Information Management for z/OS Program Administration Guide and Reference*
- *Tivoli Information Management for z/OS Problem, Change, and Configuration Management**
- *Tivoli Information Management for z/OS Reference Summary*
- *Tivoli Information Management for z/OS Terminal Simulator Guide and Reference*
- *Tivoli Information Management for z/OS User's Guide*
- *Tivoli Information Management for z/OS World Wide Web Interface Guide*

| **Note:** Publications marked with an asterisk (*) are shipped in softcopy format only.

Also included is the Product Kit, which includes the complete online library on CD-ROM.

To order a set of publications, specify order number SBOF-7028-00.

Additional copies of these items are available for a fee.

Publications can be requested from your Tivoli or IBM representative or the branch office serving your location. Or, in the U.S., you can call the IBM Publications order line directly by dialing 1-800-879-2755.

The following descriptions summarize all the books in the Tivoli Information Management for z/OS library.

Tivoli Information Management for z/OS Application Program Interface Guide, SC31-8737-00, explains how to use the low-level API, the high-level API, and the REXX interface to the high-level API. This book is written for application and system programmers who write applications that use these program interfaces.

Tivoli Information Management for z/OS Client Installation and User's Guide, SC31-8738-00, describes and illustrates the setup and use of Tivoli Information Management for z/OS's remote clients. This book shows you how to use Tivoli Information Management for z/OS functions in the AIX, CICS, HP-UX, OS/2, Sun Solaris, Windows NT, and OS/390 UNIX System Services environments. Also included in this book is complete information about using the Tivoli Information Management for z/OS servers.

Tivoli Information Management for z/OS Data Reporting User's Guide, SC31-8739-00, describes various methods available to produce reports using Tivoli Information Management for z/OS data. It describes Tivoli Decision Support for Information Management (a Discovery Guide for Tivoli Decision Support), the Open Database Connectivity (ODBC) Driver for Tivoli Information Management for z/OS, and the Report Format Facility. A description of how to use the Report Format Facility to modify the standard reports provided with Tivoli Information Management for z/OS is provided. The book also illustrates the syntax of report format tables (RFTs) used to define the output from the Tivoli Information Management for z/OS REPORT and PRINT commands. It also includes several examples of modified RFTs.

Tivoli Information Management for z/OS Desktop User's Guide, SC31-8740-00, describes how to install and use the sample application provided with the Tivoli Information Management for z/OS Desktop. The Tivoli Information Management for z/OS Desktop is a Java-based graphical user interface for Tivoli Information Management for z/OS. Information on how to set up data model records to support the interface and instructions on using the Desktop Toolkit to develop your own Desktop application are also provided.

Tivoli Information Management for z/OS Diagnosis Guide, GC31-8741-00, explains how to identify a problem, analyze its symptoms, and resolve it. This book includes tools and information that are helpful in solving problems you might encounter when you use Tivoli Information Management for z/OS.

Tivoli Information Management for z/OS Guide to Integrating with Tivoli Applications, SC31-8744-00, describes the steps to follow to make an automatic connection between NetView and Tivoli Information Management for z/OS applications. It also explains how to customize the application interface which serves as an application enabler for the NetView Bridge and discusses the Tivoli Information Management for z/OS NetView AutoBridge. Information on interfacing Tivoli Information Management for z/OS with other Tivoli management software products or components is provided for Tivoli Enterprise Console, Tivoli Global Enterprise Manager, Tivoli Inventory, Tivoli Problem Management, Tivoli Software Distribution, and Problem Service.

Tivoli Information Management for z/OS Integration Facility Guide, SC31-8745-00, explains the concepts and structure of the Integration Facility. The Integration Facility provides a task-oriented interface to Tivoli Information Management for z/OS that makes the

Tivoli Information Management for z/OS applications easier to use. This book also explains how to use the panels and panel flows in your change and problem management system.

Tivoli Information Management for z/OS Master Index, Glossary, and Bibliography, SC31-8747-00, combines the indexes from each hardcopy book in the Tivoli Information Management for z/OS library for Version 7.1. Also included is a complete glossary and bibliography for the product.

Tivoli Information Management for z/OS Messages and Codes, GC31-8748-00, contains the messages and completion codes issued by the various Tivoli Information Management for z/OS applications. Each entry includes an explanation of the message or code and recommends actions for users and system programmers.

Tivoli Information Management for z/OS Operation and Maintenance Reference, SC31-8749-00, describes and illustrates the BLX-SP commands for use by the operator. It describes the utilities for defining and maintaining data sets required for using the Tivoli Information Management for z/OS licensed program, Version 7.1.

Tivoli Information Management for z/OS Panel Modification Facility Guide, SC31-8750-00, gives detailed instructions for creating and modifying Tivoli Information Management for z/OS panels. It provides detailed checklists for the common panel modification tasks, and it provides reference information useful to those who design and modify panels.

Tivoli Information Management for z/OS Planning and Installation Guide and Reference, GC31-8751-00, describes the tasks required for installing Tivoli Information Management for z/OS. This book provides an overview of the functions and optional features of Tivoli Information Management for z/OS to help you plan for installation. It also describes the tasks necessary to install, migrate, tailor, and start Tivoli Information Management for z/OS.

Tivoli Information Management for z/OS Problem, Change, and Configuration Management, SC31-8752-00, helps you learn how to use Problem, Change, and Configuration Management through a series of training exercises. After you finish the exercises in this book, you should be ready to use other books in the library that apply more directly to the programs you use and the tasks you perform every day.

Tivoli Information Management for z/OS Program Administration Guide and Reference, SC31-8753-00, provides detailed information about Tivoli Information Management for z/OS program administration tasks, such as defining user profiles and privilege classes and enabling the GUI user interface.

Tivoli Information Management for z/OS Reference Summary, SC31-8754-00, is a reference booklet containing Tivoli Information Management for z/OS commands, a list of p-words and s-words, summary information for PMF, and other information you need when you use Tivoli Information Management for z/OS.

Tivoli Information Management for z/OS Terminal Simulator Guide and Reference, SC31-8755-00, explains how to use terminal simulator panels (TSPs) and EXECs (TSXs) that let you simulate an entire interactive session with a Tivoli Information Management for z/OS program. This book gives instructions for designing, building, and testing TSPs and TSXs, followed by information on the different ways you can use TSPs and TSXs.

Tivoli Information Management for z/OS User's Guide, SC31-8756-00, provides a general introduction to Tivoli Information Management for z/OS and databases. This book has a series of step-by-step exercises to show beginning users how to copy, update, print, create, and delete records, and how to search a database. It also contains Tivoli Information Management for z/OS command syntax and descriptions and other reference information.

Tivoli Information Management for z/OS World Wide Web Interface Guide, SC31-8757-00, explains how to install and operate the features available with Tivoli Information Management for z/OS that enable you to access a Tivoli Information Management for z/OS database using a Web browser as a client.

Other related publications include the following:

Tivoli Decision Support: Using the Information Management Guide is an online book (in portable document format) that can be viewed with the Adobe Acrobat Reader. This book is provided with Tivoli Decision Support for Information Management (5697-IMG), which is a product that enables you to use Tivoli Information Management for z/OS data with Tivoli Decision Support. This book describes the views and reports provided with the Information Management Guide.

IBM Redbooks™ published by IBM's International Technical Support Organization are also available. For a list of redbooks related to Tivoli Information Management for z/OS and access to online redbooks, visit Web site **http://www.redbooks.ibm.com** or **http://www.support.tivoli.com**

Index

A

accounting considerations for server 16, 17
adding defined groups to a list 205
allocating 121, 179
  allocating 122, 180
  allocating example 122, 180
allocating HICAs 122, 180
allocating PDBs 122, 180
APPC/MVS
  ASCH address space command 33
  HLAPI/CICS, customization for 191
  starting the environment 32
  stopping the environment 32
  support command 67
  SYS1.PARMLIB members, modifying 28
APPCPMaa parameter
  modification 28
  SCHED 29
  TPDATA 29
  TPLEVEL 22, 29
APPCPMaa parmlib member
  SCHED 29
  TPDATA 29
  TPLEVEL 22, 29
applications, developing for HLAPI/UNIX client 270, 274
ASCHPMaa parmlib member
  CLASSADD statement 29
  modification 29
  OPTIONS statement 29
ASCII DBCS code pages 115
ASCII SBCS code pages 114
asynchronous processing 110, 168

B

BLM2SAM1, setup 126, 184
BLMDBPATH environment variable 311
BLMIREAD 82
BLML parameter 193
BLMMRES 38, 56
BLMSMPATH environment variable 311
BLMYKCOM 197
BLMYKINF 196
BLMYKTRM 198
BRDCST command 19

C

C language binding
  HLAPI/2 use 71
C language binding *(continued)*
  Windows NT use 139
cancel command
  APPC 68
  TCP/IP 68
child processes 270
  IDBMAXCMS 253, 305
CICS
  DFHDCT 191
  DFHPLTSD 191
  DFHSIT 191
  enabling 191
  interface 188
  startup JCL 191
  system definitions data set 191
  termination handler 188
CICS client
  CICS/ESA 187
  CICS interface 188
  communication manager 188
  Termination Handler 188
CICS system definition file (CSD)
  JCL 195
  online customization 196
  system definition data set
    connection entry 201
    partner entry 204
    program entry 199
    sample entries 205
    session entry 202
    transaction entry 199
client
  available HLAPI transactions 2
  comparison 4
  definition 1
  HLAPI/2, overview 3
  HLAPI/CICS, overview 2
  HLAPI/UNIX
    configuration planning 237
    configuring 241
    developing applications 270, 274
    distributing 238, 239, 240
    HLAPI/AIX overview 2, 228
    HLAPI/HP overview 2
    HLAPI/Solaris overview 3
    HLAPI/USS overview 3
    installing 239, 240, 241
    removing options 248
    resources 232
    Windows NT overview 3
code pages supported by HLAPI/2 114
communication link, defining for HLAPI/UNIX 241, 246, 300
communication protocol 2, 228
Communications Manager/2
  define CPI-C side information 80

Index

Communications Manager/2 *(continued)*
define link to MVS 79
define local LU 78
define partner LU 79
setup for HLAPI/2 77
components of HLAPI/CICS 341
configuring HLAPI/UNIX 231, 237, 241
conversation management 13, 16
conversation sharing 96, 154, 253, 305
converting C programs 117, 175
converting HLAPI programs to HLAPI/UNIX 282
create record, HL08 114, 172
customizing
APPC/MVS 191
destination control table (DFHDCT) 193
resource definition online (RDO) 196
shut-down program load table (DFHPLT) 194
startup JCL 195
system definition data set JCL 195
system initialization table (DFHSIT) 192
systems definition data set online 196

D

data conversion 110, 168
data conversion characteristics, HLAPI/UNIX transactions 269
database profile 97, 155, 255, 307
definition 251
example 100, 158, 310
IDBCharCodePage keyword 98
IDBCharCodeSet keyword 156, 255
IDBDataLogLevel keyword 98, 156
IDBDataLogLevelkeyword 256, 307
IDBIdleClientTimeOut keyword 256, 308
IDBLogFileNameActive keyword 98, 156, 256, 308
IDBLogFileNameOld keyword 98, 156, 257, 308
IDBLogFileSize keyword 98, 156, 257, 308
IDBRequesterHost keyword 257, 309
IDBRequesterService keyword 257, 309
IDBServCharCodePage keyword 99
IDBServCharCodeSet keyword 157, 258
IDBServerHost keyword 99, 157, 258, 309
IDBServerService keyword 99, 157, 258, 310
IDBSymDestName keyword 99, 157, 259
database profile example 100, 158, 259, 310
DBCS considerations, HLAPI/UNIX transaction data 269
deleting HLAPI/2
from workstation 92
DFHDCT, customizing 193
DFHPLT 194
DFHSIT, customizing 192
diagnosing HLAPI/UNIX common problems 264
diagnosis of some common HLAPI/2 problems 106
diagnosis of some common HLAPI/NT problems 164
display command
APPC 67
TCP/IP 67

E

EBCDIC MIX code pages 115
EBCDIC Pure DBCS code pages 115
EBCDIC SBCS code pages 115
environment variables (HLAPI/UNIX) 260, 261, 312
environment variables (OS/2) 101
IDBDataLogLevel 101
IDBDBPATH 101
IDBSMPATH 102
profile override 101
profile search path 101
two uses 101
environment variables (Windows NT) 158
IDBDataLogLevel 159
IDBDBPATH 159
IDBSMPATH 160
profile override 159
two uses 158
error logging 103, 161
error probe logging, HLAPI/UNIX 262, 313
example
MRES with APPC configuration 7
MRES with TCP/IP configuration 9
RES, MRES with APPC, and MRES with TCP/IP configuration 10
RES and MRES with APPC configuration 8
RES configuration 6

F

FMID, HOY6108 191
force command
APPC 68
TCP/IP 68
functional components of HLAPI/CICS 188
functions, HLAPI/UNIX 275, 280

H

hardware requirements
HLAPI/2 74, 142
header file 121, 179
header files, HLAPI/UNIX 271
HICA 121, 179
allocating 122, 180
allocating example 122, 180
HICA data structure
allocating 273
initializing 273
overview 188, 272
HL01, initialize Tivoli Information Management for z/OS 111, 170
HL01 transaction 210, 278
HL02, terminate Tivoli Information Management for z/OS 113, 172
HL02 transaction 279, 326

HL06, retrieve record 113, 172
HL06 transaction 279, 326
HL08, create record 114, 172
HL08 transaction 280, 327
HL09, update record 114, 172
HL09 transaction 280, 327
HLAPI/2
  deleting from workstation 92
  installing from CD-ROM 83
  language bindings 69
  parts 69
  requester 71
  server for HLAPI/2 70
  transactions 3, 72, 139
  typical use 69, 137
HLAPI/CICS
  call sequencing 188
  customizing for APPC/MVS 191
  description 187
  functional components 188
  installing 191
  synchronous processing mode 187
  transactions 3
HLAPI for Java
  class Blmyjwc 363
  class Hicao 354
  class Pdbo 361
  overview 353
HLAPI/NT
  allocating HICAs and PDBs 179
  basic transaction flow 139
  binding prototypes 181
  C language binding 139
  client workstation requirements 141
  configuring a communication link to a server 143
  data conversion characteristics 168
  database profile keywords 155
  deleting from a workstation 151
  diagnosing common problems 164
  environment variables 158
  installation and setup summary (sample applications) 176
  installing on a network drive 146
  installing on a workstation from CD-ROM 145
  linking 184
  logging
    error 161
    server 160
    transaction 160
  overview 137
  profile override 159
  profile search path 159
  profile syntax 153
  requester
    starting 163
    stopping 163
  system profile keywords 154
  tips for writing applications 175
  transaction differences 170
  transaction operating modes 167
HLAPI/UNIX
  configuration considerations 231
HLAPI/UNIX *(continued)*
  configuration planning 237
  configuring 241
  distributing 238, 239, 240
  functions 275, 280
  installing 239, 240, 241
  introduction 225
  removing options 248
  setting up HLAPI/AIX 238
  transactions 267
HLAPI/USS 291
  configuration considerations 295
  environment variables 310
  functions 324
  HICA structures 322
  PDB structures 322
  requester overview 292
  sample program 333
  server overview 291
  system profile 304
  transaction names 331
  transactions 319
  using the HLAPI/USS interface 329

I

IDBCHARCODEPAGE database profile keyword 98, 156
IDBCHARCODESET database profile keyword 255
IDBDATALOGLEVEL database profile keyword 98, 156, 256, 307
IDBDATALOGLEVEL environment variable 260, 310
IDBDBPATH environment variable 260
idbech.h header file 272
IDBH.H file
  header file code 127
  including in your program 121, 179
idbh.h header file 271
IDBHLAPI.LIB, linking to 126, 184
IDBIDLECLIENTTIMEOUT database profile keyword 256, 308
IDBINBOUNDBUFSIZE system profile keyword 96, 154, 252, 304
IDBLOGFILENAMEACTIVE database profile keyword 98, 156, 256, 308
IDBLOGFILENAMEOLD database profile keyword 98, 156, 257, 308
IDBLOGFILESIZE database profile keyword 98, 156, 257, 308
IDBMAXCMS system profile keyword 253, 305
IDBOUTBOUNDBUFSIZE system profile keyword 96, 154, 253, 305
IDBREQUESTERHOST database profile keyword 257, 309
IDBREQUESTERHOST environment variable 260, 311
IDBREQUESTERSERVICE database profile keyword 257, 309
IDBREQUESTERSERVICE environment variable 260, 311
IDBSERVCHARCODEPAGE database profile keyword 99, 157
IDBSERVCHARCODESET database profile keyword 258

IDBSERVERHOST database profile keyword 99, 157, 258, 309
IDBSERVERSERVICE database profile keyword 99, 157, 258, 310
IDBSERVICENAME system profile keyword 253, 305
IDBSHARECMS 96, 154, 253, 305
IDBSMPATH environment variable 261
IDBSYMDESTNAME database profile keyword 99, 157, 259
IDBTIMEOUT system profile keyword 254, 306
IDBTransactionStatus() function 124, 182, 276
IDBTransactionSubmit() function 124, 182, 275, 324
initialize Tivoli Information Management for z/OS, HL01 111, 170
installing
- HLAPI/2 component from a LAN to the workstation 85
- HLAPI/2 on the workstation from HLAPI/2 installation CD-ROM 82
- HLAPI/CICS 191
- HLAPI/UNIX 239, 240, 241
- Installation and Maintenance Utility 82

J

Java applications 10
Java wrappers (HLAPI for Java)
- class Blmyjwc 363
- class Hicao 354
- class Pdbo 361
- overview 353

JCL (Job Control Language)
- EXEC statement for RES 28
- startup for CICS 195
- system definition data set for CICS 195

L

list, adding groups 205
load modules
- BLMYKCOM 191
- BLMYKINF 191
- BLMYKTRM 191

logging, error 103, 161
logging, server 102, 160
logging, transaction 102, 160, 261, 312
logical unit (LU)
- considerations for server 16
- defining an MRES to APPC/MVS 48
- defining to VTAM 30
- link definition for Communications Manager/2 79
- local definition for Communications Manager/2 78
- LUADD statement 48
- nonscheduled 48
- partner definition for Communications Manager/2 79

M

MRES with APPC/APPN
- accounting considerations 17
- cataloged procedure considerations 18, 19
- conversation management 14
- LU considerations 16
- overview 6
- parameters 39
- performance considerations 17
- pre-started API sessions considerations 18, 21
- security considerations 23

MRES with TCP/IP
- cataloged procedure considerations 18, 19
- conversation management 15
- LU considerations 16
- overview 8
- parameters 58
- performance considerations 17
- pre-started API sessions considerations 18, 21
- security considerations 24
- socket considerations 16

MVS operator command 67

O

operating modes
- asynchronous processing 110, 168
- synchronous processing 109, 167

operator command, MVS 67
overrides, startup for DFHSIT 192
overview
- HLAPI/2 requester 69

P

PDB data structure
- allocating 274
- control 210
- initializing 274
- overview 188, 272

performance considerations 17
pre-started API sessions 18
processing mode
- HLAPI/CICS, synchronous 187
- HLAPI/UNIX
  - asynchronous 268
  - synchronous 267

profile
- database 251, 255, 259
- system 251, 255

profile override 101, 159
profile search path 101
profile syntax 95, 153
profiles 95, 153
- database 97, 155
- database profile example 100, 158

profiles *(continued)*
syntax 95, 153
system 96, 154, 252, 304
system profile example 97, 155, 255, 307
two types 95, 153
protocol 2, 228

R

RACF 32
README file 82
removing HLAPI/UNIX options 248
requester
definition 1
HLAPI/2, overview 71, 138
HLAPI/UNIX
configuration planning 237
configuring 241
distributing 238, 239, 240
installing 239, 240, 241
overview 227
removing options 248
resources 233
starting 263
stopping 264
system profile 251, 255
requirements
hardware
HLAPI/2 74, 142
software
HLAPI/2 74, 141
HLAPI/CICS 190
HLAPI/Solaris 235
RES (remote environment server)
accounting considerations 16
conversation management 13
HLAPI/2 69
HLAPI/CICS 188
LU considerations 16
overview 5
performance considerations 17
security considerations 22
Windows NT 137
resource definition online (RDO) 196
restoing HLAPI/2 91
from a LAN 92
from the HLAPI/2 installation CD-ROM 91
retrieve record, HL06 113, 172
REXX HLAPI/2 interface
deregistering 133
description 131
installation and setup 131
interface calls 132
prerequisite 132
registering 132
reserved variables 133
REXX HLAPI/2 vs. HLAPI/2 133
REXX HLAPI/2 vs. HLAPI/REXX 133
sample program 134

S

security 21, 24, 32
server
accounting considerations 16, 17
comparison 13
comparison of characteristics 4, 13
definition 1
LU considerations 16
overview 5, 70, 138, 226
performance considerations 17
security considerations 21, 24
socket considerations 16
server logging 102, 160
setting up the sample code 126, 184
shut-down program load table (DFHPLT), customizing 194
side information
for Communications Manager/2 80
for TP profile 27
VSAM data set 46
software requirements
HLAPI/2 74, 141
HLAPI/CICS 190
HLAPI/Solaris 235
starting HLAPI/UNIX requester 263
startup procedure, CICS
startup for DFHSIT, parameter changes 192
stopping HLAPI/UNIX requester 264
synchronous processing 109, 167
syntax, profile 95, 153
SYS1.PARMLIB
APPCPMaa modification 28
ASCHPMaa modification 29
CLASSADD statement 29
modifying for HLAPI/2 28
OPTIONS statement 29
system definition data set
connection entry 201
customizing 195, 196
partner entry 204
program entry 199
sample entries 205
session entry 202
transaction entry 199
system initialization table (DFHSIT), customizing 192
system profile 96, 154, 252, 304
definition 251
example 97
IDBSHARECMS keyword 96, 154, 253, 305
keywords 252, 255
system profile example 97, 155, 255, 307

T

TCP/IP
cataloged procedure considerations 18, 19
conversation management 15
LU considerations 16
overview 8
parameters 58

TCP/IP *(continued)*
performance considerations 17
pre-started API sessions considerations 18, 21
security considerations 24
socket considerations 16
support command 67
terminate Tivoli Information Management for z/OS, HL02 113, 172
tips for writing an application 117, 175
TP parameter
GROUPID 27
TPNAME 27
TPSCHED_EXIT 27
TPSCHED_TYPE 27
TP profile
definition 27
GROUPID parameter 27
TPNAME parameter 27
TPSCHED_EXIT parameter 27
TPSCHED_TYPE parameter 27
transaction differences
DATABASE_PROFILE 111, 170
PASSWORD 111, 170
SECURITY_ID 111, 170
specific PDBs 111, 170
transaction list 135
transaction logging 102, 160, 261, 312
transaction program (TP)
JCL EXEC statement 28
profile 27
RFT data set 28
STEPLID DD statement 28
transaction sequence, definition 229, 293
transactions
HLAPI/2
available transactions 3
basic flow 72, 139
data conversion 110
description 109
HLAPI/2 vs. HLAPI 111
initialize Tivoli Information Management for z/OS, HL01 111
operating modes 109
retrieve record, HL06 113
truncation of mixed data 111
update record, HL09 114
HLAPI/CICS
available transactions 3
synchronous processing mode 187
HLAPI/NT
create record, HL08 172
data conversion 168
description 167
HLAPI/NT vs. HLAPI 170
IDBTransactionStatus 182
IDBTransactionSubmit 182
initialize Tivoli Information Management for z/OS, HL01 170
operating modes 167
retrieve record, HL06 172
truncation of mixed data 169
transactions *(continued)*
HLAPI/NT *(continued)*
update record, HL09 172
HLAPI/UNIX
available transactions 3
basic flow 229
client interface logging 261, 312
concurrency limitations 268
converting HLAPI programs 282, 328
data conversion characteristics 269
database profile 251, 255, 259
developing client applications 270, 274
functions 275, 280
linking applications to runtime services 280
planning applications 281
processing modes 267
server logging 20, 21
special DBCS considerations 269
validating the calling process 267, 319
IDBTransactionStatus 124
IDBTransactionSubmit 124
truncation of mixed data 111, 169

U

update record, HL09 114, 172

W

Windows NT
C language binding 71, 139
requester 138
requester overview 71, 138
server overview 70, 138

File Number: S370/30xx/4300
Program Number: 5697-SD9

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC31-8738-00