IBM Client Input Output/Sockets (CLIO/S)
User's Guide and Reference
Version 2 Release 2.0

Document Number GC28-1676-01

Note!

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

Third Edition (October 1996)

This is a major revision of, and obsoletes, CLIO-user-03.

This edition applies to Version 2 Release 2.0 of the IBM Client Input Output/Sockets (CLIO/S) Licensed Program (5648-129), and to all subsequent releases and modifications until otherwise indicated in new editions. Significant changes or additions to the text and illustrations are indicated by a vertical line (|) to the left of the change.

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

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

International Business Machines Corporation
Department 55JA, Mail Station P384
522 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+914+432-9405
FAX (Other Countries):
Your International Access Code +1+914+432-9405

IBMLink (United States customers only): KGNVMC(MHVRCFS)
IBM Mail Exchange: USIB6TC9 at IBMMAIL
Internet e-mail: mhvrcfs@vnet.ibm.com

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this book
- Page number or topic related to your comment

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

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

Contents

Notices . . . xv
Programming Interfaces . . . xv
Trademarks . . . xv

About This Book . . . xvii
Who Should Use This Book . . . xvii
What's in This Book . . . xvii
The CLIO/S Library . . . xix
Using CLIO/S Man Pages . . . xx
Downloading Files from the Anonymous FTP Server . . . xxi
Where to Find Related Publications . . . xxi
General Information About This Book . . . xxii
Special Terms . . . xxii
Syntax Notation . . . xxii
What's New in This Book . . . xxiii
Summary of Changes for GC28-1676-01 . . . xxiii
Summary of Changes for GC28-1676-00 . . . xxiii

Chapter 1. Introduction . . . 1
An Overview of CLIO/S . . . 1
Major Functions of CLIO/S . . . 2
Fast Transfer . . . 2
The Tape Server . . . 3
How CLIO/S Works . . . 4
The View from 50K Feet . . . 4
Under the Covers of CLIO/S . . . 4

Chapter 2. Migration Considerations . . . 9
Migrating to the Latest CLIO/S Release . . . 9
Installing CLIO/S . . . 9
Migrating CLIO/S Applications . . . 10
Running Different Releases of CLIO/S Simultaneously. . . . 10
Migrating from AIX 3.2.x to AIX 4.1.x . . . 10
Installing CLIO/S on Workstations . . . 10
Using SMIT to Install AIX Version 4 . . . 12
Using CLIO/S in a mixed AIX 3.2.x and AIX 4.1.x Environment . . . 13

Part 1. Configuring and Managing . . . 15

Chapter 3. Configuring and Starting CLIO/S Networks . . . 17
Planning CLIO/S Configuration and Startup . . . 17
Ensuring Complete TCP/IP Installation . . . 18
Configuring Your Channel to Channel Adapters (CTCs) . . . 19
Configuring CTCs On MVS . . . 19
Configuring CTCs On AIX . . . 21
Creating and Customizing CLIO/S Configuration Files . . . 25
The PARCONF File . . . 26
The PARCTC File . . . 32
The NETRC File . . . 34

© Copyright IBM Corp. 1993, 1995, 1996

The CLIO/S Server—PARSRV 34
Selecting a Port Number 35
Setting CLIO/S Environment Variables 36
Customizing the JCL Generation Language 40
Starting the PARSRV Server 40
Stopping the Server 40
Changing Defaults for Some CLIO/S Environment Variables 41
Running Multiple PARSRVs 41
Ensuring Password Security 41
Using the CLIO/S Striping Function in Your Application 42
Enabling CLIO/S for Striping 42
Monitoring Striping Activity 43
Using CLIO/S and Non-CLIO/S Applications on the Same Adapter 43
Sharing with Other CTC Applications 43
Sharing with CLAW Applications 43
Enabling TSO Access to the CLFTP and CLPLINK Commands 44

Chapter 4. Configuring and Managing the CLIO/S Tape Pseudo Device Driver 45
Configuring and Starting the CLIO/S Tape Mount Daemon 45
Monitoring Tape Mount Daemon Activity 48
Stopping the CLIO/S Tape Mount Daemon 49
Changing the CLIO/S Tape Pseudo Device Driver 49
Reconfiguring the Tape Pseudo Device Driver/Restarting the CLIO/S Tape Mount Daemon 50
Restarting the Tape Pseudo Device Driver After a System Reboot 50
Unconfiguring the CLIO/S Tape Pseudo Device Driver 51
Removing the CLIO/S Tape Pseudo Device Driver 52
Administrative Commands 52
cltclntd—Starting the CLIO/S Tape Mount Daemon 52
cltterm—Stopping the CLIO/S Tape Mount Daemon 54

Part 2. The CLIO/S Application Linker 55

Chapter 5. The Application Linker Graphical User Interface 57
Why Use the Application Linker? 57
Learning to Use the Application Linker 58
Terminology and Concepts 58
Applications 59
CLIO/S Pipes 59
CLIO/S Pipe Links 59
Data Objects 59
Configuration 59
The Configuration File 60
Node Hopping 61
System Authorization 61
Password Encryption 61
Object IDs 62
Start After IDs 62
Model Validation 62
About the Application Linker GUI 63
The Primary Window 63
Notebooks 66

Dialog Boxes . . . 66
Message Boxes . . . 66
Using the Graphical User Interface . . . 66
Customizing the Appearance of the GUI . . . 66
Invoking the CLIO/S Application Linker . . . 67
clalink—Starting the CLIO/S Application Linker . . . 67
Sample Configuration Scenarios . . . 67
Simple Application Linker Scenario . . . 68
Complex Application Linker Scenario . . . 69
Changing a Configuration . . . 74
Planning Considerations . . . 74
Changing an Application . . . 74
Changing a CLIO/S pipe . . . 77
Changing a Pipe Link . . . 78
Running Large Application Linker Configurations . . . 79

Part 3. Command Line Interfaces . . . 81

Chapter 6. Accessing CLIO/S Tapes With AIX Commands and Subroutines . . . 83
Allocating and Using Tapes . . . 83
User Commands . . . 84
cltalloc—Allocating a CLIO/S Tape Device for MVS . . . 84
cltalloc—Allocating a CLIO/S Tape Device for AIX . . . 88
cltunall—Unallocating a CLIO/S Tape Device . . . 89

Chapter 7. Transferring Files Using the CLIO/S File Transfer Program . . . 91
How CLIO/S FTP Works . . . 91
If There Are Multiple PARSRVs . . . 91
The CLFTP Command . . . 92
Identifying Yourself to CLFTP . . . 93
Establishing Your Working Directory . . . 94
Customizing CLFTP Defaults . . . 95
Input and Output Files for CLFTP . . . 95
CLFTP Subcommands . . . 95
Displaying Help . . . 97
HELP or ?—Displaying Help . . . 97
Obtaining Status and System Information . . . 98
ECHO—Controlling the Redisplay of Subcommands . . . 98
HASH—Controlling the Display of File Transfer Status Messages . . . 98
LOCSTAT—Retrieving Status Information From the Local Host . . . 99
STATUS—Retrieving Status Information From a Remote Host . . . 100
Working With Directories and Partitioned Data Sets on the Remote Host . . . 101
CD—Changing the Directory on the Remote Host . . . 102
LS—Listing Files in a Remote Directory . . . 104
MKDIR—Creating a Directory on the Remote Host . . . 104
PWD—Displaying the Current Working Directory on the Remote Host . . . 106
Working With Directories and Partitioned Data Sets on the Local Host . . . 107
LCD—Changing the Local Working Directory . . . 107
LLS—Listing Files in a Local Directory . . . 108
LMKDIR—Creating a Directory on the Local Host . . . 109
LPWD—Displaying the Current Working Directory on the Local Host . . . 111
Preparing the Environment . . . 111

ASCII—Changing the Data Transfer Type to ASCII . . . 111
BINARY—Changing the Data Transfer Type to Image . . . 112
ENDONERROR—Controlling Termination on Error . . . 112
LOCSITE—Specifying Site Information to the Local Host . . . 113
SITE—Sending Site Specific Information to a Remote Host . . . 114
SITE and LOCSITE Parameters . . . 115
Record Conversion Support . . . 125
TRANSLATE—Controlling Which Host Performs Data Conversion . . . 126
Transferring Data . . . 127
APPEND—Appending to a Remote File or Data Set . . . 127
DELETE—Deleting Files On the Remote Host . . . 128
GET—Copying Files From the Remote Host . . . 129
PUT—Copying Files to the Remote Host . . . 131
Disconnecting from the Host . . . 132
QUIT—Leaving the CLFTP Environment . . . 132
Starting CLFTP as a Batch Job . . . 133
Transferring Data Across Intermediate Hosts . . . 133
Specifying User and Password Information When Using Intermediate Hosts 134
Bypassing All Prompting . . . 134
Supplying All User IDs and Passwords at the CLFTP Prompt . . . 135
Supplying Some User IDs and Passwords at the CLFTP Prompt . . . 135
CLFTP MVS Return Codes and AIX Exit Values . . . 135
CLFTP Error Messages . . . 136
CLFTP MVS Abends . . . 136
VLIST Sample Files . . . 136
VLIST Sample File 1 . . . 137
VLIST Sample File 2 . . . 137
Resolving Problems With CLIO/S File Transfer Program . . . 139

Chapter 8. The CLIO/S Cross System Pipe Link . . . 141
What Is a Pipe? . . . 141
Transparent Access Across a CLIO/S Network . . . 141
Using the CLIO/S Cross System Pipe Link . . . 143
Copying Files To and From Remote Hosts . . . 144
Accessing Remote Files From an Application . . . 144
Connecting Pipes Between Remote Hosts . . . 145
Creating a CLIO/S Pipe With clpmk . . . 145
clpmk–Creating a CLIO/S Pipe . . . 146
Connecting to a CLIO/S Pipe With CLPLINK . . . 147
Pausing clplink . . . 148
clplink–Connecting To a CLIO/S Named Pipe . . . 148
Customizing CLPLINK Defaults . . . 161

Part 4. Application Programming Interfaces . . . 163

Chapter 9. Using the CLIO/S Sockets Interface . . . 165
Using the CLIO/S Sockets Interface . . . 165
Socket Calls Provided by the CLIO/S Sockets Interface . . . 166
Using Socket Calls Not Provided by the CLIO/S Sockets Interface . . . 167
Portability . . . 167
Using Preexisting Socket Applications with the CLIO/S Sockets Interface 167
Using the CLIO/S Sockets Interface in Other Environments . . . 167
Limitations of the CLIO/S Sockets Interface . . . 168

CLIO/S Sockets Interface Performance Hints 169
Linking and Compiling Sockets 169

Chapter 10. Using the Communication Manager Interface (CMI) 171
CMI Header File fcfcmi.h 171
About CLIO/S Data Conversions 171
Using the CMI Functions 172
clio_accept—Accepting a CLIO/S Connection from Another Process . . . 172
clio_async—Setting the Asynchronous Data Flag 173
clio_check—Checking the Number of Bytes Transmitted 173
clio_close—Closing a CLIO/S Connection 174
clio_connect—Requesting a CLIO/S Connection to Another Process . . . 174
clio_create—Starting a Process on a Node 175
clio_end—Ending a CLIO/S Connection 176
clio_env—Getting the Environment 176
clio_exp—Setting the Environment 177
clio_h2i—Locating Machine Information Using an Internet Address 177
clio_init—Initializing CLIO/S 178
clio_kill—Stopping a Process on a Node 178
clio_l2n or clio_n2l—Converting Data Representations 179
clio_m2i—Locating Machine Information Using a Machine Number 180
clio_map—Mapping from One Nickname Chain to Another 180
clio_n2h—Getting an Internet Address 181
clio_n2i—Locating a Machine by Symbolic Name 181
clio_n2m—Getting the Machine Information ID 181
clio_own—Allowing a Process to Change Ownership of a Connection . . 182
clio_pid—Returning a Process ID 182
clio_read—Reading Data over a CLIO/S Connection 183
clio_write—Writing Data over a CLIO/S Connection 184
tcp_accept—Accepting a TCP/IP Connection 184
tcp_connect—Establishing a TCP/IP Connection 185

Chapter 11. Using the Distributed Processing Interface (DPI) 187
The DPI Header File fcfdpi.h 187
About CLIO/S Data Conversions 187
Overview of the DPI 188
The CLIO/S DPI Master/Servant Model 189
How the CLIO/S DPI Operates 190
The CLIO/S DPI Sample Program 192
Starting the Master 193
Starting the Servant 193
After the Servant is Started 194
Using the DPI Functions 194
ALERT—Setting an Alert Group 194
ALERTGRP or ALRTGRP—Adding a Process ID to an Alert Group 195
cdpi_async or CDPIASY—Setting an Asynchronous Flag By Group 195
cdpi_check or CDPICHK—Checking Number of Bytes Transmitted by a Group 196
cdpi_s2c—Returning a CLIO/S Pointer 196
CLFLUSH—Clearing Buffers Written Using SPANY 197
clio_l2n or LNxxxx—Converting Data to Network Format 198
clio_n2l or NLxxxx—Converting Data to Local Format 199
CLIOBLK—Setting the Buffer Bypass Limit 200
CLIOBUF—Setting the Size of the Internal Buffer 200

CLIODEB—Masking Printed Status Messages . . . 201
CLIOENV—Getting Environment Information . . . 201
CLIOMST—Telling OPNSRV to Start Independent Masters . . . 202
CLIOSOCK—Returning the Socket Descriptor . . . 202
CLIOTBS—Changing Send and Receive Buffer Sizes . . . 203
CLIOTMO—Setting the Time-out Period . . . 203
CLSFAM—Stopping All Processes in a Family . . . 204
CLSPAR—Closing the DPI Environment . . . 204
CLSSRV—Stopping the Servant Process . . . 205
FNDMACH—Locating a Machine by Symbolic Name . . . 205
MSANY or MSxxxx—Transferring Data from Master to Servant . . . 206
MS_flush or MSFLSH—Confirming Data Was Sent from Master to Servant 207
NMACHS—Counting Valid Entries in PARCONF . . . 208
OPNFAM—Opening a Family of Processes . . . 208
OPNPAR—Initializing the DPI Environment . . . 209
OPNSRV—Starting a Servant Process . . . 209
QMACH—Getting Machine Information . . . 210
QSLAVE—Getting Servant Machine Information . . . 211
RPANY or RPxxxx—Receiving Data from a Process . . . 211
rw_dpi—Reading and Writing Data . . . 212
SCHEDUL—Scheduling a Task to Start . . . 213
SETFAM—Setting the Family ID . . . 214
SETSRV—Forcing Data Transfer to the Servant ID . . . 215
SMANY or SMxxxx—Transferring Data from Servant to Master . . . 215
SM_flush or SMFLSH—Confirming Data Was Sent from Servant to Master 216
SPANY or SPxxxx—Sending Data to a Process . . . 217
SYNCHRO—Waiting Until All or Selected Tasks are Completed . . . 218
TSKEXIT—Informing a Master the Servant Process is Finished . . . 219
WAITFOR—Waiting Until an Alert is Available . . . 219
WAITREQ—Indicating a Servant is Waiting for Work . . . 220
Changing the Buffering Mechanism . . . 220
Using CLIO/S ESTAE Support . . . 221
Chained Channel Programs for BMCA and ESCON . . . 222
Returning the Protocol Type . . . 222
Starting a Series of Read or Write Operations . . . 223
Using the rw_dpi Call . . . 223

Chapter 12. Using the CLIO/S Tape Programming Interface and Support 225
The CLIO/S Tape Header File fcftpint.h . . . 225
Sample CLIO/S Tape Program . . . 225
Using the CLIO/S Tape Interface Functions . . . 226
ctpalloc or tpalloc—Starting the Remote Tape Server . . . 227
ctpclose or tpclose—Closing the Tape . . . 230
ctpopen or tpopen—Opening the Tape Device for Use . . . 231
ctpread or tpread—Reading the Tape . . . 232
ctprew or tprew—Rewinding the Tape . . . 233
ctpskpf or tpskpf—Skipping Tape Files . . . 233
ctpskpr or tpskpr—Skipping Tape Records . . . 234
ctpunall or tpunall—Stopping the Remote Tape Server . . . 235
ctpvols or tpvols—Listing the Tape Volumes . . . 235
ctpweof or tpweof—Writing Tape Marks . . . 236
ctpwrite or tpwrite—Writing to Tape . . . 237
ctpwrto or tpwrto—Writing to the Operator . . . 237
CLIO/S Tape Interface Return Values . . . 239

Tape Exit On MVS and AIX 240
MVS Tape Exit FCFTPX00 240
AIX Tape Exit ctpexit 240
Effects of MVS Tape Management on CLIO/S 241
Reading and Writing Multiple Tape Volumes 241
Compiling and Linking CLIO/S Tape Interface Programs 242
On MVS/ESA 242
On the RS/6000 242
Using the FCFTTEST Sample Program 242
Using TPWRTO Support On AIX 246
Using the TP Utility for Logging Messages 246
Specifying a Message Directory with the TAPEMSG Variable 246
Determining Which Node Receives the Message 246
Writing Directly to Devices 247
Special Considerations for ANSI Standard Label (AL) Tapes 247
Preformatting the AL Tape 247
Writing the Data Set Name in the Tape Label Header File 248
Performing Tape I/O 248

Part 5. Appendixes 249

Appendix A. User Specified I/O Exits for CLFTP and CLPLINK 251
IO Parameter 251
The IO Routine 251
IO Parameters for Use with Recovery 252
Recovery with Input Files 252
Recovery with Output Files 253
Notes 254
Discussion of Recovery with Output Files and User I/O Routines 255
Linking the IO Routine on AIX 257
Loading the IO Routine on AIX 257
Linking the IO Routine on MVS 258
Loading the IO Routine on MVS 258

Appendix B. User Specified Data Translation for CLFTP and CLPLINK 259
CONV Parameter 259
The *conv_name* Routine 259
Notes 260
Record Boundaries for the CONV Routine 261
Linking the CONV Routine on AIX 261
Loading the CONV Routine on AIX 261
Linking the CONV Routine on MVS 262
Loading the CONV Routine on MVS 262

Appendix C. Support Programs and Sample JCL 263
CLIO/S Support Programs 263
Cancelling a CLIO/S Job—FCFCAN, FCFCANC 264
Verifying that CLIO/S Can Communicate Between Master and Servant—FCFIDUM 265
Starting the CLPLINK program—FCFFCLPR, FCFFCLPW, clplink 266
Switching Server Debug Flags—FCFPARDE, pardeb 268
Displaying Server Information—FCFPARIN, parinfo 268
Testing Whether PARSRV is Running—FCFPARPI, parping 269

Starting the CLIO/S Server–FCFPARSR, parsrv 269
Stopping a CLIO/S Server—FCFPARST, parstop 271
Testing the CLIO/S Cross System Pipe Link—FCFFPIPR, FCFFPIPW, fcffpipr, fcffpipw 272
Varying On and Off Subchannels—FCFVARY 273
Starting BMCA/PCA Subchannels on the Workstation With the online Program 275

Appendix D. Controlling How CLIO/S Servers Are Started 279
Specifying the Location of the FCFREXX Routine 279
Migrating From CLIO/S Version 1 279
Using REXX/MVS to Generate PARSRV JCL 280
Services Available to FCFREXX 280
Input to FCFREXX 280
Setting the Return Code In FCFREXX 282
Using the SAY and TRACE Commands 282
Under the Covers of FCFREXX 282
Sample FCFREXX Program 282
FCFSHELL 299
Setting the CLIOSHELL Environment Variable 300
Modifying FCFSHELL 300
CLIOARB 300
FCFSHELL Sample Code 301
Modifying FCFSHELL 301

Appendix E. Dynamic Device Allocation on MVS 303
The **FLAGS1** and **FLAGS2** Fields 303
Other Allocation Parameters 303

Appendix F. IOCP, SMIT, TCP/IP, and CLIO/S Cross-reference 309
The sample configuration 309
The sample configuration files and displays 309
Comments on the tagged items 313
Bibliography 315

Index 317

Figures

1. An Example of CLIO/S Communications 6
2. The Sample CLIO/S Configuration Used Throughout This Document . . 18
3. MVS Sample IOCDS Statements 20
4. Available 370 Parallel Channel 21
5. Characteristics of 370 Parallel Channel Adapter 22
6. Subchannel Definitions for ESCON Connections 23
7. Fibre definition for ESCON Connections 24
8. Adding a Remote Tape for CLIO/S 47
9. Sample CLIO/S Application Linker configuration 58
10. The Application Linker GUI primary window 64
11. The Application Linker GUI Tool Bar 65
12. The Application Linker GUI Templates 65
13. The simple Application Linker configuration in icon view 70
14. Downloading a DB2 Database Table onto Two SP Nodes 71
15. The complex Application Linker configuration in icon view 75
16. The complex Application Linker configuration in list view 76
17. JCL for Submitting CLFTP as a Batch Job 133
18. A file that can be used with VLIST 137
19. An example of a VLIST file with site commands 138
20. Piping eliminates intermediate output files. 141
21. An AIX application transparently accessing files on a remote MVS system 142
22. Piping across a CLIO/S network 142
23. Piping across a CLIO/S network 143
24. A CLPLINK instance on each host involved in a transfer. 147
25. CLIO/S Sockets Interface Flow 166
26. How the DPI Works 191
27. fcfsampm, a Sample CLIO/S Master Application. 192
28. fcfsamps, a Sample CLIO/S Servant Application. 192
29. Sample Output Produced by Default CLIO/S DPI ESTAE Handler . . . 222
30. Example of Array of RW_DPI Structures for Chained Reads and Writes (WRCTC=write RDCTC=read) 223
31. Sample Code for Using the rw_dpi Call. 224
32. Example Tape Interface C Code 226
33. Sequence of Calls for Reading and Writing Multiple Tape Volumes . . 242
34. Sample FCFTTEST Subcommands 245
35. JCL for Cancelling a CLIO/S Job (FCFCAN) 265
36. JCL for Sample Master Program (FCFIDUM) 266
37. JCL for Starting CLPLINK to Read From an MVS BatchPipe (FCFFCLPR) 267
38. JCL for Starting CLPLINK to Write to an MVS BatchPipe (FCFFCLPW) 267
39. JCL for Switching PARSRV's Debug Flag (FCFPARDE) 268
40. JCL for Displaying Information about PARSRV (FCFPARIN) 269
41. JCL for Testing If PARSRV is Running (FCFPARPI) 269
42. JCL for Starting the CLIO/S server PARSRV as a started task in an MVS Address Space (FCFPARSR) 270
43. JCL for Starting the CLIO/S server PARSRV in a TSO Address Space (FCFPRTSO) 271
44. JCL for Stopping a DPI Server (FCFPARST) 271

© Copyright IBM Corp. 1993, 1995, 1996

45. Testing the CLIO/S Cross System Pipe Link by reading from an MVS BatchPipe (FCFFPIPR) 272
46. Testing the CLIO/S Cross System Pipe Link by writing to an MVS BatchPipe (FCFFPIPW) 273
47. JCL to Vary On and Off Workstation Subchannels (FCFVARY) 275
48. Sample JCL Generation Program using REXX (FCFREXX) 284
49. Sample FCFSHELL shell script 301
50. Configuration used for this section 309

Tables

1. Configuration Files Used by CLIO/S 25
2. Programs For Starting CLIO/S 35
3. Flags for the CLIODEB Environment Variable 38
4. CLFTP Tasks 96
5. CLFTP Subcommands 96
6. SITE and LOCSITE Parameters for MVS Hosts 115
7. SITE and LOCSITE Parameters for AIX Hosts 121
8. DPI Master and Servant Control Subroutines 188
9. DPI Data Transfer and Conversion Subroutines 189
10. CLIO/S Support Programs 263
11. MVS Device Allocation Parameters and CLIO/S Equivalents 303
12. Miscellaneous Parameters 307

© Copyright IBM Corp. 1993, 1995, 1996

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the user's responsibility.

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:

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

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

IBM Corporation
Mail Station P300
522 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Request

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

Programming Interfaces

This book is intended to help the customer use the CLIO/S product. This publication documents the General-Use Programming Interface and Associated Guidance Information provided by CLIO/S. General-use programming interfaces allow the customer to write programs that request or receive the services of CLIO/S.

Trademarks

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

AIX
BatchPipes/MVS

© Copyright IBM Corp. 1993, 1995, 1996

DB2
Enterprise System/9000
ES/9000
ESCON
IBM
MVS
MVS/ESA
RACF
REXX
RS/6000
RS/6000 Scalable POWERparallel Systems
S/390
Scalable POWERparallel Systems
System/390

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

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

About This Book

This book is a guide and reference manual for using IBM Client Input Output/Sockets (CLIO/S). It describes:

- A high-level overview of CLIO/S
- How to configure and start CLIO/S networks
- Using CLIO/S to run jobs distributed across multiple hosts
- Using the fast command and programming interfaces
- Using the tape server commands and programming interface
- Using the CLIO/S File Transfer Program
- Using the CLIO/S Cross System Pipe Link
- Sample programs, JCL, and AIX commands for starting and managing CLIO/S networks

CLIO/S is a set of programs and application programming interfaces (API) that can be used for high-speed file and data transfer, for distributing work across a client-server environment, and for accessing files and tape drives on remote hosts.

Who Should Use This Book

This book is written for a wide class of CLIO/S users:

- System programmers
- System administrators
- Application developers
- End users

This book assumes the user has some familiarity with the UNIX operating system (environment variables, shells). It also assumes that the user has some familiarity with the MVS operating system (JCL, submitting jobs). The amount of knowledge you need to have depends on which CLIO/S functions you want to use. The chapters describing the CLIO/S Cross System Pipe Link, CLIO/S File Transfer Program, and the command interface to the tape server assume a basic familiarity with UNIX, but no programming knowledge is required. The chapters describing the programming interfaces assume that the user is familiar with C language or FORTRAN programming. It also assumes familiarity with the socket protocol. “Where to Find Related Publications” on page xxi lists related publications describing the socket protocol.

What's in This Book

This book includes the following sections and chapters:

- **Front Matter**

  Contains the Table of Contents, Special Notices, and Preface. Use these to find or interpret information in the book.

- **Chapter 1, Introduction**

  Gives an introduction to CLIO/S. Read this chapter first to determine which aspects of CLIO/S you would like to use.

- **Chapter 2, Migration Considerations**

© Copyright IBM Corp. 1993, 1995, 1996

This chapter helps you migrate from CLIO/S version 1.

- Information about configuring and managing your CLIO/S network:
  - **Chapter 3, Configuring and Starting CLIO/S Networks**

    Describes how you configure and start a CLIO/S network. This chapter explains the CLIO/S configuration files, the SMIT panels, and the MVS configuration steps you use to customize CLIO/S for your installation. A sample CLIO/S configuration is presented that is used for examples here and throughout this document.
  - **Chapter 4, Configuring and Managing the CLIO/S Tape Pseudo Device Driver**

    Describes how to configure the workstation to configure and start the CLIO/S tape pseudo device driver and tape mount daemon.
  - **Chapter 5, The Application Linker Graphical User Interface**

    Describes the CLIO/S Application Linker and how to use it.
- Using CLIO/S's command line interfaces:
  - **Chapter 6, Accessing CLIO/S Tapes With AIX Commands and Subroutines**

    Explains how to access the CLIO/S tape server using standard UNIX commands and programs.
  - **Chapter 7, Transferring Files Using the CLIO/S File Transfer Program**

    Tells you how to use the CLIO/S FTP to transfer files between networked hosts.
  - **Chapter 8, The CLIO/S Cross System Pipe Link**

    Tells you how to use the CLIO/S Cross System Pipe Link to extend the pipe concept to data across a CLIO/S network.
- Using CLIO/S application programming interfaces
  - **Chapter 9, Using the CLIO/S Sockets Interface**

    Provides information about how to use the CLIO/S sockets interface.
  - **Chapter 10, Using the Communication Manager Interface (CMI)**

    Describes how to use the Communications Manager Interface API for fast transfer. The syntax for the CMI calls is also included.
  - **Chapter 11, Using the Distributed Processing Interface (DPI)**

    Describes how to use the Distributed Processing Interface API. Use this information to understand the functions available in the DPI and their syntax.
  - **Chapter 12, Using the CLIO/S Tape Programming Interface and Support**

    Tells you how to use the programming interface to CLIO/S tape support. Use this information when designing and coding your tape server program.
- Additional information
  - **Appendix A, User Specified I/O Exits for CLFTP and CLPLINK**

    Describes user specified I/O operation using the IO site parameter.

– **Appendix B, User Specified Data Translation for CLFTP and CLPLINK**

  Describes a user specifed data translation operation using the CONV site parameter.

– **Appendix C, Support Programs and Sample JCL**

  Describes the support programs that help you manage a CLIO/S network. Sample JCL coding is provided.

– **Appendix D, Controlling How CLIO/S Servers Are Started**

  This Appendix describes:

  - How to use FCFREXX to generate JCL to control new server jobs that are submitted on MVS.
  - How to specify which language each PARSRV server should use.
  - Considerations for migrating from earlier versions of CLIO/S.
  - How to use FCFSHELL to control the way that new AIX server processes are started.

– **Appendix E, Dynamic Device Allocation on MVS**

  Lists the MVS dynamic device allocation parameters supported by CLIO/S and maps the CLIO/S syntax to the corresponding JCL statements.

– **Appendix F, IOCP, SMIT, TCP/IP, and CLIO/S Cross-reference**

  Shows the relationships between various configuration items required to create ESCON connections between MVS and AIX.

The CLIO/S Library

The CLIO/S library of information includes hardcopy documents, listings you can print for yourself, and man pages (online help for AIX users).

The CLIO/S library consists of:

- *CLIO/S Licensed Program Specifications*, GC23-3878

  This publication describes the terms and conditions for the CLIO/S product.

  You can order printed copies of this publication from the IBM Distribution Center.

- *CLIO/S General Information*, GC23-3879

  This publication provides a high level overview of the CLIO/S product, discusses the major features and benefits of CLIO/S, shows examples of the types of CLIO/S networks you can create, and lists the hardware and software requirements for this product.

  This document is available in these ways:

  – You can order printed copies from the IBM Distribution Center.

  – You can download a PostScript listing from the anonymous FTP server **lscftp.pok.ibm.com**. The file name is **/pub/pps/clios/cliogi.ps.Z**. “Downloading Files from the Anonymous FTP Server” on page xxi describes how to connect to the server and download the file.

- *CLIO/S User's Guide and Reference* (GC28-1676)

The document you're reading now is available as a listing you can browse or print for yourself. You can get it one of these ways:

- Listings formatted for 1403, 3820, and PostScript printers are included in the CLIO/S product:

  1403 — A file formatted for browsing or printing on a system printer is located on MVS in **SYS1.FCFUGDE**.

  3820 — A file formatted for AFP printing is located on MVS in **SYS1.FCFU3820**.

  PostScript — A file formatted for PostScript previewing or printing is available on AIX in the file **/usr/lpp/fcf/clio_user_ref.ps**.

- You can download a PostScript listing from the anonymous FTP server **lscftp.pok.ibm.com**. The file name is **/pub/pps/clios/clio_user_ref.ps.Z**. "Downloading Files from the Anonymous FTP Server" on page xxi describes how to connect to the server and download the file.
- A PostScript document discussing performance information is available from the anonymous FTP server. The file name is **/pub/pps/clios/clioperf.ps**.

The CLIO/S Program Directory describes how to install man pages on your workstation, how to print or browse the LIST1403 file, and how to install the PostScript version of this guide on your workstation. Print the LIST3820 file as you would any other file of this type.

You can also view or download this book from the IBM RS/6000 web site at **http://www.rs6000.ibm.com.**.

Using CLIO/S Man Pages

AIX man pages, or online help, are provided for the CLIO/S commands and subroutines. They are installed in the directory **/usr/lpp/fcf/man**. If the person who installs CLIO/S copies them to the system's default man page directory structure, **/usr/man**, the **man** command automatically finds them. If the man pages are not in the **/usr/man** structure, the AIX **man** command provides two ways to specify the manual path. Tell the **man** command which man page directories to search, using one of:

- Specifying the **–M***manpath* flag when you invoke the **man** command.

  For example, you can display the CLIO/S man pages using

  ```
  man -M/usr/lpp/fcf/man topic
  ```

- Setting the MANPATH environment variable to the top of the man page structure. Consult the reference information for the AIX operating system for information about setting MANPATH.

To view a CLIO/S man page, specify by name any of the CLIO/S commands or subroutines when you invoke the **man** command. For example, `man CLIOBUF` displays the man page for the **CLIOBUF** DPI call. Specify the name in the case shown in this manual, for example `man clio_create` and `man CLIOBUF`.

To view the man pages for subroutines more quickly, request manual section 3 by issuing:

```
man 3 CLIOBUF
```

Downloading Files from the Anonymous FTP Server

To copy files from the anonymous FTP server, use **ftp** to connect to **lscftp.pok.ibm.com** using the user ID anonymous. Use your Internet address as the password. Download file **/pub/pps/clios/cliogi.ps.Z** or **/pub/pps/clios/clio_user_ref.ps.Z** in binary mode.

The file is compressed. Uncompress it using the AIX **uncompress** command. The following command uncompresses the PostScript file for the *CLIO/S User's Guide and Reference*:

```
uncompress clio_user_ref.ps.Z
```

Print it as you print other PostScript files.

Where to Find Related Publications

An overview of related programming and hardware publications you may find useful are listed below:

- *AIX Version 3.2 Technical Reference: Base Operating System and Extensions, Volumes 1, 2, and 3*, SC23-2382, SC23-2383, and SC23-2384 and *AIX Version 4.1 Technical Reference: Base Operating System and Extensions, Volumes 1, 2, 3, and 4*, SC23-2614, SC23-2615, SC23-2616, and SC23-2617 and

  These three publications describe the AIX socket calls.

- *AIX Version 3.2 Communication Programming Concepts*, SC23-2206 and *AIX Version 4.1 Communication Programming Concepts*, SC23-2610

  These publications are an overview of AIX socket programming.

- *AIX Version 3.2 Problem Solving Guide and Reference*, SC23-2204, *AIX Version 4.1 Problem Solving Guide and Reference*, SC23-2606, and *AIX Version 4.1 Software Debugging and Reporting for the RISC System/6000*, GG24-2513

  These publications explain how to debug AIX system hardware and software problems.

- *Internetworking with TCP/IP, Volumes 1 and 2*, SC31-6144 and SC31-6145

  These publications describe TCP/IP programming concepts.

- *IBM Block Multiplexer Channel Adapter User's Guide and Service Information*, SC31-8196

  This publication describes the BMCA hardware.

- *RISC/6000 to Mainframe Using S/370 Channel Connections*, SG24-4589.
- *Backup, Recovery and Availability with DB2 PE on RS/6000 SP*, SG24-4695
- *Migrating and Managing Data on RISC/6000 SP with DB2 PE*, SG24-4658
- *AIX Tape Management*, SG24-4705
- *IBM ESCON Channel Adapter User's Guide and Service Information for AIX 4.1*, SC31-8197

  This publication describes the ESCON CTC hardware.

- *TCP/IP Version 3 Release 1 for MVS: Programmer's Reference*, SC31-7135

This publication describes MVS sockets calls and includes an overview of MVS socket programming.

- *IBM TCP/IP for MVS: Customization and Administration Guide Version 3 Release 1*, SC31-7134

  This publication describes TCP/IP on MVS.

- *Application Development Guide: Authorized Assembler Language Programs MVS/ESA System Product: JES2 Version 4, JES3 Version 4,* GC28-1645

  This publication describes how to use MVS dynamic allocation setting.

- *MVS/ESA JCL Reference*, GC28-1654

  This publication describes how to write MVS Job Control Language (JCL).

- *BatchPipes/MVS V1R2 User's Guide and Reference*, GC28-1215

  This publication describes how to use BatchPipes/MVS.

- *TSO Extensions Version 2 REXX/MVS User's Guide*, SC28-1882, and *TSO Extensions Version 2 REXX/MVS Reference*, SC28-1883

  These publications describe the REXX/MVS programming language.

General Information About This Book

The following sections explain terminology, syntax, and other information you need to use this book.

Special Terms

Standard data processing terms are used in this book.

The terms Block Multiplexer Channel Adapter (BMCA) and Parallel Channel Adapter (PCA) are synonymous; this book uses the term BMCA/PCA.

The term workstation includes both RS/6000 workstations and the IBM Scalable POWERparallel Systems.

Syntax Notation

Here is how to interpret the syntax used in this book:

- AIX is case sensitive, so AIX commands and file names should be entered exactly as shown. These are shown in bold face type, for example, /**etc**/**parconf** and **fcfdpi.h**.
- MVS program names are shown in upper case letters in the standard type face. For example, the PARSRV program.
- When the same program or configuration file can be on both MVS and AIX, the name is in upper case letters to refer either to the MVS copy or the generic name. (For example, PARSRV.) When the specific instance refers to the AIX copy, the name is in lower case letters. (For example, **parsrv**.) In most cases, the text will identify the specific location of the program or configuration file.
- MVS file and data set names, such as **SYS1.SFCFSAMP**, are bold.
- Subroutine names and programming language elements should be entered exactly as shown. These are shown in bold face type, for example, **char *CLIOENV(char *_name_)**.

- Parameters for which you substitute an appropriate value, such as *volser*, are in italic type face.
- Machine names, such as ***mvsnode***, are in bold italics.
- Square brackets (**[ ]**) enclose optional items in syntax diagrams.
- The notation <**ctrl-***x*> indicates a control character sequence. For example, <**ctrl-c**> means that you hold down the control key while pressing the character **c**.

What's New in This Book

Summary of Changes for GC28-1676-01

This revision contains changes to support CLIO/S Version 2 Release 2 with PTF numbers UN96987 and UN96988 installed.

This revision also contains technical and editorial improvements.

Changes or additions are indicated by a vertical change bar in the left margin.

Summary of Changes for GC28-1676-00

This edition documents:

- Enhancements to the CLFTP and CLPLINK commands.
- Addition of the CLIO/S Application Linker and the CLALINK command.
- The I/O and CONV site parameters to facilitate the use of user-written routines.

This edition also contains technical and editorial corrections.

Chapter 1. Introduction

This book is a guide and reference manual for using IBM Client Input Output/Sockets (CLIO/S). This chapter provides an introduction to the CLIO/S product, including:

- An overview of CLIO/S
- An explanation of how CLIO/S works
- A description of CLIO/S documentation

An Overview of CLIO/S

IBM Client Input Output/Sockets (CLIO/S) is a set of commands and application programming interfaces (API) that can be used for high-speed communication and for accessing tape devices on a network of AIX workstations and MVS mainframes. CLIO/S makes it easier to distribute work and data across a network of mainframes, workstations, and Scalable POWERparallel Systems. CLIO/S also provides an API to tape drives anywhere in your network. CLIO/S can be used to:

- **Quickly move data between your MVS/ESA system and your workstation (or SP)**. For example, you can store large volumes of seismic data on tape and manage it using a mainframe acting as a data server to multiple workstations. This solution retains tape management as the responsibility of a single mainframe system, while permitting seismic processing capacity to increase by distributing the work.
- **Transfer very large files**. For example, you can use applications on AIX to update customer files during the day, then use CLIO/S for fast backups to take advantage of MVS as a file server with extensive data management capabilities. Using CLIO/S for frequent file copying can mean shorter interruptions to your ongoing applications.
- **Transfer files using familiar workstation commands**. The CLIO/S CLFTP subcommands are similar to those of TCP/IP's FTP command, so there's no need for users to learn a new interface. Users can even access tape data on MVS with the CLFTP subcommands.
- **Access a tape drive on MVS from your workstation as though it were a local tape drive**. For example, you can store data on MVS controlled tape drives and access it using CLIO/S connections to the compute servers.
- **Start servers on other workstations and mainframes in your network to create a parallel processing environment**. For example, CLIO/S can be used to schedule work on several workstations running in parallel. It also provides high data transfer rates and low processor utilization, permitting very high parallel efficiency.
- **Use AIX named pipes and BatchPipes/MVS**. For example, you can access data on MVS (either on DASD or on tape) with an AIX named pipe. Or, an MVS program can use an MVS BatchPipe to send its output to AIX, where another program using an AIX named pipe can do further processing.

© Copyright IBM Corp. 1993, 1995, 1996

Major Functions of CLIO/S

The CLIO/S product provides these major functions:

- Fast transfer of data using:
  - A file transfer command (CLFTP)
  - A pipe link command (CLPLINK)
  - A sockets interface
  - A communications manager programming interface (CMI)
  - A distributed processing programming interface (DPI)

  For more information, see "Fast Transfer."
- The CLIO/S Application Linker to build cross-systems pipe applications. For more information, see Chapter 5, "The Application Linker Graphical User Interface" on page 57.
- A tape server that can be accessed either by the CLIO/S tape programming interface or a pseudo device driver.

For more information, see "The Tape Server" on page 3.

Fast Transfer

CLIO/S provides a user file transfer command (CLFTP), a pipe link command (CLPLINK), and three programming interfaces—the sockets, the communications manager interface (CMI), and the distributed processing interface (DPI). You select the interface that best meets your needs for fast data transfer. One major advantage of CLIO/S fast transfer interfaces is that they can automatically select the fastest technology available in your CLIO/S network. All of the interfaces support the following high-speed communication technologies:

- System/390 (S/390) Channel to Block Multiplexer Channel Adapter (BMCA). BMCA is also known as the Parallel Channel Adapter (PCA)
- S/390 ESCON channel to a RS/6000 ESCON adapter

TCP/IP communications are also supported. If your TCP/IP network is configured to use a high speed attachment such as FDDI or FCS, CLIO/S can provide good performance over TCP/IP.

File Transfer (CLFTP)

CLFTP lets users transfer files between hosts (without programming) using high speed channels. It also supports *routing*, the ability to reach files by going through multiple machines. This is especially valuable in an SP environment, as routing gives you the ability to reach files on different nodes.

Because the CLFTP subcommands are similar to those of TCP/IP's FTP command, there's no need for users to learn a new interface. Users can also access tape data on MVS with the CLFTP subcommands.

For details about using CLIO/S File Transfer Program, see Chapter 7, "Transferring Files Using the CLIO/S File Transfer Program" on page 91.

Cross System Pipe Link (CLPLINK)

CLPLINK lets users and programs pipe data to and from remote locations using AIX named pipes and, optionally, BatchPipes/MVS.

For details about the CLIO/S Cross System Pipe Link, see Chapter 8, “The CLIO/S Cross System Pipe Link” on page 141.

Sockets

The CLIO/S sockets interface provides communication channels that enable unrelated processes to exchange data locally and across networks. The sockets interface runs a combination of TCP/IP and CLIO/S CMI calls and is available as C subroutines.

Chapter 9, “Using the CLIO/S Sockets Interface” on page 165 describes how you use the CLIO/S sockets interface interface.

This interface lets you convert an existing TCP/IP sockets program into a CLIO/S sockets program with little or no change to the application.

Communication Manager Interface (CMI)

The Communication Manager Interface (CMI) supports high-speed communications. It provides functions to start programs on a given node, establish connections between processes, and enable fast data transfer. The CMI is a lower level interface than the sockets, providing more control with more functions to choose from. The CMI is available as C subroutines.

See Chapter 10, “Using the Communication Manager Interface (CMI)” on page 171 for details.

Distributed Processing Interface (DPI)

The Distributed Processing Interface (DPI) provides functions for distributed process control as well as high-speed communications. This programming interface allows programs on different systems to communicate synchronously or asynchronously over high-speed links. This interface provides the most control. The DPI is available in both C and FORTRAN.

See Chapter 11, “Using the Distributed Processing Interface (DPI)” on page 187 for details.

The Tape Server

The CLIO/S tape server provides two ways to access any tape device on a RS/6000, SP, or MVS/ESA system:

- Using standard UNIX commands and programming calls
- Using a special CLIO/S tape programming interface

The CLIO/S tape pseudo device driver lets you use standard UNIX methods to access tapes on remote systems. For example, you can use the UNIX **tar** command to back up files to a tape on an MVS system. See Chapter 6, “Accessing CLIO/S Tapes With AIX Commands and Subroutines” on page 83 for more information.

The CLIO/S tape server lets you access all standard tape operations using a programming interface. The tape server has a read-ahead function and supports

routing to reach a tape drive by going through multiple machines. The tape interface also cooperates with tape management systems that run on MVS. The tape interface is available in both C and FORTRAN. See Chapter 12, “Using the CLIO/S Tape Programming Interface and Support” on page 225 for more information.

How CLIO/S Works

The View from 50K Feet

A typical CLIO/S command or application consists of a *master program* that you start on your local system and one or more *servant programs* that can run anywhere in your distributed CLIO/S network.

When you invoke one of CLIO/S's command line interfaces, the master program is the command: CLFTP, CLPLINK or the tape pseudo device driver. The command does all the work of starting and managing servant programs and of transferring data. The servant programs are included in the CLIO/S product.

Similarly, when you write an application using CLIO/S's tape programming interface, you write only the master program. The tape subroutines call code internal to CLIO/S that starts a servant program on the machine or machines to which your tape drives are physically attached. The servant program, the TAPESRV tape server, is internal to CLIO/S.

When you write a CLIO/S application using the CMI and DPI interfaces, you write both the master and servant programs. Your servant programs can communicate with your master program and with other servants.

The CLIO/S sockets interface is the exception to this rule. It simply speeds up the standard sockets programming interface by using the faster CTC connections instead of TCP/IP.

Under the Covers of CLIO/S

Under the covers of all of the interfaces (except the CLIO/S sockets interface) is the CLIO/S server, PARSRV. When a CLIO/S master wants to start a servant, it sends a request to the PARSRV on the remote host using a control connection over TCP/IP.

When you program using the CMI or DPI interfaces, your master program interacts directly with PARSRV to start servant processes. After the servant is created, you control the interactions between master and servant using CMI and DPI subroutines.

If you are using one of CLIO/S's command line interfaces (CLPLINK, CLFTP, the tape pseudo device driver) or the tape programming interface, subroutines internal to CLIO/S cause PARSRV to start the appropriate servant program on your behalf. For example, when you use the CLIO/S File Transfer Program for high-speed transfer of a file, CLFTP instructs a PARSRV server on the remote host to start CLIO/S's internal file transfer server (FCFFSERV). FCFFSERV communicates across the high-speed connection and transfers your file from the remote host to your local host, as requested in your CLFTP subcommands.

For portability and flexibility, CLIO/S applications do not contain the specific information needed to start communications. This information is contained in the configuration files PARCONF, PARCTC, and NETRC. “Creating and Customizing CLIO/S Configuration Files” on page 25 explains how to create these configuration files for your installation or for your individual program.

Figure 1 on page 6 shows how components of CLIO/S's DPI interact to provide communication and control functions between an MVS/ESA mainframe (called ***mvsnode***) and a workstation (called ***riscnode***) running AIX. In this example, a master application running on AIX starts a servant process on MVS and then exchanges data with that process over a CTC connection.

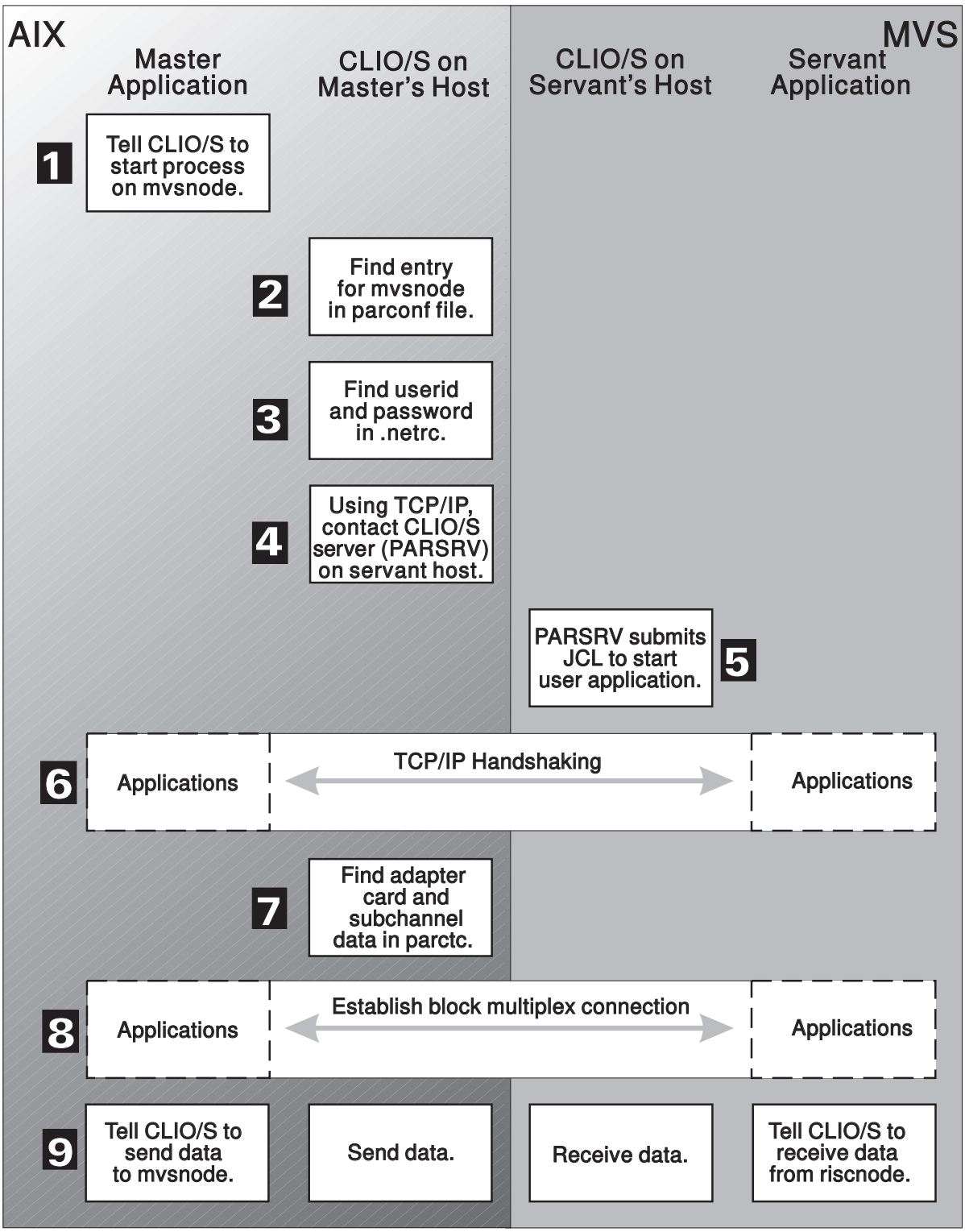


Figure 1. An Example of CLIO/S Communications

The following steps, corresponding to steps in Figure 1, describe how the DPI works:

1. The application on ***riscnode***, the master, calls CLIO/S to start a process on ***mvsnode***.

2. CLIO/S looks in the PARCONF configuration file to find out where ***mvsnode*** is, what type of communication links connect to it, and what operating system is running there.
3. CLIO/S looks in NETRC to determine which user ID to use to start the process.
4. Using a TCP/IP connection, the workstation side of CLIO/S contacts the PARSRV server running on ***mvsnode***, asking it to start the servant application.

The MVS and RS/6000 sides of CLIO/S now cooperate to establish a working connection.

5. PARSRV creates and submits JCL to start the requested application on MVS. See Appendix C, “Support Programs and Sample JCL” on page 263 for more details on JCL generation.
6. The application at the MVS servant and the application at the AIX master establish a TCP/IP connection.
7. CLIO/S looks up adapter and subchannel information specified in PARCTC.
8. The applications establish a fast transfer connection for exchanging data.
9. The two processes can exchange data using generalized communication commands that do not require the processes to be aware of what technology or protocol is being used for the transmission.

Chapter 2. Migration Considerations

Migrating to the Latest CLIO/S Release

This chapter describes the tasks involved in migrating to CLIO/S Version 2 Release 2 from Version 1 or from CLIO/S Version 2 Release 1. Some of these task are performed by systems administrators who install CLIO/S. Others are performed by individuals who maintain CLIO/S applications. If your installation did not use CLIO/S prior to CLIO/S Version 2 Release 2, you do not need to read this chapter.

Installing CLIO/S

This section describes tasks performed by the system administrator who installs the CLIO/S product.

On MVS

Before installing CLIO/S Version 2 Release 2 on your MVS system, you must increase the size of CLIO/S data sets. Refer to the CLIO/S Program Directory for complete instructions.

After installing CLIO/S Version 2 Release 2 on MVS, you must:

- Relink installation-maintained applications as described in “Migrating CLIO/S Applications” on page 10.

  No changes to the source code for your CLIO/S applications are required.

- Migrate any customized JCL generation programs to the new JCL generation language, REXX. See “Migrating From CLIO/S Version 1” on page 279 for more information about the migration process. “Using REXX/MVS to Generate PARSRV JCL” on page 280 describes how to create JCL generation programs using REXX. The pseudo-JCL method of generating JCL is no longer supported.

On AIX

Before installing CLIO/S Version 2 Release 2 on each AIX system, you must:

- Backup the files **/etc/parconf**, **/etc/parctc**, and **/etc/netrc**.

  These files are replaced during the installation of CLIO/S on AIX. Backing them up preserves your installation's customization. After CLIO/S is installed on AIX, restore these files from your backup.

- Remove or rename all existing CLIO/S files, including binaries, Makefiles, header files, and man pages.

  In Version 1, the person who installed CLIO/S selected the name of the CLIO/S directory. In Version 2, the AIX **installp** command places CLIO/S product files and subdirectories in the directory **/usr/lpp/fcf** and **/isr/lpp/clnettape**. Ask the person who installed CLIO/S Version 1 for the name of the directory in which it was installed, and remove those files. If files were copied to other directories (for example, **/usr/man** or **/usr/local/bin**), remove them from those directories. Ask the person who installed Version 1 if CLIO/S's files were copied to other directories.

© Copyright IBM Corp. 1993, 1995, 1996

If you mix files from Version 1 or CLIO/S Version 2 Release 1 (other than **/etc/parconf**, **/etc/parctc**, and **/etc/netrc**) with CLIO/S Version 2 Release 2, the results are unpredictable.

After installing CLIO/S Version 2 Release 2 on the AIX system, you must:

- Restore your saved copies of **/etc/parconf**, **/etc/parctc**, and **/etc/netrc**.
- Relink any installation-maintained CLIO/S applications.

  No changes to the source code for your CLIO/S applications are required.

Migrating CLIO/S Applications

If you have existing CLIO/S applications, you do not need to make changes to the source code. However, minor migration tasks are necessary.

On AIX, modify the Makefiles for your CLIO/S applications to use the correct location for include files and libraries. CLIO/S's files are located in subdirectories of **/usr/lpp/fcf** and **/usr/lpp/fcf/clnettape**. Use the Makefile in **/usr/lpp/fcf/sample** as a guide to the changes you need to make.

On both MVS and AIX, relink existing CLIO/S applications. Binary compatibility with previous versions of CLIO/S is not guaranteed.

Running Different Releases of CLIO/S Simultaneously.

In general, the client and server code for CLIO/S processes must be at the same level. For example, a CLIO/S Version 2 Release 2 CLFTP client must connect to a CLIO/S Version 2 Release 2 CLFTP server on the remote system, and a CLIO/S Version 2 Release 1 CLFTP client must connect to a CLIO/S Version 2 Release 1 CLFTP server on the remote system.

Using multiple PARSRV processes, it is not difficult to set up your system so that both older and newer releases can run on the same system, with each release using a different PARSRV and executing code from different libraries.

Migrating from AIX 3.2.x to AIX 4.1.x

CLIO/S requires different installp images for AIX 3.1 and AIX 4.2 platforms. The following is from the CLIO/S Program Directory.

Installing CLIO/S on Workstations

Some of CLIO/S's function needs to be transferred from MVS to workstations in the network.

Note: There are always two files that need to be transferred. This is a change from previous releases of CLIO/S

If you already have CLIO/S installed, remember to delete the old files in your CLIO/S directory. You may, however, want to preserve the following files by moving them temporarily to a different directory, especially if you have modified them. In most cases, you will want to preserve these files.

Save these files

- **/etc/parconf**
- **/etc/netrc**
- **/etc/parctc**

The **installp** files are now in separate data sets.

If your workstation(s) have AIX Version 3, use data set **SYS1.SFCFAIX3**, and members **FCF325** and **FCFCL325**.

If your workstation(s) have AIX Version 4, use data set **SYS1.SFCFAIX4**, and members **FCF41** and **FCFCL41**.

- **FCF325** – installp file containing source, headers, user's guide, and executables for CLIO/S for RS/6000 systems with AIX 3.2.5
- **FCFCL325** – installp file containing additional source, headers, and executables for CLIO/S for RS/6000 systems with AIX 3.2.5
- **FCF41** – installp file containing source, headers, user's guide, and executables for CLIO/S for RS/6000 systems with AIX Version 4
- **FCFCL41** – installp file containing additional source, headers, and executables for CLIO/S for RS/6000 systems with AIX Version 4

Installing CLIO/S on Workstations with AIX 3.2.5

Follow these directions to install CLIO/S on workstations with AIX 3.2.5 installed.

1. Login to the workstation where you want to install CLIO/S.
2. Decide where you want the installp images placed on the workstation. If necessary, use the **mkdir** command to create a directory. In this program directory, the directory /**usr**/**sys**/**inst.images** is used as an example.

   You will need 25 megabytes of free disk space to download and uncompress this file. After the CLIO/S installation is completed, the files can be erased, freeing up the 25 megabytes of disk space. Use the **ftp** command on your workstation to connect to the MVS system where **SYS1.SFCFAIX3(FCF325)** and **SYS1.SFCFAIX3(FCF325)** reside. Then issue the following commands:

   ```
   binary
   get 'sys1.sfcfaix3(fcf325)' /usr/sys/inst.images/fcf325.Z
   get 'sys1.sfcfaix3(fcfcl325)' /usr/sys/inst.images/fcfcl325.Z
   quit
   ```

   The **fcf325** and **fcfcl325** compressed installp files should now be on the workstation.
3. Uncompress the files:

   ```
   uncompress /usr/sys/inst.images/fcf325.Z
   uncompress /usr/sys/inst.images/fcfcl325.Z
   ```

Updating man Pages

If you installed **man** pages from a previous version of CLIO/S, you must remove them. See the README file in /**usr**/**lpp**/**fcf**/**README** for instructions.

Installing CLIO/S on Workstations with AIX Version 4

Follow these directions to install CLIO/S on workstations with AIX Version 4 installed.

1. Login to the workstation where you want to install CLIO/S.
2. Decide where you want the installp images placed on the workstation. If necessary, use the **mkdir** command to create a directory. In this program directory, the directory /**usr**/**sys**/**inst.images** is used as an example.

   You will need 25 megabytes of free disk space to download and uncompress this file. After the CLIO/S installation is completed, the files can be erased, freeing up the 25 megabytes of disk space. Use the **ftp** command to connect to the MVS system where **SYS1.SFCFAIX4(FCF41)** and **SYS1.SFCFAIX4(FCFCL41)** reside. Then issue the following commands:

   ```
   binary
   get 'sys1.sfcfaix4(fcf41)' /usr/sys/inst.images/fcf41.Z
   get 'sys1.sfcfaix4(fcfcl41)' /usr/sys/inst.images/fcfcl41.Z
   quit
   ```

   The **fcf41** and **fcfcl41** compressed installp files should now be on the workstation.
3. Uncompress the file:

   ```
   uncompress /usr/sys/inst.images/fcf41.Z
   uncompress /usr/sys/inst.images/fcfcl41.Z
   ```

Using SMIT to Install AIX Version 4

SMIT can be used to install the workstation code for AIX Version 4. Once the code has been downloaded to the workstation using **FTP**, the following commands can be used. In this example, it was assumed that the code was placed in /usr/sys/inst.images and uncompressed.

- su root
- <password>
- smit
- On the next panel, select Software Installation and Maintenance
- On the next panel, select Install / Update Software
- On the next panel, select Install / Update Selectable Software
- On the next panel, select Install Software Products at Latest Level
- On the next panel, select Install New Software Products at Latest Level
- On the next panel, enter your chosen path for software location
  In this example, enter /usr/sys/inst.images and in the Software to Install field enter:
  - fcf for the basic fast transfer and tape installation
  - clnettape for the remainder of the fast transfer and tape function

Using CLIO/S in a mixed AIX 3.2.x and AIX 4.1.x Environment

The application linker interface is available only on AIX 4.1 systems. However, you can use it from an AIX 4.1 system to start and control pipe link processes on AIX 3.2 systems.

The other CLIO/S interfaces work on either AIX 3.2 or 4.1 systems, and it does not matter what AIX level the server processes run on.

Part 1. Configuring and Managing

Chapter 3. Configuring and Starting CLIO/S Networks 17

Chapter 4. Configuring and Managing the CLIO/S Tape Pseudo Device Driver 45

© Copyright IBM Corp. 1993, 1995, 1996

Chapter 3. Configuring and Starting CLIO/S Networks

Note

Before using the procedures in this chapter, you must have installed CLIO/S on MVS, downloaded its files to the AIX workstation, and installed the CLIO/S fast transfer feature on AIX. See the CLIO/S Program Directory for instructions for installing CLIO/S on MVS, downloading files to the workstation, installing CLIO/S on AIX, and running the Installation Verification Procedure (IVP).

Planning CLIO/S Configuration and Startup

You can divide the CLIO/S configuration and startup tasks into two main parts:

- Tasks done once for an installation, typically by an administrator. These tasks include:
  - "Ensuring Complete TCP/IP Installation" on page 18
  - "Configuring Your Channel to Channel Adapters (CTCs)" on page 19
  - "Creating and Customizing CLIO/S Configuration Files" on page 25
  - "The CLIO/S Server—PARSRV" on page 34
  - "Changing Defaults for Some CLIO/S Environment Variables" on page 41
  - "Running Multiple PARSRVs" on page 41
  - "Ensuring Password Security" on page 41
  - "Using the CLIO/S Striping Function in Your Application" on page 42
  - "Using CLIO/S and Non-CLIO/S Applications on the Same Adapter" on page 43
  - "Enabling TSO Access to the CLFTP and CLPLINK Commands" on page 44
- Tasks done for all of your applications or configuration files read at the start of an individual application, typically done by a programmer. These tasks include:
  - Creating personal copies of the configuration files, if needed. This is described in "Creating and Customizing CLIO/S Configuration Files" on page 25.
  - "Setting CLIO/S Environment Variables" on page 36

Some tasks, done initially for an entire installation, can be overridden later by an individual user for a specific application.

If your installation wants to make the CLIO/S tape pseudo device driver available, so that workstation users can access tapes using UNIX commands and programming calls, someone must configure the CLIO/S tape pseudo device driver and must perform management tasks related to the tape mount daemon. These optional tasks are described in Chapter 4, "Configuring and Managing the CLIO/S Tape Pseudo Device Driver" on page 45.

The following example of a CLIO/S network is used in the explanations of the configuration files and in examples throughout this document.

© Copyright IBM Corp. 1993, 1995, 1996

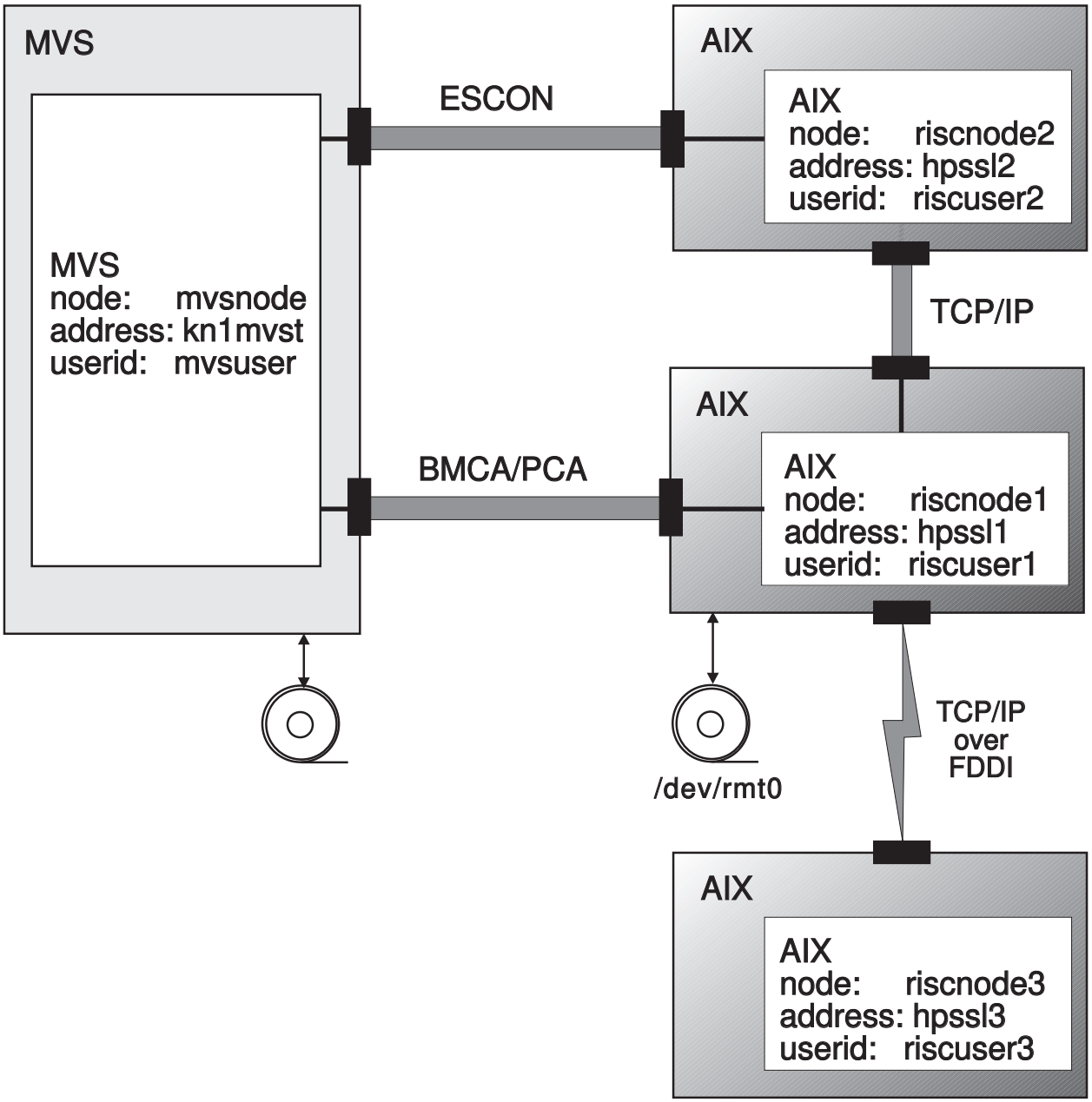


Figure 2. The Sample CLIO/S Configuration Used Throughout This Document

Ensuring Complete TCP/IP Installation

CLIO/S performs a standard set of socket operations to establish initial connections. Your TCP/IP installation must support these standard socket calls in order for CLIO/S to work effectively. These include, but are not restricted to, **socket()**, **gethostname()**, and **gethostbyname()**.

In addition to the calls returning successfully, the returned data must be correct, enabling CLIO/S to establish the required socket connections. It is possible to install TCP/IP such that TCP/IP functions such as FTP work correctly without supporting the socket calls needed by CLIO/S. TCP/IP documentation, listed in

"Where to Find Related Publications" on page xxi, describes how you configure TCP/IP for socket support.

You can override the local host name returned by **gethostname()** using the CLIONODE environment variable. Use this when you have more than one instance of TCP/IP installed on a single host. See "Setting CLIO/S Environment Variables" on page 36 for more information about CLIO/S environment variables.

Configuring Your Channel to Channel Adapters (CTCs)

Configuring CTCs On MVS

CLIO/S uses CTC connections. The tasks that configure your CTC connections on MVS are:

- "Modifying the IOCP"
- "Adding a Missing Interrupt Handler Entry" on page 20

Modifying the IOCP

CLIO/S requires a CTC for fast transfer. If one is not generated already, an IOCP generation or HCD and an MVS CP generation are required. Figure 3 on page 20 shows the definitions required to add CTC addresses 640 to 67F to CHPID 1E for four RS/6000s with 16 subchannels each (BMCA) and to add CTC addresses 6C0-6CF to CHPID 26 (ESCON).

Figure 3 also shows the MVS unitnames you must specify. The esoteric names are required for the CLIO/S DPI, and require an MVS generation only. The esoteric names must be the letters "RS6K" suffixed with a number between 0-99 (for example, `RS6K0`, `RS6K1`, and `RS6K42`). Do not create esoteric names with leading zeros (for example, `RS6K1` is correct; `RS6K01` is not valid). The *adapterid* number specified in **$HOME/parctc** or **/etc/parctc** at the RS/6000 must match the esoteric name.

IBM recommends that you use the same subchannel addresses in both the IODEVICE statement and the UNITNAME statement.

Multipath definitions are not supported. If HCD is used, the unit field should be RS6K.

If you are not using the Hardware Configuration Definition (HCD) program, a "power on reset" and MVS IPL are required after an IOCP generation. An IPL is required after an MVS IOCP generation for MVS releases prior to MVS/SP V4.2.

```
**************************************************
*           PCA CHPID                            *
**************************************************
         CHPID     PATH=((1E)),TYPE=BL
**************************************************
*           ESCON CHPID                          *
**************************************************
         CHPID     PATH=((26)),TYPE=CNC
**************************************************
*           PCA   CNTLUNIT/IODEVICE              *
**************************************************
 CNTLUNIT  CUNUMBR=640,PATH=(1E),SHARED=N,UNIT=3088,PROTOCL=S4,       X
              UNITADD=((40,64))               3088 CTC
 IODEVICE  UNIT=RS6K,ADDRESS=(640,16),CUNUMBR=640,FEATURE=370
 IODEVICE  UNIT=RS6K,ADDRESS=(650,16),CUNUMBR=650,FEATURE=370
 IODEVICE  UNIT=RS6K,ADDRESS=(660,16),CUNUMBR=660,FEATURE=370
 IODEVICE  UNIT=RS6K,ADDRESS=(670,16),CUNUMBR=670,FEATURE=370
*
**************************************************
*           ESCON CNTLUNIT/IODEVICE              *
**************************************************
 CNTLUNIT  CUNUMBR=6C0,PATH=(26),UNIT=3088,                           X
              UNITADD=((C0,64))               RS6K CTC
 IODEVICE  UNIT=RS6K,ADDRESS=(6C0,16),CUNUMBR=6C0,FEATURE=370
*
**************************************************
*            UNITNAMES FOR PCA                   *
**************************************************
*            RISC - UNITNAME                     *
**************************************************
 UNITNAME NAME=RS6K0,UNIT=((640,16))
 UNITNAME NAME=RS6K9,UNIT=((650,16))
 UNITNAME NAME=RS6K20,UNIT=((660,16))
 UNITNAME NAME=RS6K30,UNIT=((670,16))
*
**************************************************
*            UNITNAMES FOR ESCON                 *
**************************************************
*            RISC - UNITNAME                     *
**************************************************
 UNITNAME NAME=RS6K5,UNIT=((6C0,16))
**************************************************
```

Figure 3. MVS Sample IOCDS Statements

Adding a Missing Interrupt Handler Entry

Add a Missing Interrupt Handler (MIH) entry in **SYS1.PARMLIB(IECIOS00)** for each device represented by a BMCA/PCA or ESCON adapter, to disable the missing interrupt handler. This prevents READ from timing out on the channel. Entries for BMCA/PCA devices 640 through 67F and ESCON devices 6C0 through 6CF follows:

```
MIH TIME=00:00,DEV=(640-67F)
MIH TIME=00:00,DEV=(6C0-6CF)
```

Configuring CTCs On AIX

You need to configure AIX with the information from your MVS system generation. This section describes the tasks that configure the CTC hardware on your AIX systems:

- "Configuring and Starting BMCA/PCA Subchannels"
- "Configuring ESCON Subchannels" on page 23

The examples in this section show the SMIT configuration panels you use to configure BMCA/PCA and ESCON channels. These examples show SMIT panels from AIX 3.2.5. You can install if your operating system is at a different level—the panels you see may be slightly different.

Configuring and Starting BMCA/PCA Subchannels

This section describes configuring BMCA/PCA. To configure ESCON, refer to "Configuring ESCON Subchannels" on page 23.

Step 1 - Check Availability of a BMCA/PCA: Logged in with root authority, type **smit**.

Make the following selections:

1. On the Systems Management frame, select 'Devices'
2. On the Devices frame, select 'Communications'
3. On the Communication frame, select '370 Parallel Channel'

   Note: You can skip directly to this menu using the command **smit cat**.
4. On the 370 Parallel Channel frame, select 'Adapter'
5. On the Adapter frame, select 'Change/Show Characteristics of a 370 Parallel Channel Adapter'

   This menu is displayed:

```
                    370 Parallel Channel

Move cursor to desired item and press Enter.

cat0 Available 00-02 Parallel Channel Adapter
```

Figure 4. Available 370 Parallel Channel

Note: The BMCA/PCA **must** be "Available". If "Available" is not shown, check your installation before proceeding.

Press Enter to continue.

Step 2 - Configure the BMCA/PCA Subchannels: The following shows how to configure the single PCA adapter at address 640. Repeat this step for each PCA adapter in your IOCP. The following panel is displayed:

```
Change/Show Characteristics of a 370 Parallel Channel Adapter

                                          Entry Fields
  370 Parallel Channel                      cat0
  Description                               Parallel Channel Adapter
  Status                                    Available
  Location                                  00-02
  Receive Data Transfer OFFSET              0
  Channel SPEED                             4
  NUMBER of transmit buffers                26
  NUMBER of receive buffers                 26
  Transmit buffer SIZE                      4096
  Receive buffer SIZE                       4096
  Starting valid subchannel                 0x60
  Number of valid subchannels               16
  CLAW mode
    Host Name
    Adapter Name
    Subchannel set
  Online/Offline SWITCH                     online
  Online/Offline INDICATOR                  offline
  Apply change to database only             no
```

Figure 5. Characteristics of 370 Parallel Channel Adapter

Change the following fields on this panel:

1. Update the Channel SPEED to reflect what is defined in your MVS IOCP. Figure 3 on page 20 shows Protocol=S4. This corresponds to a "Channel SPEED" of 4.
2. Update the Starting valid subchannel to reflect the MVS starting address. This example configures the BMCA adapter from Figure 3 whose esoteric name is RS6K20. The starting address is 660. Specify the low-order two digits of the starting address (60) as a hex address, '0x60', for "Starting valid subchannel".
3. Update the Number of valid subchannels to reflect the number of subchannels defined in Figure 3 on page 20. In that example MVS defined 16 addresses for the CTC connection to this RS/6000.

Note: You do not have to change or set the CLAW mode fields.

For more information see the *IBM Block Multiplexer Channel Adapter User's Guide and Service Information* manual, SC31-8196.

Step 3 - Starting the BMCA/PCA Subchannels: After the AIX configuration is complete, start the RISC BMCA subchannels using either the FCFVARY or **online** programs. For more information about the these programs see "Varying On and Off Subchannels—FCFVARY" on page 273 and "Starting BMCA/PCA Subchannels on the Workstation With the online Program" on page 275.

ESCON subchannels do not need to be explicitly started up on AIX. Varying them on from the MVS side is sufficient.

Configuring ESCON Subchannels

When using CLIO/S with ESCON and multiple subchannel groups on the RS/6000, CLIO/S uses only one subchannel in the group. Multiple subchannel groups should not be used for CLIO/S.

To use ESCON with CLIO/S, use SMIT to setup and configure your subchannels.

Step 1 - Add Subchannels: Logged in with root authority, type **smit**.

Make the following selections:

1. On the Systems Management frame, select 'Devices'
2. On the Devices frame, select 'Communications'
3. On the Communication Devices frame, select 'ESCON Channel Adapter'

   Note: You can skip directly to this menu using the command **smit esca**.
4. On the ESCON Channel Adapter frame, select 'Subchannel Definitions'
5. On the Subchannel Definitions frame, select 'Add a Subchannel'

   This menu is displayed:

```
                    Add a Subchannel

Type or select values in entry fields.
Press Enter After making all desired changes.

* Subchannel Set Name                    EC1
* Address (local)                        0C0
  Description (max 30 characters)
* Address (remote)                       0C0
  Number of addresses in the group       1
  Path through the ESCD                  010
  Perform a device-end on startup?       yes
  Local Name (Workstations)
  Remote Name (390 Host)
  Type of emulation:                     3088
```

Figure 6. Subchannel Definitions for ESCON Connections

6. Change the following fields on this panel:
   a. Update the Subchannel Set name to reflect the name of the subchannel set. The name used in this example is EC1.
   b. Set the address (local) and address (remote) fields. The local address must be unique per adapter, but is otherwise an arbitrary two-digit number. CLIO/S expects the local address to be the last two characters of the corresponding MVS subchannel, but this can be overridden. See "The PARCTC File" on page 32 for more details.

      The local address of an ESCON adapter must be unique on each RS/6000; the remote address need not be unique. The remote address is used for address decoding of the System/390 host address. It corresponds to the "unit address" in the host IOCP or HCD. The local address is used by the

RS/6000 as a way of connecting the application to the particular subchannel definition.

For example, a single ESCON adapter in the RS/6000 could be dynamically connected to two hosts. Each host may define the RISC as unit address `x'00'`. In this case, both of the remote addresses would be 00, however, the local addresses must be unique. This allows the applications to talk to the correct definition, even though they both have the same unit address.

c. Update the number of addresses in the group. Put the number **1** here. CLIO/S uses only **one** address in a group.

d. Press Enter or select 'Do' to add the subchannel.

Note: If you define more than one ESCON address on MVS, you need to configure AIX to have as many subchannels (1 per group) as you defined in MVS. To add another subchannel, repeat "Step 1 - Add Subchannels" on page 23.

Step 2 - Define the Channel: To define the fibre for the subchannels do the following:

Logged in with root authority, type **smit**.

Make the following selections:

1. On the Systems Management frame, select 'Devices'
2. On the Devices frame, select 'Communications'
3. On the Communications frame, select 'ESCON Channel Adapter'

   Note: You can skip directly to this menu using the command **smit esca**.
4. On the ESCON Channel Adapter frame, select 'Fibre Definitions'
5. On the Fibre Definitions frame, select 'Add a Fibre'

   This menu is displayed: This menu allows you to define the physical connection between the workstation and the mainframe.

```
                    Add a Fibre Definition

Type or select values in entry fields.
Press Enter After making all desired changes.

* Fibre label                              cable1
  Description (max 30 characters)
  Location (slot number)                   6
* Subchannel-Set Name List                 EC1
```

Figure 7. Fibre definition for ESCON Connections

6. Change the following fields on this panel:

   a. Update Fibre label to reflect the name of the cable. This example uses cable1, set during the hardware installation.

b. Update the Location (slot number) to reflect the RS/6000 slot number where the CTC is installed.

c. Update the Subchannel-Set Name List to reflect the names that were defined for the ESCON connection. In Figure 6 on page 23 the name used in this example is EC1, set during your hardware installation.

d. Press Enter or select 'Do' to add the fibre definition.

For more information see the *IBM ESCON Channel Adapter User's Guide and Service Information for AIX 4.1*, SC31-8197.

Step 3 - Creating mbufs for ESCON Subchannels: Sufficient mbufs and clusters must be available for the ESCON adapter card to operate properly. See *AIX Performance Monitoring and Tuning Guide*, SC23-2365, for discussion of mbufs and clusters.

Use the **no** –**a** command to examine the values of several important mbuf-related settings. The output of this command will vary with different operating system levels. Among other data, the results may contain something like the following:

```
     lowclust = 40
      lowmbuf = 80
      thewall = 3584
mb_cl_hiwat = 250
```

If the values of these variables are much lower than those shown, I/O problems may occur. To change the settings, use the **no** –**o** command, as in:

```
no -o thewall=16000
```

If the ESCON adapter is used heavily, you may need to assign even more mbufs. If the output of the **netstat** –**m** command consistently shows "requests for memory denied," you may need to increase *thewall*.

Creating and Customizing CLIO/S Configuration Files

CLIO/S configuration files are used so that applications using CLIO/S functions do not have to know the details of the network connections, hardware technologies, or software protocols being used to support the high-speed data transfer CLIO/S provides.

CLIO/S uses the configuration files listed in the following table.

Table 1. Configuration Files Used by CLIO/S

File Name	Created by	Resides on	Purpose	Page	Notes
PARCONF	Administrator, User	MVS and AIX	Used to establish connection	26	Each user needs access
PARCTC	Administrator, User	Each workstation using ESCON or BMCA/PCA	Used when creating a CTC connection	32	Not needed on MVS
NETRC	User	MVS and AIX	Making a connection	34	Used by other services, also.

The PARCONF File

The PARCONF file allows you to use nicknames for the nodes in your CLIO/S configuration. It maps nicknames to TCP/IP names and contains additional information about the type of machine that the nodes are and the connections that are available to them.

In many cases, the PARCONF file is optional. You can get along without it if you are willing to use TCP/IP addresses instead of nicknames. However, when using the tape pseudo device driver through an intermediate node, a PARCONF entry is required. On AIX, CLIO/S looks for **$HOME**/**parconf** first, then for /**etc**/**parconf**. /**etc**/**parconf** is recommended because all users can share the same PARCONF file. On MVS it is in *user_ID*.**PARCONF**, or it can be supplied with DD statements in the JCL for starting the application. Figure 36 on page 266 demonstrates how to supply PARCONF information inline in your JCL. CLIO/S tries to locate the DD PARCONF statements first. If that is unsuccessful then PARCONF is used. A sample PARCONF file is included in **SYS1.SFCFSAMP(FCFPARCO)** for MVS, and in /**usr**/**lpp**/**fcf**/**sample** for AIX. It must be modified to match your environment.

The PARCONF file consists of a series of one-line entries, each defining a unique nickname. Include at least one entry for each remote host you want to access. Each system's own entry is optional. You can have multiple entries for any remote host, providing different nicknames for different CLIO/S protocols. Hosts that are intermediate nodes for the tape server or tape pseudo device driver should have an entry defining the route. You do not need routing nicknames for intermediate nodes for CLFTP or CLPLINK. The syntax of each entry is:

```
nickname type protocol nodeaddress back=addr extra
```

Where:

nickname Is an arbitrary name chosen for the node. The name must be unique within the PARCONF file.

To define a nickname for use by the tape server or tape pseudo device driver as an intermediate host, specify it as *intermediate_nickname*[**:***intermediate_nickname*]**:***remote_nickname*. This is a colon-separated list of nicknames defined in this PARCONF file, starting with a nickname for the first intermediate host downstream, including nicknames for other intermediate hosts, and ending with the nickname for the remote host.

The PARCONF file on any intermediate host must contain an entry defining the next host downstream of it, using the same nickname that is used in the local host's PARCONF file.

type Represents the processor type:

1 RS/6000
2 MVS

When routing data through intermediate nodes for the tape server and tape pseudo device driver, specify the machine type of the final remote node, not the intermediate node or nodes.

protocol Represents the communications method:

0 Dynamic, CLIO/S selects the fastest protocol
1 TCP/IP

3 Block Multiplexer/Parallel Channel Adapter
4 ESCON

When routing data through intermediate nodes for the tape server and tape pseudo device driver, specify the protocol of the final remote node, not the intermediate node or nodes.

nodeaddress Is the internet address (long or short text, or dotted decimal). If you use the dotted decimal IP address, you can prefix it with an at sign (@) to indicate that CLIO/S should not use the name server to resolve the IP address. For example,

```
risc1   1  2  @150.20.30.5
mvs2    2  4  @150.20.30.11
```

When routing data through intermediate nodes for the tape server and tape pseudo device driver, specify only the internet address of the remote node, not the intermediate node or nodes.

back= Specifies that when a client is starting a server at a certain TCP/IP address, the server should connect back to the client at the address specified by *back=*. The following example shows two nodes connected by TCP/IP over an ESCON connection, with TCP/IP address *mvsescon1* used on one side of the ESCON connection and *risc1escon* on the other. *back=* is specified so when a client on *mvsescon1* starts a server on *risc1escon*, that server connects back to the client at *mvsescon1*.

```
risc1escon 1 4 risc1escon back=mvsescon1
```

In the same way, if a client on RS/6000 starts a server on *mvsescon1*, the client tells the server to connect back at *risc1escon*.

```
mvsescon1  2 4 mvsescon1  back=risc1escon
```

extra Is optional protocol-specific data, used to override defaults:

TCP/IP SNDBUF/RCVBUF buffer size
PCA Adapter number (for example, specify 2 for /**dev**/**cat2**)
ESCON Adapter number (for example, specify 2 for /**dev**/**esca2**)

The PARCONF file can contain blanks, blank lines, and comments. Any text following a pound sign (#) is treated as a comment.

Here are the PARCONF entries for each of the nodes in the configuration shown in Figure 2 on page 18:

- mvsnode: *user_ID*.**PARCONF** or PARCONF DD statement

```
riscnode1  1  3  hpssl1
riscnode2  1  4  hpssl2
```

- riscnode1: **/etc/parconf** or **/u/riscuser1/parconf**

```
mvsnode    2  3  kn1mvst
riscnode2  1  2  hpssl2
riscnode3  1  1  hpssl3
```

- riscnode2: **/etc/parconf** or **/u/riscuser2/parconf**

```
mvsnode    2  4  kn1mvst
riscnode1  1  2  hpssl1
```

- riscnode3: **/etc/parconf** or **/u/riscuser3/parconf**

  ```
  riscnode1  1  1  hpssl1.kgn.ibm.com
  riscnode1:mvsnode 2 3 kn1mvst
  ```

  The nickname `riscnode1:mvsnode` lets a tape server or tape pseudo device driver on ***riscnode3*** use ***riscnode1*** as an intermediate node for access to tape drives that are physically attached to ***mvsnode***. The nickname of the intermediate node, in this case `riscnode1`, must also be defined in the PARCONF file. The nickname of the remote host, in this case `mvsnode`, must be defined in the PARCONF file used on the intermediate node. The machine type and protocol values (in this example, `2` and `3`, respectively) describe the remote host ***mvsnode***, not the intermediate host directly attached to this system.

Each PARCONF file reflects the network configuration of its node. All nodes can share a single PARCONF file either by copying it or by using a shared file system.

Note: You can define multiple PARCONF entries for a single host. Use this technique to access a single host with multiple protocols, each protocol having a separate entry with a unique *nickname*. For example, ***riscnode2*** and ***mvsnode*** are connected by an ESCON adapter and TCP/IP. If the PARCONF file on ***mvsnode*** contains these entries:

```
riscnode2  1  4  hpssl2   # RS6K, protocol=ESCON
rs2dyn     1  0  hpssl2   # RS6K, protocol=dynamic
rs2tcp     1  1  hpssl2   # RS6K, protocol=TCP/IP
```

you can select one of three protocols (ESCON, dynamic, or TCP/IP) by specifying the associated *nickname*. For example, select TCP/IP by specifying **rs2tcp**.

Node Hopping with PARCONF

Many users use CLIO/S to transfer data between MVS and an SP. Often an SP with many nodes is connected to MVS via only one or two ESCON connections. In this set up, only those nodes that are directly connected to MVS can transfer data quickly with MVS. To allow all nodes on the SP to get data quickly, CLFTP, CLPLINK, and the tape server support transfers by using intermediate nodes. In this way, a node that is not directly connected to MVS can quickly get data by "hopping" through a node that is directly connected. For example, if node **node2** is not directly connected to MVS, but node **node1** is, data can be transferred to or from **node2** by hopping through **node1** as an intermediate node. The path to MVS is specified as a chain of hosts separated by colons (:), for example `node1:mvs`.

In the past, a user of CLFTP, CLPLINK, or the tape server needed to specify the path that he or she was hopping through on every transfer. For example, from **node2**, the user had to specify `node1:mvs`, while from **node1** he or she would specify just MVS. Thus the user always had to know where he or she was on the SP, and how to get to MVS. This method still works but now there is also the mapping function.

The CLIO/S mapping function provides a way to configure node hopping at configuration time, so that all users can hop through nodes transparently. Each mapping entry has the format:

```
nickname nodeaddress nicknamechain
```

Where:

nickname is an arbitrary name chosen for a destination node.

nodeaddress is the TCP/IP address of a node that is running the client part of CLFTP, CLPLINK, or the tape server. The asterisk (*) character can be used at the end of the node name to identify groups of nodes that share the same address.

nicknamechain is a chain of nicknames to which the first nickname maps.

Each mapping entry is interpreted to mean that if *nickname* is specified as a host name on node *nodeaddress*, then *nicknamechain* should be substituted for *nickname* in order to reach the desired destination node. For example, if node *mvsescon1* is the destination, and *mvsescon1* is reached from **node2** by hopping through **node1**, we could create a mapping called MVS from **node2** to *mvsescon1* as follows:

```
mvs node2 node1:mvsescon1
```

From **node1**, there is no need to hop, so the same mapping from **node1** to *mvsescon* would be:

```
mvs node1 mvsescon1
```

There can be a mapping for MVS for each node on which a client can run. As long as each node is specified, CLIO/S finds the correct mapping based on the node on which the application is running. The following shows mappings for every node on an eight-node SP.

```
mvs node1 mvsescon1
mvs node2 node1:mvsescon1
mvs node3 node1:mvsescon1
mvs node4 node1:mvsescon1
mvs node5 node1:mvsescon1
mvs node6 node1:mvsescon1
mvs node7 node1:mvsescon1
mvs node8 node1:mvsescon1
```

In order to avoid creating an entry for every node, the wild card character, asterisk (*), can be used. In the above example, we could specify the path for every node using only two mappings.

```
mvs node1 mvsescon1
mvs * node1:mvsescon1
```

In this case, a CLFTP, CLPLINK, or tape server running on **node1** goes directly to *mvsescon1*, while the same application running on any other node hops through **node1** to *mvsescon1*.

The wild card character, asterisk (*), can be used with partial names as well as by itself. If **node1** also has an address of *node1escon*, then we could specify a mapping for both **node1** and *node1escon* as follows.

```
mvs node1* mvsescon1
```

The *nicknamechain* can contain nicknames that are mapped to other nickname chains. In the following example, specifying MVS produces a chain of `node1:node2:mvsescon3`.

```
mvs     * node1:mvsdest
mvsdest * node2:mvsescon3
```

If CLIO/S finds a recursive loop while resolving mappings, it returns an error. For example, the following mappings for MVS can not be resolved since the mappings recursively include each other. It is impossible to find a final value for MVS.

```
mvs     * node1:mvsdest
mvsdest * node2:mvs
```

The following example PARCONF file shows node hopping on an eight-node SP system connected to an MVS system with ESCON connections on **node1** and **node5**.

```
------------------------------------------------------------------------
# mvsescon1 and mvsescon2 are tcp/ip addresses of MVS over an escon
# connection. These addresses can only be used by node1 for mvsescon1
# and node5 for mvsescon2.
mvsescon1  2 4 mvsescon1 back=node1escon
node1escon 1 4 node1escon back=mvsescon1
mvsescon2  2 4 mvsescon2 back=node5escon
node5escon 1 4 node5escon back=mvsescon2

# mvs1 is an ethernet or token ring address of the MVS system. In
# this example, we will assume that the RS/6000 SP
# can not contact MVS at this address.
# It can only contact MVS through mvsescon1 or mvsescon2.
mvs1   2 1 mvs1

# node1 through node8 are SP  nodes.
node1  1 1 node1
node2  1 1 node2
node3  1 1 node3
node4  1 1 node4
node5  1 1 node5
node6  1 1 node6
node7  1 1 node7
node8  1 1 node8

# The following will map nickname "mvs" for every node on the system.
# node1 through node4 will access MVS via mvsescon1.
# node5 through node8 will access MVS via mvsescon2.
# A clftp, clplink, or tape server user can access MVS by using the
# nickname "mvs" from any node on the SP.

# The first entry specifies that on node1, applications can map
# directly to mvsescon1 without hopping through any intermediate nodes.
mvs node1* mvsescon1

# The next three entries specify that node2 through node4 must map
# to mvsescon1 by hopping through node1.
mvs node2 node1:mvsescon1
mvs node3 node1:mvsescon1
mvs node4 node1:mvsescon1

# The next entry specifies that on node5, applications can map
# directly to mvsescon2 without hopping through any intermediate nodes.
mvs node5* mvsescon2

# The next entry specifies that from any other nodes (node6 through
# node8) applications must map to mvsescon2 by hopping through
# node5.
mvs * node5:mvsescon2

# The next entries are used when the destination is one of the nodes
# on the SP.  If MVS is the starting point, the application will hop
# through node1 or node5. If any other node is the starting point, the
# application goes directly to the node. When specifying MVS as the
# starting point, we must include entries for all of the valid tcp/ip
# addresses that can be used on MVS - mvsescon1, mvsescon2, and mvs1.
# We do this using the wildcard character '*' to identify all three
# mvs hosts.
```

```
n1 mvs* node1escon
n1 *         node1
n2 mvs* node1escon:node2
n2 *         node2
n3 mvs* node1escon:node3
n3 *         node3
n4 mvs* node1escon:node4
n4 *         node4
n5 mvs* node5escon
n5 *         node5
n6 mvs* node5escon:node6
n6 *         node6
n7 mvs* node5escon:node7
n7 *         node7
n8 mvs* node5escon:node8
n8 *         node8
-----------------------------------------------------------------------
```

The PARCTC File

The PARCTC file contains information CLIO/S needs to create CTC connections between mainframes and workstations. You must have this file on each workstation using a CTC connection. PARCTC is found in **$HOME**/**parctc** or /**etc**/**parctc** on RS/6000s. If both files are present, **$HOME**/**parctc** is used. A sample PARCONF file is included in /**usr**/**lpp**/**fcf**/**sample**. It must be modified to match your environment. You do not need a PARCTC file on MVS systems.

Like PARCONF, the PARCTC file consists of a series of one-line entries, one for each BMCA/PCA or ESCON connection. The syntax of each entry is:

hostnode workstationnode adaptername adapterid [*unitrange* [*localaddr*]]

Where:

hostnode Is the internet address of the remote host node. The address may be text or dotted decimal format.

workstationnode Is the internet address of the workstation. Use the address returned by the *hostname* command, which may not be the address being used by CLIO/S. A single PARCTC file can be shared across many nodes, and the **workstationnode** field is simply used to determine if the PARCTC entry being examined is for the node on which the current process is running. CLIO/S does a *hostname* call internally, and compares the result to this field. The address can be text or dotted decimal format.

adaptername Is a BMCA/PCA or ESCON adapter name. The format for *adaptername* is /**dev**/**cat***n* or /**dev**/**esca***n*, where *n* is the adapter number returned by the **lsdev -C -c adapter** command.

adapterid Is a unique integer corresponding to the MVS esoteric name defining the device address of the adapter. For example, if the adapter ID is 40, then the MVS IOCDS definition for that adapter must contain the esoteric name *RS6K40*. See “Configuring CTCs On MVS” on page 19 for more information.

unitrange Is an optional argument specifying the set of subchannels used by MVS for the adapter. A subset of the subchannels defined in the IOCDS may be used. Do not include CLAW subchannels in

this range; it doesn't generate an error, but performance is reduced and extraneous debug messages are printed.

localaddr is an optional argument specifying the local address associated with the first subchannel specified in the *unitrange* parameter. *unitrange* is required if *localaddr* is used.

The local address is set in the SMIT ESCON adapter subchannel definition panels. The format is L*nn*, where *nn* is the local address. For example, if you had subchannels 6C0 through 6C6 available, with 6C0 and 6C1 used by TCP/IP, then the SMIT 'Show defined Subchannels' panel display is:

```
Subchannel        System Name     Address
Set Name        Local    Remote   Loc Rem Group Type Path Connections
------------------------------------------------------------------------
ec1             TSERV8   KN1MVS   10  c0  2     CLAW e70  es0
ec2                               11  c2  1     3088 e70
ec3                               12  c3  1     3088 e70
ec4                               13  c4  1     3088 e70
ec5                               14  c5  1     3088 e70
ec6                               c6  c6  1     3088 e70
```

Note that the remote addresses in this case correspond to the last two digits of the MVS subchannel. This is common but not required. Then the corresponding PARCTC entries are:

```
kn1mvs   tserv8 /dev/esca0  30    6c2-6c5  L11
kn1mvs   tserv8 /dev/esca0  30    6c6-6c6
```

The 'L11' tells CLIO/S to use the local address 11 when it wants to connect to subchannel 6C2. CLIO/S increments the local address for each of the remaining subchannels defined in the *unitrange*, that is, local address 12 will be used for subchannel 6C3 even though it is not explicitly listed here. Subchannel 6C6 gets its own PARCTC entry since its local address is C6, which is what CLIO/S expects by default. It would be permissible to add Lc6 to that PARCTC entry if desired.

The PARCTC file can contain blanks, blank lines, and comments. Any text following a pound sign (#) is treated as a comment.

The PARCTC entries for the workstation nodes in the configuration shown in Figure 2 on page 18:

- riscnode1: /**etc**/**parctc** or /**u**/**riscuser1**/**parctc**

  ```
  kn1mvst hpssl1 /dev/cat0 20 660-66F
  ```

- riscnode2: /**etc**/**parctc** or /**u**/**riscuser2**/**parctc**

  ```
  kn1mvst hpssl2 /dev/esca0 5 6C0-6CF
  ```

riscnode3 does not need a PARCTC file because does not have a CTC attachment to the mainframe.

For total performance or redundancy reasons, you may want to have more than one channel adapter in a single node. Add additional PARCTC entries as required. For different adapters on the same node, the **workstationnode**, and **hostnode**

fields should be the same, while the **adaptername**, **adapterid**, and **unitrange** fields should be different.

To have a single CLIO/S process use more than one adapter simultaneously, see "Using the CLIO/S Striping Function in Your Application" on page 42 for details.

The NETRC File

The NETRC file is not CLIO/S-specific, but is used by other services such as **rsh**. CLIO/S uses NETRC for the same purpose as **rsh**, to provide user ID and authorization information. NETRC is found in **$HOME/.netrc** on RS/6000s. On MVS it is in *user_ID***.NETRC**, or it can be supplied with DD statements in the JCL for starting the application. Figure 36 on page 266 demonstrates how to supply NETRC information inline in your JCL. CLIO/S attempts to locate the NETRC DD statement first. If the DD statement is not found, then *user_ID***.NETRC** is used. A sample NETRC file is supplied with CLIO/S as **sys1.SFCFSAMP(FCFNETRC)** on MVS and in /**usr/lpp/fcf/sample** on AIX. You must copy this sample to the appropriate place and modify it to match your environment.

Here are the NETRC files for the nodes in the configuration shown in Figure 2 on page 18:

- mvsnode: *user_ID***.NETRC** or NETRC DD statement

```
machine hpssl1.kgn.ibm.com login riscuser1 password secret1
machine hpssl2.kgn.ibm.com login riscuser2 password secret2
```

- riscnode1: /**u/riscuser1/.netrc**

```
machine kn1mvst.kgn.ibm.com login mvsuser   password secret
machine hpssl2.kgn.ibm.com login riscuser2 password secret2
machine hpssl3.kgn.ibm.com login riscuser3 password secret3
```

- riscnode2: /**u/riscuser2/.netrc**

```
machine kn1mvst.kgn.ibm.com login mvsuser   password secret
machine hpssl1.kgn.ibm.com login riscuser1 password secret1
```

- riscnode3: /**u/riscuser3/.netrc**

```
machine hpssl1.kgn.ibm.com login riscuser1 password secret1
```

The CLIO/S Server—PARSRV

Most CLIO/S applications, with the exception of the CLIO/S sockets interface, use the PARSRV CLIO/S server to start jobs on remote hosts. This section describes the PARSRV program, the environment variables that PARSRV uses to control CLIO/S operations, and how you customize the CLIO/S environment by modifying the default values for the environment variables.

PARSRV is the CLIO/S server; it starts CLIO/S servant applications. PARSRV listens at a known socket waiting for applications to contact it. An administrator can start PARSRV and leave it running for others to use. Or, users can start their own PARSRV. There can be a single or multiple PARSRV servers on a host. There must be a PARSRV server started on each host on which you want to start CLIO/S remote applications. You can run CLIO/S master programs on hosts in your CLIO/S network that do not have a PARSRV server running, provided you don't want to start servants on that host.

One PARSRV server can be shared among multiple users (and applications) and by users on multiple hosts. Each CLIO/S application needs PARSRV only to establish the initial communication. After the client and server programs are communicating directly over the CTC or TCP/IP connection, PARSRV is not involved.

The steps involved in starting a PARSRV server are:

Step	on page
Deciding which port to use	35
Setting environment variables	36
Selecting the JCL Generation method (MVS only)	40
Starting the PARSRV server	40
Stopping the Server	40

The following table lists the customizable programs you use to start CLIO/S. Other CLIO/S support programs are described in Appendix C, “Support Programs and Sample JCL” on page 263.

Table 2. Programs For Starting CLIO/S

Program name	Function	Resides on	Purpose	Page	Notes
FCFPARSR, **parsrv**	The CLIO/S server	MVS and AIX	Submits CLIO/S jobs	34	If there are more than one or if not using default port number, each user needs to know the port number.
fcfconf.c	Some CLIO/S defaults	MVS and AIX	Default environment variables and global variables	41	Compile and relink CLIO/S programs after changing.

Selecting a Port Number

By default, PARSRV listens on TCP/IP port 6667 for a CLIO/S application to request a connection. Use the CLIOPSPT and CLIOPSID environment variables to override the default PARSRV port. Be sure to set the environment variables before starting the server, as described in “Setting CLIO/S Environment Variables” on page 36. You can also run multiple PARSRVs on a single host, each with its own port number (see “Running Multiple PARSRVs” on page 41 for more information).

If your installation chooses to have a single PARSRV on each host, all applications can use the default port number defined in **fcfconf.c**. The CLIO/S product defaults to port 6667; you can change as described in “Changing Defaults for Some CLIO/S Environment Variables” on page 41.

This is the algorithm that CLIO/S uses to determine the port used by PARSRV:

```
Get environment variable CLIOPSPT;
Get environment variable CLIOPSID;
if (CLIOPSPT != 0) {
  port = CLIOPSPT;
} else {
  service = getservbyname(CLIOPSID, "tcp");
  if (service == NULL) {
    port = Cliopspt;  /* default=6667,
                         from clioconf.c */
  } else {
    port = service->s_port;
  }
}
```

Setting CLIO/S Environment Variables

When your CLIO/S servant program is started, PARSRV passes it the value of the master program's CLIO/S environment variables. The master's values override any defaults established on the server system.

The way the master process initializes environment variables depends on whether you start your master program on AIX or on MVS. In the AIX environment, use the **setenv** (C shell) or **export** (Korn shell) commands to set the environment variables before you start your master application. For example, a C shell user who wants to see all CLIO/S debug messages issues:

```
setenv CLIODEB -1
```

A Korn shell user who wants to see only messages relating to opening and closing a CTC connection issues:

```
export CLIODEB=8192
```

MVS does not have the UNIX environment facility. The environment in MVS is defined in the CLIOENV DD statement in the JCL that starts the master or in the file *user_ID*.**CLIOENV**.

For example, to get messages relating to tape allocation, include this line in *user_ID*.**CLIOENV** or following the CLIOENV DD statement:

```
CLIODEB=256
```

The *user_ID*.**CLIOENV** data set must be allocated with the following characteristics:

Record Format	Fixed, or fixed block with one record per block
Logical record length	80
Blocksize, if fixed block records	80
Data set organization	sequential
Sequence numbers	no

In CMI and DPI applications you can change some of these environment variables for a specific servant using the **clio_exp** subroutine or the specialized subroutine listed under the environment variable's description below.

CLIO/S's environment variables are listed in the following section:

CLFTPINPUT AIX only. When this variable is specified, input is taken from the filename specified, rather than from stdin.

MVS uses DD INPUT instead of CLFTPINPUT and input is read from DD INPUT instead of stdin.

CLFTPOUTPUT AIX only. When this variable is specified, output is sent to the filename specified, rather than stdout.

MVS uses DD OUTPUT instead of CLFTPINPUT, and output is sent to the file specified in DD OUTPUT instead of stdout.

CLIOADEB Contains flags defined by the CLIO/S application. Use this environment variable to pass information from the master to the servant process.

CLIOARB Provided for use with **FCFSHELL**. See "FCFSHELL" on page 299

CLIOBIN Sets the private directories for UNIX search path when a process is being started. The default value is **$HOME/bin**. See "Changing Defaults for Some CLIO/S Environment Variables" on page 41 for more information about defaults. The total path used when starting a new process is PATH = $CLIOBIN:$CLIOSYS:$CLIOPATH. The path statement is a colon-delimited list of directories.

On MVS this environment variable is ignored.

CLIOBLK Sets the buffer size that turns off buffering. See "CLIOBLK—Setting the Buffer Bypass Limit" on page 200 for more information. The default CLIO/S buffer limit is 32768 (32K)

CLIOBUF Sets the size of CLIO/S buffer. See "CLIOBUF—Setting the Size of the Internal Buffer" on page 200 for more information. The default CLIO/S buffer size is 32K. The minimum buffer size is 1024 (1K).

CLIODEB Sets debug flags.

CLIO/S generates many messages as it performs its functions. If CLIODEB is set to one of the following values, the messages are printed to the user's terminal. These messages are provided "as-is", with no other explanation, but they can prove very helpful in establishing the CLIO/S environment and developing CLIO/S applications.

Messages are issued for both master and servant programs. The value of CLIODEB is the same for both master and servant processes, because the master process's CLIODEB setting is copied when the servant is started. Examining messages on both master and servant can help you debug problems. Messages on MVS and RS/6000 are similar but not identical.

The default value of CLIODEB is 0, which requests that no messages be displayed. You set the CLIODEB environment variable to indicate the level of messages you want to receive. You can set CLIODEB using either hexadecimal or decimal notation. For example, `CLIODEB=0x2000` is equivalent to `CLIODEB=8192`. Table 3 on page 38 lists the debug flags and their meanings.

You can select multiple messages by adding values together. For example, to receive messages about stopping and starting clients (0x01). and about events related to the PARCONF file (0x20), set CLIODEB to the value 0x21.

Table 3. Flags for the CLIODEB Environment Variable

Decimal	Hex	Messages displayed
–1	0xFFFF	Displays all messages
0	0x00	No status info printed
1	0x01	Starting/stopping clients
2	0x02	SCHEDULE/SYNCHRO/WAITREQ and ALERT/WAITFOR
4	0x04	Connect/disconnect
8	0x08	MSANY/SMANY and SPANY/RPANY
16	0x10	Actual data transfer and SETSRV calls
32	0x20	Parconf file
64	0x40	Reserved
128	0x80	Reserved
256	0x0100	Tape allocation, unallocation, open and close
512	0x0200	Tape read and write
1024	0x0400	Tape write EOF, rewind, skip file, skip record, and write to operator
2048	0x0800	Tape interface to CLIO/S communication subroutine library
4096	0x1000	TCP/IP calls.
8192	0x2000	CTC open or close to trace abnormal but acceptable situations
16384	0x4000	CTC read or write to trace normal data flow
32768	0x8000	Print detailed error messages about CTC read or write

CLIODFILE Redirects debug messages normally sent to STDOUT. If CLIODFILE is set to the name of a file (or dataset on MVS), the debug messages will be written to that file.

CLIOEFILE Redirects error messages normally sent to STDERR. If CLIOEFILE is set to the name of a file (or dataset on MVS), the error messages are written to that file.

TAPEMSG Specifies a message directory. See “Specifying a Message Directory with the TAPEMSG Variable” on page 246.

CLIONODE Specifies the local host name when you don't want to use the host name returned by the **gethostname()** call.

If your installation has installed more than one instance of TCP/IP on a host, each TCP/IP instance is associated with a specific local host name. You should set CLIONODE to the local host name associated with the instance of TCP/IP you want to use.

CLIOPATH Sets the other directories for UNIX search path when a process is being started. The default value is the list of directories specified in the UNIX environment variable **$PATH**. See “Changing Defaults for Some CLIO/S Environment Variables” on page 41 for

more information about defaults. The total path used when starting a new process is
PATH = $CLIOBIN:$CLIOSYS:$CLIOPATH.

On MVS this environment variable is ignored.

CLIOPSID Sets the string used to identify PARSRV when calling **getservbyname()**. The default value is "parsrvd".

If PARSRV is registered for use with **getservbyname()**, **getservbyname()** returns the port for the server registered with the matching name. Together with CLIOPSPT, CLIOPSID identifies the correct instance of PARSRV if multiple PARSRVs are running. The PARSRV program and any applications that access it must have their CLIOPSID variables set to identical values.

A system administrator can change the default value, as described in "Changing Defaults for Some CLIO/S Environment Variables" on page 41.

CLIOPSPT Sets the port that PARSRV should use. The default is port number 6667. Together with CLIOPSID, CLIOPSPT identifies the correct instance of PARSRV if multiple PARSRVs are running. The PARSRV program and any applications that access it must have their CLIOPSPT variables set to identical values.

A system administrator can change the default value, as described in "Changing Defaults for Some CLIO/S Environment Variables" on page 41.

CLIOSHELL Controls whether **FCFSHELL** is used to start the remote process. See "FCFSHELL" on page 299.

CLIOSTAE Controls whether ESTAE processing is enabled. The default is to enable ESTAE. Any value other than NO enables ESTAE processing.

On AIX this environment variable is ignored.

CLIOSTRIPES Sets the number of data stripes. The default value is 1. The maximum value is 4. See "Using the CLIO/S Striping Function in Your Application" on page 42 for more information about striping.

CLIOSYS Sets the installation directories for UNIX search path when a process is being started. The default value is **/usr/lpp/fcf**. See "Changing Defaults for Some CLIO/S Environment Variables" on page 41 for more information about defaults. The total path used when starting a new process is
PATH = $CLIOBIN:$CLIOSYS:$CLIOPATH.

On MVS this environment variable is ignored.

CLIOTBS Sets the TCP/IP buffer sizes. See "CLIOTBS—Changing Send and Receive Buffer Sizes" on page 203 for more information. The default value of CLIOTBS is zero (0), which indicates that CLIO/S should use the system default TCP/IP buffer size.

CLIOTMO Sets the time-out value in seconds. See "CLIOTMO—Setting the Time-out Period" on page 203 for more information. The default value of CLIOTMO is 20 seconds.

Reserved CLIO/S Environment Variables

The following environment variable names are reserved for CLIO/S use. Do not use them as user-defined environment variables:

CLFTPSRV
CLIODBGFILE
CLIOFLAG
CLFTPFAKEIOLOC
CLFTPFAKEIOREM
CLIOF_PORT
CLIOF_STATFREQ
CLIOF_MAXBUFLEN
CLIOF_FIRSTBUFLEN
CLIOF_SECONDBUFLEN
CLPLINKTIMING
CLPLINKX

Customizing the JCL Generation Language

To use PARSRV on MVS, you must modify the FCFREXX JCL generation exec to match your MVS environment. See Appendix D, “Controlling How CLIO/S Servers Are Started” on page 279 for more information.

Starting the PARSRV Server

On MVS you start PARSRV by submitting JCL. Sample JCL for starting PARSRV as a started task is provided in **SYS1.SFCFSAMP(FCFPARSR)**. You may modify this JCL to start PARSRV as a batch job. The JCL that starts PARSRV specifies the data sets containing the jobs that PARSRV can start. On MVS, logging information for PARSRV is written to the STDOUT and STDERR files.

On AIX, PARSRV is started with the **parsrv** command. **parsrv** is installed in the directory /**usr**/**lpp**/**fcf**/**bin**. The AIX **parsrv** submits jobs in the directories specified by the path $CLIOBIN:$CLIOSYS:$CLIOPATH. These environment variables are described on pages 37 through 39. On AIX, normal processing messages are written to stdout; error messages are written to stderr. You can redirect stdout and stderr to a file to log the activities of the PARSRV server. For example:

```
parsrv 1> parsrv.log 2>&1
```

For both AIX and MVS, if you try to start the PARSRV program on a port where another PARSRV is already running or has recently stopped, you get the following message:

```
CLIO/S parallel server failed to start on nodeid
Probable reason:
  Another parallel server is either running or has been
  stopped very recently.
  Please wait a few seconds for TCP/IP to release port nnnn
```

Stopping the Server

Servant jobs already submitted by PARSRV are not affected when you stop the PARSRV server.

See “Stopping a CLIO/S Server—FCFPARST, parstop” on page 271 for more information about stopping a CLIO/S server.

Changing Defaults for Some CLIO/S Environment Variables

The **fcfconf.c** file shipped with CLIO/S lets an installation customize the defaults for certain CLIO/S environment variables. On MVS **fcfconf.c** is in **SYS1.SFCFSAMP(FCFCONF)**. On AIX it is shipped in the file **/usr/lpp/fcf/sample/fcfconf.c**. The environment variables whose defaults can be specified are CLIOBIN, CLIOSYS, CLIOPATH, CLIOPSID, and CLIOPSPT. To change these defaults, modify the appropriate values in **fcfconf.c**, compile **fcfconf.c** with the sample Makefile, and link all CLIO/S programs with the resulting object code.

Running Multiple PARSRVs

You can run multiple PARSRVs on a host. Running multiple PARSRVs on MVS lets you generate different JCL to start different types of servant processes.

If your installation runs multiple PARSRVs, CLIO/S commands and applications must specify which to use. Do this by setting the CLIOPSPT and CLIOPSID environment variables. CLIOPSPT is the port number on which your desired PARSRV is listening. If PARSRV is registered as a server in **/etc/services** (on AIX) or **TCPIP.profile.TCPIP** (on MVS) and you run multiple PARSRV servers on a single host, each PARSRV server on that host must have a unique CLIOPSID value; PARSRV and all applications that contact it must set CLIOPSID to the appropriate value. “Setting CLIO/S Environment Variables” on page 36 describes how you set CLIO/S environment variables.

Ensuring Password Security

CLIO/S masks passwords before sending them over the network. IBM provides source code for the password lookup, masking, and unmasking subroutines used by CLIO/S. You can change these routines to meet the security requirements of your installation. Be sure to make corresponding change on all hosts in your CLIO/S network.

Source code is included in **SYS1.SFCFSAMP(FCFFINDP)** on MVS and in **/usr/lpp/fcf/sample/fcffindp.c** on AIX. The functions provided are:

- **find_pw**(*node*)

  This function returns a structure containing the user ID and password associated with that node name. The default is to find the information in the user's **.netrc** file.

- **encr_pw**(*arg*)

  The argument is the structure returned by **find_pw**. The function returns structure containing an masked version of the argument.

- **decr_pw**(*arg*)

  This function converts the masked structure back to the original structure containing user ID and password.

find_pw and **encr_pw** are called by the master program. **decr_pw** is called by PARSRV.

When the master runs on MVS, PARSRV on the workstation starts the servant processes using the **rexec** subroutine. This requires that the user ID and password are supplied from the MVS side. This information comes from the MVS data set *user_ID*.**NETRC**.

Using the CLIO/S Striping Function in Your Application

The CLIO/S striping function allows a single CLIO/S data transfer to be distributed across two or more channel adapters to a single host. This can significantly improve performance if you have a process than can transfer data at rates higher than a single channel can support. Otherwise, there is little or no performance benefit. It is also possible to stripe across multiple subchannels on a single adapter, but the performance benefit is less than when multiple adapters are used.

Striping across multiple adapters is allowed, following these rules:

- All adapters must be the same type, either all ESCON or all BMCA/PCA
- All adapters must be attached to the same MVS host

CLIO/S's striping support requires no special installation or configuration steps for the adapter. The following sections describe how you enable CLIO/S to use striping and how to monitor striping activity.

Enabling CLIO/S for Striping

To use striping in a CLIO/S application, you must:

- Modify the PARCTC file
- Set the CLIOSTRIPES environment variable

Modifying PARCTC for Striping

Using Multiple Adapters: To enable CLIO/S to use multiple adapters you must put entries for each into the PARCTC file. This example configures two stripes using two adapters:

```
kn1mvst hpssl1 /dev/cat0 20 660-66F
kn1mvst hpssl1 /dev/cat1 50 650-65F
```

Using A Single Adapter: To stripe across multiple subchannels on a single adapter in the configuration shown in Figure 2 on page 18, you must duplicate the adapter's entries in the PARCTC file. The entries can define different subchannels, or they can be identical and CLIO/S will select available subchannels dynamically. To configure four stripes between ***riscnode1*** and ***kn1mvst*** on one adapter, create a PARCTC file with four entries. To let CLIO/S select the subchannels, code /**etc**/**parctc** on ***riscnode1*** like this:

```
kn1mvst hpssl1 /dev/cat0 20 660-66F
kn1mvst hpssl1 /dev/cat0 20 660-66F
kn1mvst hpssl1 /dev/cat0 20 660-66F
kn1mvst hpssl1 /dev/cat0 20 660-66F
```

To select the subchannels in each stripe, code /**etc**/**parctc** on ***riscnode1*** like this:

```
kn1mvst hpssl1 /dev/cat0 20 660-663
kn1mvst hpssl1 /dev/cat0 20 664-667
kn1mvst hpssl1 /dev/cat0 20 668-66B
kn1mvst hpssl1 /dev/cat0 20 66C-66F
```

Setting the CLIOSTRIPES Environment Variable

Set the CLIO/S internal environment variable CLIOSTRIPES to indicate the number of stripes you want. The maximum is 4. See “Setting CLIO/S Environment Variables” on page 36 for details.

Monitoring Striping Activity

To observe the striping function in action, use CLIO/S debug messages. Set CLIODEB to 8192 (0x2000) and examine the messages issued as the CTC connections are established. On the MVS console, use display commands such as `/d u,,,660` to observe the status of the CTC subchannels.

Using CLIO/S and Non-CLIO/S Applications on the Same Adapter

CLIO/S can share a channel adapter with other applications. They can be either CTC applications (like CLIO/S) or applications that use the Common Link Access to Workstations (CLAW) protocol.

Multiple processes sharing the same adapter may result in decreased performance.

Sharing with Other CTC Applications

If your CLIO/S application is sharing an adapter with another CTC application, only one application can have each subchannel. For example, if CLIO/S is using subchannel 2, then another application can't use subchannel 2, but could use subchannel 3. Each active CLIO/S process will use a subchannel, or more than one if striping is used. If CLIO/S does not find an available subchannel, it uses TCP/IP instead.

No special configuration is required for CLIO/S to share a channel adapter with another CTC application.

Sharing with CLAW Applications

CLIO/S can also share a channel adapter with a CLAW application, for example, TCP/IP. CLIO/S requires a TCP/IP connection between MVS and the workstation, but that requirement can be met by running TCP/IP over the same channel adapter card that CLIO/S uses.

CLIO/S uses CTC mode subchannels, while TCP/IP uses CLAW mode subchannels. An adapter that is shared between them must be configured such that some subchannels use CTC mode while others use CLAW mode. “Configuring and Starting BMCA/PCA Subchannels” on page 21 and “Configuring ESCON Subchannels” on page 23 describe using the SMIT configuration tool to configure subchannels on an adapter. CTC subchannels are used one at a time, but usually you will want to assign a range of possible subchannels. CLIO/S searches the range until it finds an available subchannel. CLAW subchannels are assigned in pairs, beginning on an even subchannel.

Complete procedures for configuring a BMCA/PCA or ESCON adapter for TCP/IP are in the *IBM Block Multiplexer Channel Adapter User's Guide and Service Information*, SC31-8196, or the *IBM ESCON Channel Adapter User's Guide and Service Information for AIX 4.1*, SC31-8197.

Enabling TSO Access to the CLFTP and CLPLINK Commands

To let users invoke CLFTP from a TSO session, the CLIO/S load library must be allocated under DD name EDCMTF and added to your system STEPLIB concatenation. A TSO system administrator can do this for all users by adding the following DD statements to the LOGON PROCEDURE (LOGON PROC):

```
//EDCMTF   DD DSN=SYS1.SFCFLOAD,DISP=SHR
//STEPLIB  DD DSN=SYS1.SFCFLOAD,DISP=SHR
```

Alternatively, a TSO user can issue this command before using either the CLFTP or CLPLINK command:

```
ALLOC DD(EDCMTF) DSN(SYS1.SFCFLOAD) SHR REUSE
```

Chapter 4. Configuring and Managing the CLIO/S Tape Pseudo Device Driver

Note

Before using the procedures in this chapter, you must have installed CLIO/S on MVS, downloaded its files to the AIX workstation, and installed the optional CLIO/S tape server feature on AIX. See the CLIO/S Program Directory for instructions for installing CLIO/S on MVS, downloading files to the workstation, installing CLIO/S on AIX, and running the Installation Verification Procedure (IVP).

CLIO/S provides a standard AIX interface to the CLIO/S tape server (TAPESRV), enabling users to access tape data on remote MVS and RS/6000 systems using standard UNIX commands, for example, **tar**, **cpio**, or **backup**.

An administrator make this interface available to users on an AIX host by configuring a CLIO/S tape mount daemon for each tape device you want to make available. This is an optional step. If you do not want to provide the standard UNIX interface to CLIO/S tapes, or if you are not the person at your installation who performs this task, skip this chapter. If CLIO/S's optional tape server feature is installed, the CLIO/S tape application programming interface is available, independent of whether anyone performs the tasks in this chapter to provide access through UNIX commands.

You don't need to do any special installation or configuration on MVS, even if the tape drives you want to share are physically attached to the MVS host.

This table shows the typical tasks a CLIO/S administrator performs to provide a standard UNIX interface to tape devices:

Task	Page
Configuring and starting the CLIO/S tape mount daemon	45
Monitoring tape mount daemon activity	48
Stopping the tape mount daemon	49
Changing the tape pseudo device driver	49
Reconfiguring the tape pseudo device driver/restarting the tape mount daemon	50
Restarting the tape pseudo device driver after a system reboot	50
Unconfiguring the tape pseudo device driver	51
Removing the tape pseudo device driver	52

Note: CLIO/S supports the “norewind” tape special file (/**dev**/**pst***n***.1**). However, you need to create only the /**dev**/**pst***n* tape pseudo device driver; the user specifies whichever device is appropriate, /**dev**/**pst***n* or /**dev**/**pst***n***.1**.

Configuring and Starting the CLIO/S Tape Mount Daemon

You configure and start one CLIO/S tape mount daemon for each tape device you want to share between hosts. The tape mount daemon runs on the AIX host on which you want to create a standard UNIX interface, even if the tape device is physically attached to an MVS host.

© Copyright IBM Corp. 1993, 1995, 1996

The sample CLIO/S configuration in Figure 2 on page 18 shows two tape devices. One is physically attached to an MVS system, the other to an AIX system. This section describes how to configure ***riscnode3*** to access both of these tape drives. These steps configure and start the tape mount daemon:

1. Login with root authority.
2. If your installation has started more than one CLIO/S server (PARSRV) on the host to which the tape drive is attached, set the CLIOPSPT environment variable to the appropriate port number. For example, if you want the tape mount daemon to use a PARSRV server that is listening on port 1955 and you are running on the Korn shell, set the environment variable with this command:

   ```
   export CLIOPSPT=1955
   ```

   If you are using the C shell, use:

   ```
   setenv CLIOPSPT 1955
   ```

   Multiple tape mount daemons can use the same PARSRV.

   Note: PARSRV must be running at this point. If you have not already started PARSRV, refer to "The CLIO/S Server—PARSRV" on page 34.

   If PARSRV is registered in **/etc/services** on AIX or in **TCPIP.profile.TCPIP** on MVS, you must set the CLIOPSID environment variable to the name corresponding to the server/port combination you want to use.
3. Invoke SMIT to perform device configuration:

   ```
   smit
   ```

   You can install if your operating system is at a different level—the panels you see may be slightly different. Make the following selections in the SMIT panels.
4. On the Systems Management panel, select 'Devices'
5. On the Devices panel, select 'Tape Drive'

   Note: You can skip directly to this menu using the command **smit tape**.
6. On the Tape Drive panel, select "Add a Tape Drive"
7. In the Single Selection List of device types, select "cliotape"

   Note: If your menu doesn't list the "cliotape" type, the CLIO/S tape pseudo device driver wasn't installed correctly. Exit from SMIT and try installing again, as described in the Program Directory.

   A menu similar to the following is displayed. The fields you fill in or modify are shown in bold face type.

```
                                Add a remote tape

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

  Tape drive type                                      cliotape
  Tape drive interface                                 remote
  Description                                          CLIO remote tape
  Remote host                                          riscnode1
  Allocation parameter / device name                   /dev/rmt0
  Maximum block size                                   32768
  Unallocate on close ?                                no
  Delayed open ?                                       yes
  Default volser for allocation                        DUMMY
```

Figure 8. Adding a Remote Tape for CLIO/S

8. Change the following fields on this panel:

   a. Update the Remote host field with the name of the remote host to which the tape drive is attached. The remote host used in this example is riscnode1.

      As seen in Figure 2 on page 18, for ***riscnode3*** to access the tape drive on the MVS system ***mvsnode***, it must use an intermediate system, ***riscnode1***. To configure ***riscnode3*** to access this tape drive, specify both host names in the Remote Host field by using the routed nickname ***riscnode1***:***mvsnode*** instead of ***riscnode1***. The routed nickname must be defined in the PARCONF file on the local AIX system. "The PARCONF File" on page 26 explains how to specify nicknames for intermediate hosts.

      Note: For best performance, do not use CLIO/S tape support over an Ethernet or token ring TCP/IP connection. If you specify a PARCONF nickname in this field, it should not be one that specifies the TCP/IP protocol. If you specify an internet name:

      - When connecting MVS and AIX systems, be sure an ESCON or BMCA/PCA adapter is available
      - When connecting AIX machines be certain that TCP/IP is set up to use a fast connection such as FDDI or HIPPI.

   b. Update the Allocation parameter / device name field.

      AIX For tape drives physically attached to AIX, you supply the name of the tape device on the remote host, for example, /**dev**/**rmt0**. The tape device used in this example is /**dev**/**rmt0**.

      MVS For tape drives physically attached to MVS, you supply the default MVS allocation parameters separated by underscores. For example, to define the DD name of TAPE01 on a 3480 tape drive using no label (NL) processing, put the value **DD**=**TAPE01_UNIT**=**3480_NL_DSN**=*dataset-name* in this field. Users will want to know the default MVS dataset name.

      These default parameters can be overridden by using the corresponding flags on the **TPALLOC** function and **cltalloc** command. See "cltalloc—Allocating a CLIO/S Tape Device for MVS" on page 84 for more information on allocation parameters.

Make sure the machine type field in the PARCONF file accurately defines the physical host type of the system to which the device is physically attached. If they don't match, you can complete this configuration, but users will not be able to allocate the device using the **cltalloc** command.

c. Update the Unallocate on close ? field, specifying “yes” or “no.”

yes When a command (for example, **tar**) issues a close, the tape device is unallocated (except for AL and SL tapes).

no When a command (for example, **tar**) issues a close, the tape device is not automatically unallocated; the tape stays mounted.

d. Update the Delayed open ? field, specifying “yes” or “no.”

yes The tape device waits for a read, write, or **ioctl** before the device is opened. This is useful for backup operations.

no The device will be opened when allocated without waiting for a read or write.

e. Update the Default volser for allocation field.

AIX This field is ignored on AIX.

MVS Specifies the default volser for tape allocation.

9. Press Enter or select “Do” to continue.

After you complete these steps, SMIT automatically configures the CLIO/S tape pseudo device driver and starts the CLIO/S tape mount daemon. To configure additional tape drives, repeat from step 6 on page 46.

SMIT automatically names the tape pseudo device drivers, starting with **/dev/pst0** for the first and continuing, if defined, up to **/dev/pst99**. The UNIX file permissions for the CLIO/S tape pseudo device driver allow all users to read and write to the device. To restrict access to the tape pseudo device driver, use standard UNIX access control methods, for example, **chmod** and **chown**.

Monitoring Tape Mount Daemon Activity

The tape mount daemon records information about its activities by:

- Writing to a log file, one file per daemon
- Using the AIX error logging facility

The tape mount daemon writes informational and error messages into the log file **/tmp/cltclntd.pst***n*. For example, the tape mount daemon associated with tape pseudo device driver **/dev/pst1** records activities in the log file **/tmp/cltclntd.pst1**. When you stop a daemon, the log remains on the system; when you restart the daemon, the new log replaces the old one.

Using the AIX error logging facility, the tape mount daemon puts records into the system error log. Read the system error log using the **errpt** command. This log contains information about all CLIO/S tape mount daemons, and is not affected by stopping or restarting a daemon.

Stopping the CLIO/S Tape Mount Daemon

Stopping the tape mount daemon prevents users from using standard UNIX commands to access CLIO/S tapes. You do not need to do this during normal system operation. Circumstances that might require you to stop the tape mount daemon include:

- You want to change the characteristics of the tape pseudo device driver
- There are hardware problems with the physical device

Stop the tape mount daemon with the **cltterm** command by following these steps:

1. Login with root authority.
2. Make sure the device is not being used by any users on your system. If necessary, use **cltunall** to unallocate it.

   The /**tmp**/**cltclntd.pst***n* file shows whether the device is currently allocated.
3. Use **cltterm** to stop the tape mount daemon.

   ```
   cltterm -f /dev/pst0
   ```

   When the tape mount stops, the system responds:

   ```
   cltterm: daemon on /dev/pst25 is now terminated.
   ```

To restart the daemon, follow the procedures in “Reconfiguring the Tape Pseudo Device Driver/Restarting the CLIO/S Tape Mount Daemon” on page 50.

Changing the CLIO/S Tape Pseudo Device Driver

If you want to change the default allocation parameters, change the CLIO/S tape pseudo device driver. To override the defaults for a specify tape, use **cltalloc**'s command line flags. Permanently change the daemon by following these steps:

1. Login with root authority.
2. Make sure the tape device is not being used by any users on your system. If necessary, use **cltunall** to unallocate it.

   The /**tmp**/**cltclntd.pst***n* file shows whether the device is currently allocated.

   Changing the characteristics of the tape pseudo device driver while it is allocated to a user has unpredictable results.
3. Use **cltterm** to stop the tape mount daemon, as shown in “Stopping the CLIO/S Tape Mount Daemon.”
4. Use SMIT panels to change the tape pseudo device driver:

   ```
   smit tape
   ```
5. On the Tape Driver panel, select “Change / Show Characteristics of a Tape Drive.”

   Verify that the tape drive is listed as “Defined” (rather than “Available”). If the device is not “Defined,” follow the instructions in “Unconfiguring the CLIO/S Tape Pseudo Device Driver” on page 51. Then return to this step and continue.
6. Select your tape drive from the Single Selection List that is displayed.

7. On the Change / Show Characteristics of a Remote Tape Drive panel, make the desired changes.
8. Press Enter or select “Do” to change the characteristics of the daemon.
9. Reconfigure the tape pseudo device driver and restart the tape mount daemon as described in “Reconfiguring the Tape Pseudo Device Driver/Restarting the CLIO/S Tape Mount Daemon.”

Reconfiguring the Tape Pseudo Device Driver/Restarting the CLIO/S Tape Mount Daemon

When you add a new tape pseudo device driver, SMIT automatically configures the pseudo device driver and starts the tape mount daemon. You need to configure/start after you have stopped the daemon or unconfigured the pseudo device driver, following these steps:

1. Login with root authority.
2. If your installation has started more than one CLIO/S server (PARSRV) on the host to which the tape drive is attached, set the CLIOPSPT environment variable to the appropriate port number. For example, if you want the tape mount daemon to use a PARSRV server that is listening on port 1955 and you are running on the Korn shell, set the environment variable with this command:

   ```
   export CLIOPSPT=1955
   ```

   If you are using the C shell, use:

   ```
   setenv CLIOPSPT 1955
   ```

   Multiple tape mount daemons can use the same PARSRV.

   If PARSRV is registered in **/etc/services** on AIX or in **TCPIP.profile.TCPIP** on MVS, you must set the CLIOPSID environment variable to the name corresponding to the server/port combination you want to use.
3. Invoke SMIT to perform device configuration:

   ```
   smit tape
   ```
4. On the Tape Drive panel, select “Configure a Defined Tape Drive”
5. Select your tape drive from the Single Select List menu that appears.

You can also restart the daemon use the **cltclntd** command. For example, restart the daemon associated with tape pseudo device driver **/dev/pst0** with these commands:

```
su root
cltclntd -f /dev/pst0
```

Restarting the Tape Pseudo Device Driver After a System Reboot

The tape pseudo device driver is not automatically restarted when the system reboots. To restart the tape pseudo device driver, use the **mkdev** command. For example, to restart tape pseudo device driver **/dev/pst1** after the system reboots, issue this command:

```
mkdev -l pst1
```

You can also restart the tape pseudo device driver using SMIT, using these steps:

1. Login with root authority.
2. If your installation has started more than one CLIO/S server (PARSRV) on the host to which the tape drive is attached, set the CLIOPSPT environment variable to the appropriate port number. If PARSRV is registered in **/etc/services** on AIX, set CLIOPSID to the appropriate string. Use the same CLIOPSPT and CLIOPSID values you used in "Configuring and Starting the CLIO/S Tape Mount Daemon" on page 45.
3. Invoke SMIT to perform device configuration:

   ```
   smit tape
   ```

4. On the Tape Drive panel, select "Configure a Defined Tape Drive"
5. Select your tape drive from the Single Select List that appears.

Unconfiguring the CLIO/S Tape Pseudo Device Driver

To unconfigure the tape pseudo device driver:

1. Login as **root**.
2. Make sure the device is not in use by another process. If necessary, use the **cltunall** command to unallocate the device.

   The **/tmp/cltclntd.pst***n* file shows whether the device is currently allocated.
3. Use **cltterm** to stop the tape mount daemon, as shown in "Stopping the CLIO/S Tape Mount Daemon" on page 49.
4. If your installation has started more than one CLIO/S server (PARSRV) on the host to which the tape drive is attached, set the CLIOPSPT environment variable to the appropriate port number. For example, if you want the tape mount daemon to use a PARSRV server that is listening on port 1955 and you are running on the Korn shell, set the environment variable with this command:

   ```
   export CLIOPSPT=1955
   ```

   If you are using the C shell, use:

   ```
   setenv CLIOPSPT 1955
   ```

   Multiple tape mount daemons can use the same PARSRV.

   If PARSRV is registered in **/etc/services** on AIX or in **TCPIP.profile.TCPIP** on MVS, you must set the CLIOPSID environment variable to the name corresponding to the server/port combination you want to use.
5. Use SMIT to "unconfigure" the tape pseudo device driver:

   ```
   smit tape
   ```

6. On the Tape Driver panel, select "Remove a Tape Drive."
7. Select your tape drive from the Single Selection List that is displayed.
8. On the Remove a Tape Drive panel, set the "KEEP definition in database" field to "yes" (the default).
9. Press Enter or select "Do" to process.

The tape pseudo device driver is now in the "defined" state. To reconfigure it, see "Reconfiguring the Tape Pseudo Device Driver/Restarting the CLIO/S Tape Mount Daemon" on page 50.

Removing the CLIO/S Tape Pseudo Device Driver

You can remove the CLIO/S tape pseudo device driver. This is usually necessary only if you physically remove the tape device from the remote host. These steps remove the CLIO/S tape pseudo device driver:

1. Login as **root**.
2. Make sure the device is not in use by another process. If necessary, use the **cltunall** command to unallocate the device.

   The /**tmp**/**cltclntd.pst***n* file shows whether the device is currently allocated.
3. Use **cltterm** to stop the tape mount daemon, as shown in "Stopping the CLIO/S Tape Mount Daemon" on page 49.
4. If your installation has started more than one CLIO/S server (PARSRV) on the host to which the tape drive is attached, set the CLIOPSPT environment variable to the appropriate port number. For example, if you want the tape mount daemon to use a PARSRV server that is listening on port 1955 and you are running on the Korn shell, set the environment variable with this command:

   ```
   export CLIOPSPT=1955
   ```

   If you are using the C shell, use:

   ```
   setenv CLIOPSPT 1955
   ```

   Multiple tape mount daemons can use the same PARSRV.

   If PARSRV is registered in /**etc**/**services** on AIX or in **TCPIP.profile.TCPIP** on MVS, you must set the CLIOPSID environment variable to the name corresponding to the server/port combination you want to use.
5. Use SMIT to "remove" the tape pseudo device driver:

   ```
   smit tape
   ```
6. On the Tape Driver panel, select "Remove a Tape Drive."
7. Select your tape drive from the Single Selection List that is displayed.
8. On the Remove a Tape Drive panel, set the "KEEP definition in database" field to "no".
9. Press Enter or select "Do" to process.

To create the device after you have removed it, follow the procedures in "Configuring and Starting the CLIO/S Tape Mount Daemon" on page 45.

Administrative Commands

cltclntd—Starting the CLIO/S Tape Mount Daemon

Syntax

cltclntd –**f** *device*

Parameters

–**f** *device* Specifies the tape device. For both AIX and MVS, this is the pseudo device associated with the desired physical device (for example, /**dev**/**pst0**).

Description

The **cltclntd** command starts the tape mount daemon. The **cltclntd** command is automatically issued by SMIT when you configure the tape pseudo device driver. The tape pseudo device driver signals the tape mount daemon whenever one of these supported I/O action occurs. The tape pseudo device driver does this by issuing the appropriate application programming interface function as follows:

Command	Tape API function
allocate	**ctpalloc()**
open	**ctpopen()**
read	**ctpread()**
write	**ctpwrite()**
close	**ctpclose()**
unallocate	**ctpunall()**

In addition, the tape pseudo device driver passes requests to the application programming interface to obtain tape device parameters defined in the ODM database and to service the following **ioctl()** requests:

STFSF	Forward space by file
STFSR	Forward space by record
STRETEN	Retension the tape, leaving at load point
STREW	Rewind the tape
STRSF	Reverse space by file
STRSR	Reverse space by record
STWEOF	Write an end-of-file record

Usage Notes

The tape mount daemon is automatically restarted by the Configuration Manager when the system is rebooted, using the configuration files /**etc**/**parconf** and /**.netrc**.

Security Restrictions

You must have **root** authority to use this command.

Messages and Responses

Along with debugging messages, the daemon writes the following messages to stderr upon completion:

- `cltclntd: /dev/pst`*n*` unmounted cleanly`
- `cltclntd: Error - can only be run by root`
- `cltclntd: Host type UNKNOWN for host `*node*

  The PARCONF file contains an invalid host type.
- `cltclntd: Usage: cltclntd -f device.`

The following messages are placed in the tape mount daemon log in /**tmp**/**cltclntd.pst***n*:

- `cltclntd: Can't open device /dev/mst`*n*

  This message is followed by one describing the reason for the failure to open.

- `cltclntd: Terminating abnormally due to signal` *signal*

  A signal sent to the tape mount daemon caused it to terminate.

Files
/tmp/cltclntd.pst*n* Log of tape mount daemon activity

Related Topics
Commands: **cltterm**

cltterm—Stopping the CLIO/S Tape Mount Daemon

Syntax
cltterm –**f** *device*

Parameters
–**f** *device* Specifies the tape device associated with this daemon's occurrance (for example, /**dev/pst0**).

Description
This command signals the pseudo device driver to terminate the local tape client, **cltclntd** daemon.

Security Restrictions
You must have **root** authority to use this command.

Messages and Responses
This command displays one of the following messages upon completion:

- `cltterm: daemon on /dev/pst`*n* `is now terminated`
- `cltterm: Error - can only be run by root`
- `cltterm: Usage: cltterm -f device`
- `cltterm: openx error for dev/pst1, errno=22 invalid argument`
- `cltterm: openx error for dev/pst0, errno=73, correction reset by peer`

Related Topics
Commands: **cltclntd**

Part 2. The CLIO/S Application Linker

© Copyright IBM Corp. 1993, 1995, 1996

Chapter 5. The Application Linker Graphical User Interface

The Application Linker graphical user interface (GUI) that lets you build a distributed pipe environment to rapidly transfer data between:

- AIX systems
- MVS systems
- AIX and MVS systems.

The Graphical User Interface is an object-oriented interface, consisting of application, pipe link, and pipe objects. Using the Application Linker, you can create and connect these objects to perform a particular task, such as splitting a DB2/MVS database and processing its data on one or more SP nodes.

The GUI makes it easy to define applications and their pipe connections because all components – applications, pipes, and pipe links – are represented visually and can be manipulated using the graphical user interface.

The Application Linker GUI also gives you centralized control of your distributed pipe environment because you can build, run, and monitor multiple applications, pipes, and pipe links through a single window.

This chapter covers the following topics:

- "Why Use the Application Linker?"
- "Learning to Use the Application Linker" on page 58
- "Terminology and Concepts" on page 58
- "About the Application Linker GUI" on page 63
- "Using the Graphical User Interface" on page 66
- "Sample Configuration Scenarios" on page 67
- "Changing a Configuration" on page 74
- "Running Large Application Linker Configurations" on page 79

Why Use the Application Linker?

The Application Linker GUI gives you centralized control of your distributed pipe environment by allowing you to manipulate an entire configuration as a single unit. You can also build, run, monitor, and save multiple configurations. For example, you can save a configuration in a file and then reload that configuration file in a subsequent Application Linker session. You can also start a configuration file in a session outside of the GUI and have the results logged in a file.

When you use CLIO/S without the Application Linker, CLIO/S does not check that your data flow is correctly defined. Parameters are not checked until the CLIO/S commands are executed.

The Application Linker, however, validates many CLIO/S pipe and pipe link parameters when you create the objects. It also checks the completeness and consistency of the data flow connections.

© Copyright IBM Corp. 1993, 1995, 1996

Learning to Use the Application Linker

Before you use the Application Linker, you should be familiar with:

- The CLIO/S product in general
- Chapter 8, “The CLIO/S Cross System Pipe Link” on page 141
- “Creating a CLIO/S Pipe With clpmk” on page 145
- “Connecting to a CLIO/S Pipe With CLPLINK” on page 147
- The information in this chapter.

In addition, the online help system, accessible through the GUI, provides detailed explanations of the GUI functions and the tasks you'll be performing.

This chapter explains the concepts you need to know before you run the Application Linker graphical user interface (GUI) and presents sample scenarios for building configurations. Refer to the online help system for details on performing the tasks mentioned to here, such as saving a configuration and starting a configuration.

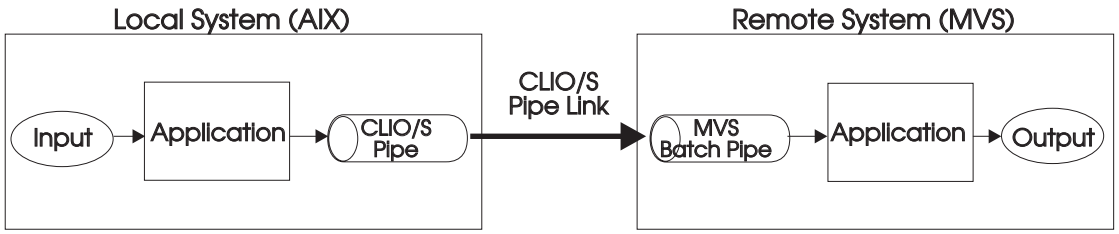


Figure 9. Sample CLIO/S Application Linker configuration

Figure 9 shows a typical CLIO/S Application Linker configuration. On AIX, an application writes to a CLIO/S pipe, which is connected to a CLIO/S pipe link. The CLIO/S pipe link is also connected to a BatchPipe on MVS, which is read as input by an MVS application.

Terminology and Concepts

This section explains the terminology and concepts you need to understand before you use the Application Linker GUI. Note that in this chapter, terms such as CLIO/S pipe and CLIO/S pipe link have meanings specific to their use with the Application Linker.

In the Application Linker GUI, configuration components (applications, pipes, and pipe links) are referred to as objects.

Data objects represent anything you read data from or write data to other than a CLIO/S pipe. These include files, data sets, MVS BatchPipes, devices, volumes, and AIX named pipes that are created by your application rather than the Application Linker.

Applications

An application is anything that can be executed on either MVS or AIX. On MVS, applications are jobs. On AIX, applications are commands, shell scripts, and executable files. An application does one or more of the following:

- Runs stand-alone on a local or remote system.
- Produces output that is piped to a local or remote system.
- Receives as input data that is piped from a local or remote system.
- Processes multiple inputs, outputs, or both.

Applications can have multiple inputs and outputs but can only read from and write to CLIO/S pipes and other data objects.

Note

The Application Linker cannot run an application that reads from the terminal.

CLIO/S Pipes

A CLIO/S pipe is an AIX named pipe created by the **clpmk** command. Only applications and pipe links can read from or write to a CLIO/S pipe.

CLIO/S Pipe Links

A CLIO/S pipe link transfers data from one CLIO/S pipe or data object to another, generally on different hosts.

A pipe link consists of two components, each running on a different host, that work together to transfer the data between the two hosts.

The data transfer is uni-directional and the pipe link can have only one input and one output.

Data Objects

Data objects such as files, data sets, MVS BatchPipes, devices, volumes, and AIX named pipes are created by your application rather than the Application Linker. You can specify data objects as inputs to or outputs of both pipe links and applications. Data objects can be input to and output from multiple applications or links.

The actual data objects are not managed by the Application Linker

Configuration

A configuration is the group of application, pipe, and pipe link objects that perform a particular data transfer task, such as splitting a DB2/MVS data base and loading its contents on one or more SP nodes.

In general, a configuration consists of multiple objects, where the output of one object is the input to another object. However, each object can only accept input from and deliver output to certain types of objects. For example, an application can only write to a CLIO/S pipe or to a data object. An application cannot write directly to a pipe link.

The Configuration File

When you use the Application Linker GUI to define and save the application, pipe, and pipe link objects associated with a particular data transfer task, all the information associated with your configuration is automatically saved in a **configuration file**. This information includes the nodes, user IDs, and passwords necessary to run the configuration. You can load this saved configuration in a future Application Linker session. Once you have created a configuration file, you can run it either using the GUI or without using the GUI. The **clalink** command offers options for both.

Note

Do not attempt to manually edit a configuration file, except to change a password.

By default, a configuration file name is in the form *filename***.config**.

A configuration file contains records for all the information associated with the configuration. The first word (or phrase) on a line indicates the record type:

- **CLA Version** indicates the level of the Application Linker which generated the configuration file. The version levels of the Application Linker and the configuration file must be compatible.
- **clplink** indicates a pipe link record.
- **clpmk** indicates a CLIO/S pipe record.
- **Regular Pipe** indicates a non-CLIO/S pipe record.
- **File/Dataset** indicates a file or MVS dataset record.
- **Device** indicates a device record.
- **Raw Volume** indicates a raw volume record.
- Any other notation is considered an application record.

All except the Version Record have a user id, node, password, ID, and input and output lists.

The record syntax for pipe links and pipes are the same as the **clplink** and **clpmk** commands, respectively, with the addition of the fields listed above. The "other data" records is the name of the data object.

The only record delimiter in a configuration file is the new line character. Anything following a pound sign (#) is ignored.

Partial Saves and Loads

It is possible for you to save a configuration that is not complete. For example, if you have begun building a configuration but have not finished, you can save the work you have done so far. When you save or start an incomplete configuration, the Application Linker's validation program will display a series of messages. These are merely warnings which tell you that you have an incomplete configuration. You can choose to ignore these warnings. When you reload the configuration, you will be reminded that it is incomplete.

Node Hopping

If you are not familiar with the concept of node hopping, see “Node Hopping with PARCONF” on page 28.

The host name can take the form of a node name, an IP address, a chain of node names or IP addresses separated by colons (for instance, node1:node2:node3), or a PARCONF nickname. A PARCONF nickname is a unique name representing the node where the application is to run (the executing host) or representing a chain of nodes through which the executing node is reached.

When you use node hopping, you specify the intermediate userid/password combinations either of the following ways:

- By explicitly specifying a password and user ID for each intermediate node.
- By having the Application Linker use the passwords and user IDs found in **.netrc** files. The Application Linker begins with the **.netrc** file of the userid under which the Application Linker is running and searches there for the userid and password for the next node in the chain. The userid on each node must have in its **.netrc** file the userid and password for the next node. Note that the Application Linker uses the That is, to reach Node C from Node A via intermediate Node B, the userid and password for Node B must be in the **.netrc** file on Node A and the userid and password for Node C must be in the **.netrc** on Node B.

System Authorization

You must have access to the systems where you plan to move data to or from. For example, suppose you create an Application object to be run on an MVS system, and you define a configuration that moves data from that MVS system to an AIX system. You need to have a valid user ID and password for both the MVS system and the AIX system. If you do not explicitly specify a user ID and password, the Application Linker searches the **.netrc** file from the home directory to find a user ID and password for the specified host. If a correct user ID and password cannot be found, the associated object (application, link, or pipe) will not start, or you will be prompted to supply a password at the command line. This can appear to be a "hang" in the application but is really a wait for input.

Password Encryption

When the Application Linker saves a configuration, for any object that has an explicit password – that is, a password is not to be extracted from your **.netrc** file – you are permitted to use an encryption key to encrypt the password. When you encrypt an object's password, the password appears as an unintelligible character string in the configuration file. When you load the saved configuration, you must enter this same encryption key in order to decrypt the passwords.

If You Forget the Encryption Key

If you forget the encryption key, you must edit the configuration file and remove the encrypted password. In the configuration file, you can do either of the following:

- Replace “-epassword *encrypted password*” with “-password *unencrypted password*”
- Replace “-epassword *encrypted password*” with “-password .” and replace “-user *userid*” with “-user .” This causes the Application Linker to get the user

ID and password from the **.netrc** file of the user ID which is running the Application Linker GUI.

Object IDs

Each object managed by the Application Linker is identified by a unique object ID. Two pipes on different systems can have the same name, but they will each have a unique ID. For instance, `pipe0001`. The Application Linker automatically generates these IDs. They are displayed in the icon titles in the GUI and on the title bar of the object's notebook.

Object IDs have prefixes that identify the type of object:

app Indicates an application

link Indicates a pipe link

pipe Indicates a CLIO/S pipe

Note that an object's ID can change when you save and then reload a configuration.

Start After IDs

You can defer starting some activities until after some other activities have completed by specifying Start After IDs. For an application or a pipe link, you can specify a list of object IDs (separated by colons), such that the application or pipe link will not start until all of the objects listed have completed. For example, suppose you have a validation routine that checks whether a data set was transferred successfully. You can specify that the validation program is to start after the transfer through the pipe link has completed.

You can specify Start After IDs only to defer starting an application or a pipe link. However, you can specify applications, pipes, and pipe links as objects to start after.

Model Validation

When you save a configuration, the Application Linker first attempts to verify that it has been defined correctly and can run successfully. This is referred to as *model validation* If the Application Linker finds errors in the way you have defined the configuration or any component objects, a message box appears with one or more informational or error messages.

If model validation errors occur, you may continue anyway, but you should correct the errors before starting the configuration. the configuration. When you open a configuration file that was saved with validation errors, will be informed of them.

The Application Linker performs the following checks:

All objects The user ID and password for the object's host system must be specified or accessible through the appropriate **.netrc** file.

Application objects All inputs and outputs must be on the same host as the application object. Any inputs or outputs that are CLIO/S pipes must run under the same user ID and password as the application object. Inputs and outputs

	are not checked for consistency with command line parameters.
Pipe link objects	The direction of the pipe link must be consistent with the direction of the pipes attached to it. Local and remote site options must be valid. But, site options that are operating system specific are not checked. The pipe link must have an input and an output.
Pipe objects	The pipe must have both an input and an output, and the pipe's mode (read or write) must be compatible with its direction and placement in relation to connecting pipe link and application objects.

About the Application Linker GUI

The Graphical User Interface consists of application, pipe link, and pipe objects, and actions that you can perform on them. The components of the Application Linker configuration applications, pipe links, and pipes), and the Application Linker configuration itself are the objects. Component objects are represented either as icons or as entries in a list, depending on how you choose to display the configuration. The actions you can perform and the ways you can perform them depend on the type of object.

The Application Linker GUI consists of the following elements:

- The Primary Window
- Notebooks
- Dialog boxes
- Message Boxes

The Primary Window

Figure 10 on page 64 show the Application Linker GUI's primary window.

The Application Linker GUI's primary window includes:

- **The Menu Bar** contains pulldown menus for performing actions on the entire Application Linker configuration or on individual objects, as well as for tailoring features of the primary window.
- **The Tool Bar** contains icons representing frequently-performed tasks.
- **The Template Area** contains templates of Application Linker objects for adding application, pipe link, and pipe objects to a configuration.
- **The Work Area** displays the configuration in either of two views: **Icon View** or **List View**.
- **The Information Area** displays a brief description of the window area, menu option, or item at the current cursor location. Whenever the cursor is over an object icon in the Work Area, the object's name and host are displayed.

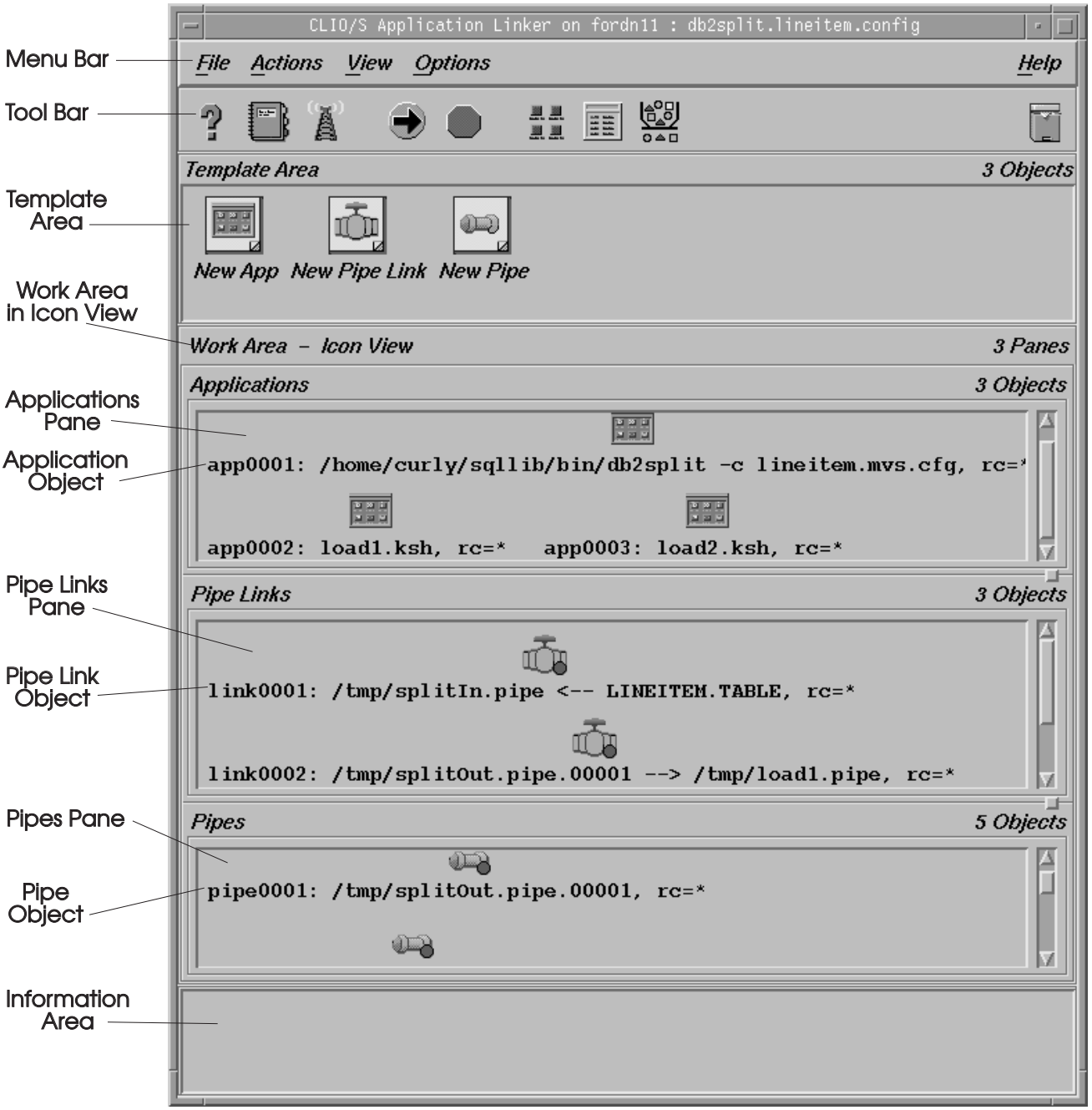


Figure 10. The Application Linker GUI primary window

The Menu Bar

The Menu Bar contains pulldown menus for performing actions on the entire Application Linker configuration or on individual objects, as well as for tailoring features of the primary window.

The Tool Bar

The Tool Bar, shown in Figure 11 on page 65,
contains icons representing the following frequently-performed tasks:

Help displays context-sensitive help for a specific object, action, or window area.

Notebook displays the notebook for an application, pipe link, or pipe object.

Signal signals an application or pipe link object.

Start Configuration starts all objects in the configuration.

Stop Configuration stops all objects in the configuration.

Icon View displays Work Area objects in Icon View.

List View displays Work Area objects in List View.

Sort/filter sets sort criteria to change the display order of Work Area objects and/or sets filtering criteria to include/exclude objects Work Area objects.

Delete deletes an object from the configuration.

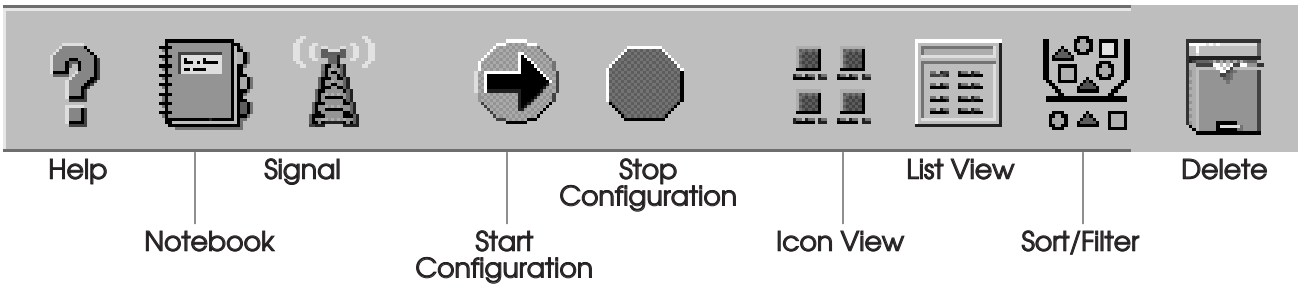


Figure 11. The Application Linker GUI Tool Bar

The Template Area

The Template Area contains templates of Application Linker objects for adding application, pipe link, and pipe objects to a configuration.

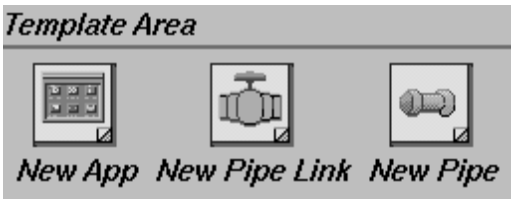


Figure 12. The Application Linker GUI Templates

To create an object, click on an object template. An object creation dialog, in the form of a notebook, is then displayed to allow you to define the new object. The templates are:

New App Creates a new application object.

New Pipe Link Creates a new pipe link object.

New Pipe creates a new pipe object.

The Work Area

The Work Area allows you to view the configuration in either of two ways: Icon View or List View.

Viewing a Configuration in Icon View: The icon view of the Work Area displays objects as icons, with each type of object in a separate pane. The form of an icon indicates the object's state — running, stopped, waiting, paused, or getting output from MVS.

Each pane displays the total number of objects of that type, including any hidden by a filter. If applicable, a separate count of the hidden objects is also displayed.

Viewing a Configuration in List View: The list view of the Work Area shows objects or combinations of objects as entries in a scrollable list. The pane displays the total number of objects in the configuration, including any hidden by a filter. If applicable, a separate count of the hidden objects is also displayed.

Objects connected to form a data path are presented as a single entry, showing the path from data source to data target, or as much of the path as is connected. An application connected to a pipe, for instance, is incorporated into the entry for the pipe, appearing as either an input or output application.

The Information Area

The Information Area displays a brief description of the window area, menu option, or item at the current cursor location, If you place the cursor over an object icon in the Work Area, the object's name and host are displayed.

Notebooks

Each configuration object has a notebook containing information about the object and its status. Related attributes and information are grouped on a page, with a labeled “tab” identifying the general category of information.

Dialog Boxes

When you use the Graphical User Interface for certain tasks, such as sort/filter, a dialog box is presented to solicit additional information needed to complete the action. The specific information you need to enter in the dialog box depends on the task you are performing.

Message Boxes

Graphical User Interface message boxes identify the box as a **clalink** message. The messages contain a message identifier, message text, and push buttons to respond to the message.

Using the Graphical User Interface

This topic explains how to customize and bring up the Application Linker GUI, and how to run the Application Linker without the GUI.

Customizing the Appearance of the GUI

The X resources for the CLIO/S Application Linker reside in **/usr/lpp/fcf/app-defaults/Clalink**. If you want to customize these X resources, do one of the following:

- Add and tailor the specific X resources in your **.Xdefaults** file.
- Specify an X resource with one of the standard command line functions. For instance, "-fg red" for red foreground color.
- Copy **/usr/lpp/fcf/app-defaults/Clalink** into your application defaults directory (specified by the XAPPLRESIDR environment variable) and tailor the X resources.

Invoking the CLIO/S Application Linker

The **clalink** command provides a subset of the function available from the GUI in command line form. You can run the Application Linker by entering the **clalink** command.

When you issue the **clalink** command to bring up the CLIO/S Application Linker GUI, a splash dialog first appears. This dialog displays status messages regarding window initialization, Pipe Monitor subsystem startup, and configuration processing.

clalink—Starting the CLIO/S Application Linker

Syntax

clalink [**–h** | **[–f** *config_file* **] [–k** *crypt_key***] [–l [** *log_file***]] [–N]**

Parameters

–h Displays help information.

–f *config_file*
Specifies the configuration file to load. *config_file* contains a configuration saved from a previous invocation of clalink.

–k *cryp_key* Specifies the encryption/decryption key for passwords saved in the configuration file.

The user is prompted for the encryption/decryption key if the **–k** parameter is not specified and **–f** *config_file* parameter is specified for a configuration file that contains previously encrypted passwords.

–l [*log_file*] Specifies to enable logging.

Log messages are written to *log_file* if it is specified. Otherwise, log messages are written to **clalink.log**.*process_id*, where *process_id* is the corresponding clalink process id.

–N Indicates not to use the CLIO/S Application Linker GUI. Use this option to run the Application Linker with a configuration file previously created using the GUI.

Description

Invokes the CLIO/S Application Linker.

Messages and Responses

See the Application Linker Messages section of the Online Help.

Related Topics

Commands: **clpmk, clplink**

Sample Configuration Scenarios

This section shows you how to set up two sample Application Linker configurations: one simple and one complex.

Simple Application Linker Scenario

This sample configuration transfers the CLIO/S REXX exec from the MVS dataset **'SYS1.SFCFSAMP(FCFREXX)'** to a CLIO/S pipe. This CLIO/S pipe provides the input for a **cat** command to display the contents of FCFREXX.

Note that this configuration file is shipped with the CLIO/S product as **/usr/lpp/fcf/sample/simple.config**.

To create this configuration, do the following:

1. Enter **clalink** to start the Application Linker.
2. Create a pipe link object:
   a. In the primary window of the GUI, select the Actions pulldown. Then select Pipe Links⇒Create to display the Pipe Link Creation Notebook.
   b. In the Local Data field, specify /**tmp**/**pipe1** as a CLIO/S pipe.
   c. In the Local Host field, specify the hostname of the AIX client to which you are downloading.
   d. In the Remote Data field, specify **'SYS1.SFCFSAMP(FCFREXX)'**
   e. In the Remote Data area, click on the selection menu and select File/Dataset.
   f. In the Remote Host field, specify the name of the MVS system from which you are downloading. Specify the Remote user id and password if they are not in your *.netrc* file.
   g. Click on the Direction arrow so that the data flow is from Remote Host to Local Host.
   h. Select a Data Type of ASCII so that the MVS EBCDIC characters will be converted to ASCII.
   i. Click the OK (or Apply) button on the bottom of the Pipe Link Creation Notebook window.
3. Create an application object:
   a. In the primary window of the GUI, select the Actions pulldown. Then select Applications⇒Create to display the Application Creation Notebook.
   b. In the Application Name field, specify **cat** /**tmp**/**pipe1**.
   c. Select the I/Os page. Then do the following:
      1) Under Current Inputs, specify /**tmp**/**pipe1** as a CLIO/S pipe.
      2) Click the Add button to add this input.
      3) Click the OK (or Apply) button at the bottom of the Application Creation Notebook window.

         Note that the Application Linker automatically creates a pipe object when you specify a pipe as I/O to the application.
   d. Start the configuration by clicking the Start Configuration icon from the toolbar. Figure 13 on page 70 shows the way the configuration looks in icon view when it is running.
   e. You can use the GUI to monitor the status of the configuration in the following ways:

- In the Icon view, you can view object states and return codes.
- In the List view, you can object states and data transfer information.
- In each object's notebook, you can view object status and messages.

4. When the configuration has completed, double-click on the icon for the application object to display the Application Status Notebook. Select the messages page to view the output of the **cat** command.

The running of the configuration is considered completed when all objects are in a Stopped state.

Note that the Application Linker automatically creates all pipe objects through the I/O entries in the Create Application notebook and the Local/Remote data entries in the Create Pipe Link notebook.

Complex Application Linker Scenario

This sample configuration downloads a DB2/MVS database table from MVS to an RS/6000 SP, and then splits and loads the table to various nodes on the SP system.

Note that this configuration file is shipped with the CLIO/S product as **/usr/lpp/fcf/sample/db2split.lineitem.config**.

Figure 14 on page 71 shows the configuration you will be creating in this scenario.

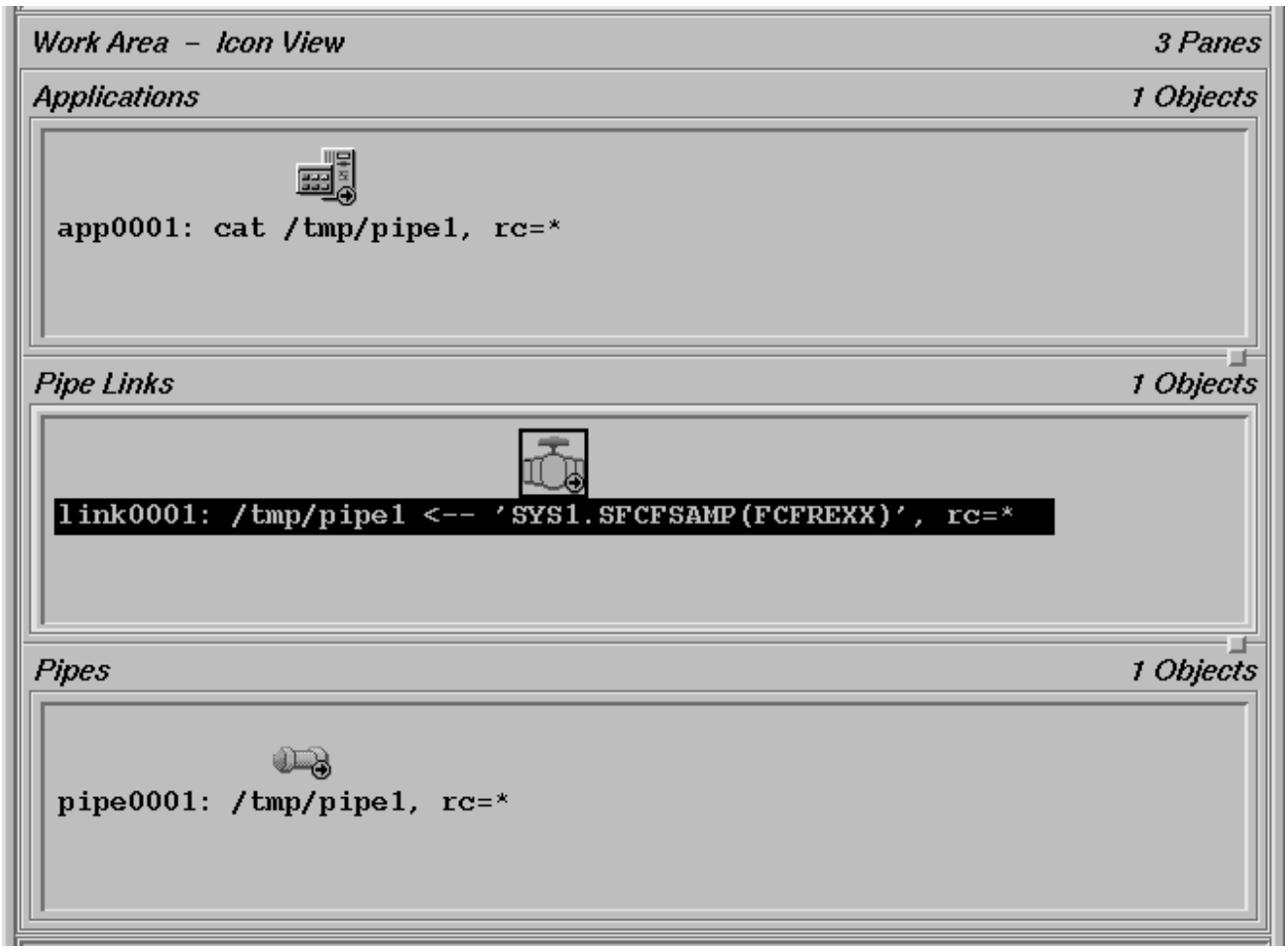


Figure 13. The simple Application Linker configuration in icon view

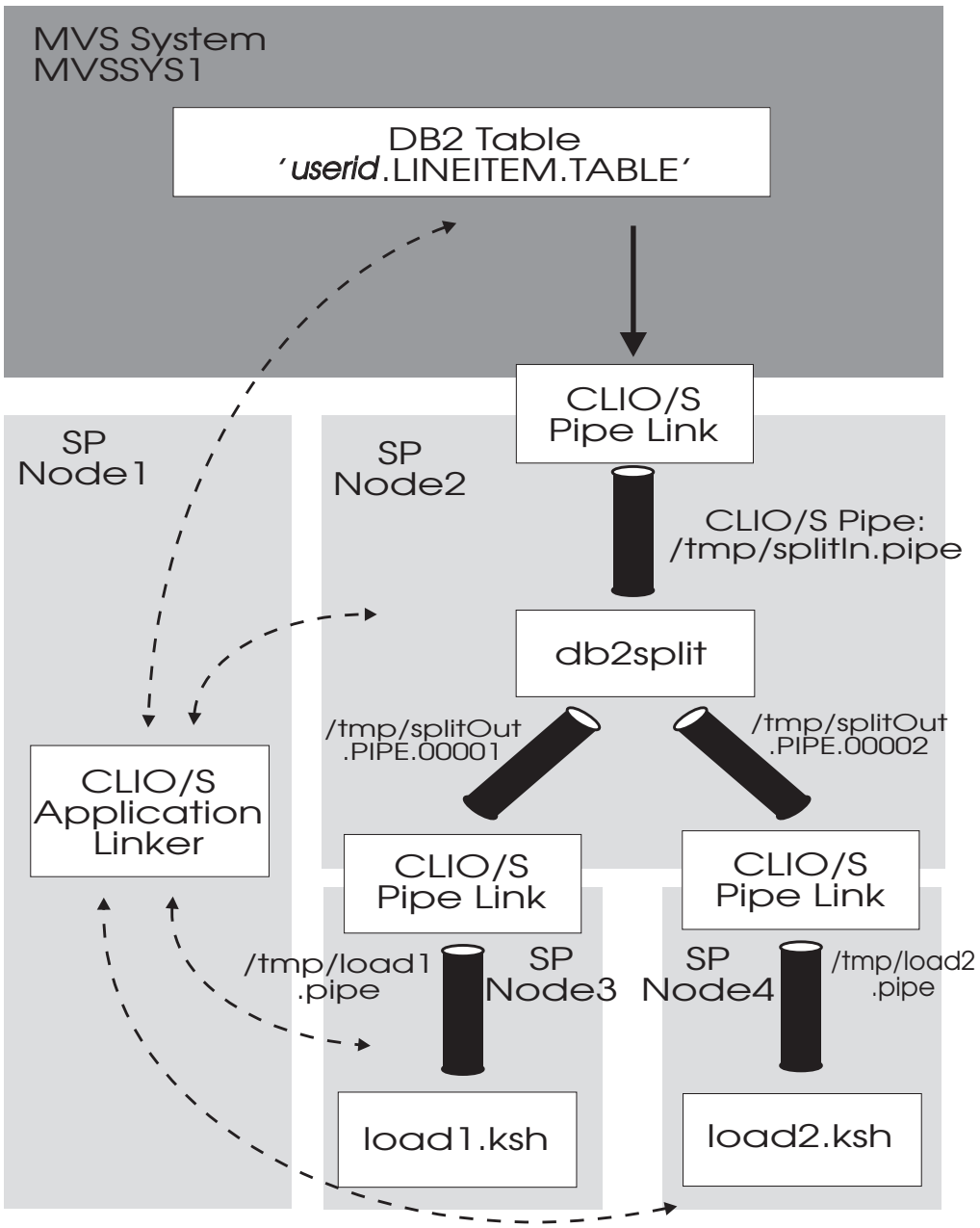


Figure 14. Downloading a DB2 Database Table onto Two SP Nodes

To create this configuration, do the following:

1. Enter **clalink** to start the Application Linker.
2. Create an application object for Node 2 called **db2split**. This application invokes the command **/home**/*user*/**sqllib**/**bin**/**db2split -c lineitem.cfg**.
   a. In the primary window of the GUI, select the Actions pulldown. Then select Applications⇒Create to display the Application Creation Notebook.
   b. In the Application Name field, specify **/home**/*user*/**sqllib**/**bin**/**db2split -c lineitem.cfg**.
   c. In the Host field, specify **Node2** (the name of the host where the command will run).

d. Select the I/Os page. Then do the following:

1) Under Current Inputs, specify /**tmp**/**splitIn.pipe** as a CLIO/S pipe. Note that this name must match the name you specify on the InFile parameter in your **db2split** configuration file. For more information, see *DB2 Parallel Edition for AIX Administration Guide and Reference* , SC09-1982.

2) Click the Add button to add this input.

3) Under Current Output, specify /**tmp**/**splitOut.pipe.00001** as a CLIO/S pipe.

4) Select the Add button to add this output.

5) Under Current Output, specify /**tmp**/**splitOut.pipe.00002** as a CLIO/S pipe.

6) Click the Add button to add this output.

7) Click the OK (or Apply) button at the bottom of the Application Creation Notebook window.

3. Create an application object for Node 3 called **load1.ksh** to load the generated split file from **db2split**.

a. In the primary window of the GUI, select the Actions pulldown. Then select Applications⇒Create to display the Application Creation Notebook.

b. In the Application Name field, specify **load1.ksh**.

c. In the Host field, specify **Node3** (the name of the host where **load1.ksh** will run).

d. Select the I/Os page. Then do the following:

1) Under Current Inputs, specify /**tmp/load1.pipe** as a CLIO/S pipe.

2) Click the Add button to add this input.

3) Click the OK (or Apply) button at the bottom of the Application Creation Notebook window.

4. Create another application object for Node 4 called **load2.ksh** to load the generated split file from **db2split**.

a. In the primary window of the GUI, select the Actions pulldown. Then select Applications⇒Create to display the Application Creation Notebook.

b. In the Application Name field, specify **load2.ksh**.

c. In the Host field, specify **Node4** (the name of the host where **load2.ksh** will run).

d. Select the I/Os page. Then do the following:

1) Under Current Inputs, specify /**tmp/load2.pipe** as a CLIO/S pipe.

2) Click the Add button to add this input.

3) Click the OK (or Apply) button at the bottom of the Application Creation Notebook window.

5. Create a pipe link object for transferring the table from MVS to the **db2split** command.

a. In the primary window of the GUI, select the Actions pulldown. Then select Pipe Links⇒Create to display the Pipe Link Creation Notebook.

b. In the Local Data field, specify /**tmp**/**splitIn.pipe** as a CLIO/S pipe

c. In the Local Host field, specify **Node2**.

d. In the Remote Data area, click on the selection menu and select File/Dataset.

e. In the Remote Data area, specify **'***userid***.LINEITEM.TABLE'**.

f. In the Remote Host field, specify **MVSSYS1**.

g. Click on the Direction arrow so that the data flow is from Remote Host to Local Host.

h. Select a Data Type of ASCII.

i. Select Convert Nulls to Blanks if this conversion is necessary.

j. Click the OK (or Apply) button at the bottom of the Application Creation Notebook window.

6. Create a pipe link object for transferring the split data for Node 3 to the **load1.ksh** script.

   a. In the primary window of the GUI, select the Actions pulldown. Then select Pipe Links⇒Create to display the Pipe Link Creation Notebook.

   b. In the Local Data field, specify /**tmp**/**splitOut.pipe.00001** as a CLIO/S pipe.

   c. In the Local Host field, specify **Node2**.

   d. In the Remote Data field, specify /**tmp**/**load1.pipe** as a CLIO/S pipe.

   e. In the Remote Host field, specify **Node3**.

   f. Click the OK (or Apply) button at the bottom of the Pipe Link Creation Notebook window.

7. Create another pipe link object for transferring the split data for Node 4 to the **load2.ksh** script.

   a. In the primary window of the GUI, select the Actions pulldown. Then select Pipe Links⇒Create to display the Pipe Link Creation Notebook.

   b. In the Local Data field, specify /**tmp**/**splitOut.pipe.00002** as a CLIO/S pipe.

   c. In the Local Host field, specify **Node2**.

   d. In the Remote Data field, specify /**tmp**/**load2.pipe** as a CLIO/S pipe.

   e. In the Remote Host field, specify **Node4**.

   f. Click the OK (or Apply) button at the bottom of the Pipe Link Creation Notebook window.

   Figure 15 on page 75 shows the way the Work Area should look with the configuration you have created.

8. Start the configuration by clicking the Start Configuration icon from the toolbar.

9. You can use the GUI to monitor the status of the configuration in the following ways:

   - In the Icon view, you can view object status and return codes.

- In the List view, you can view Status and data transfer information. Figure 16 on page 76 shows the Work Area in list view once the configuration has stopped running. Note that the right-most part of the screen is not shown. You must scroll to the right to see it.
- In the Object Notebook, you can view object status and messages.

The running of the configuration is considered complete when all objects are in a Stopped state.

Note that the Application Linker automatically creates all pipe objects through the I/Os entries in the Create Application notebook and the Local/Remote data entries in the Create Pipe Link notebook.

Changing a Configuration

This section describes the things you need to know before you attempt to change a saved configuration.

Planning Considerations

You can change or delete an object only when it is not running and the change will not affect any running objects.

If you plan to move an application from one host to another, you need to consider what to do with the application's inputs and outputs. Because an application can only read or write from a pipe that is on the same host, moving the application also moves its I/O. Similarly, moving a pipe to a new host would require any application to which it is connected to move, as well as moving all of the application's other I/O. Such pervasive changes are difficult to un-do and are not done automatically by the Application Linker.

If the input or output is a CLIO/S pipe, the Application Linker automatically moves the pipe when you move the application object. For objects other than CLIO/S pipes, you must ensure that the actual objects (files, tapes, etc.) are residing or will reside on the new host.

Whenever you make more than one change to an object, all of the changes are applied simultaneously when you click either OK or Apply. For example, changing Pipe A on Host X to Pipe B on Host Y is legal even if there is a Pipe A on Host Y and a Pipe B on Host X. As long as there is no Pipe B on Host Y, the change is performed. If there is already a Pipe B on Host Y, the pipe is **NOT** left as Pipe B on Host X or Pipe A on Host Y. If one of the changes is not valid, no changes are made.

Changing an Application

The following are considerations and restrictions for changing an application.

Deleting an Application

You can always delete an application. Any CLIO/S pipes or data objects that are connected to inputs or outputs are disconnected but not deleted. If deleting an application leaves a data object unused, that object is not stored when you save the configuration.

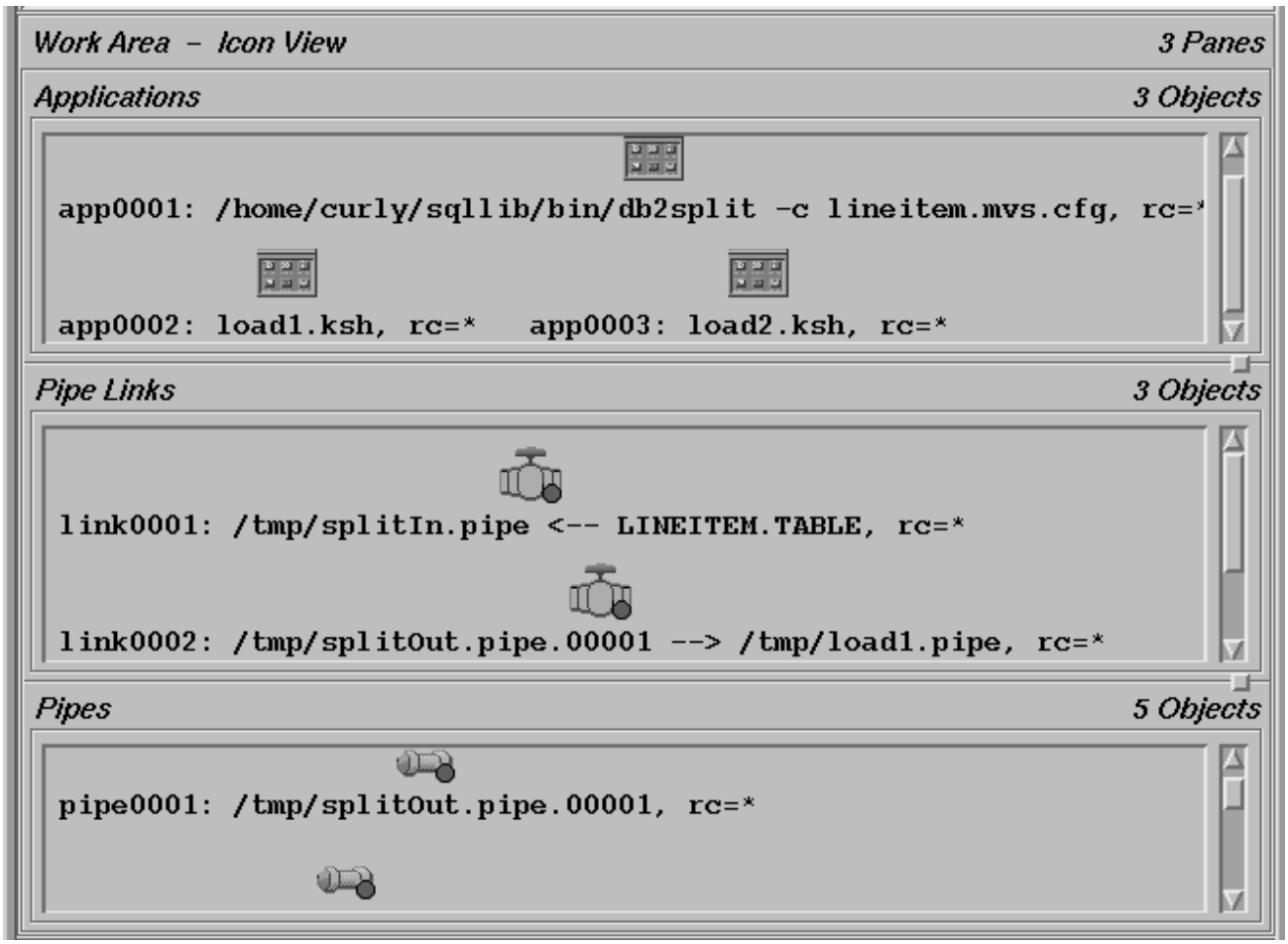


Figure 15. The complex Application Linker configuration in icon view

Changing the Name and the Parameters

You can change the name of an application (the command which is executed) and any parameters as long as the application is not running. If a parameter specifies a new input or output to the application, you must also update the application's I/O page to identify the new input or output.

Changing the Host Machine

You can change an application's host machine provided the application is not running and the Application Linker is able to change the hosts of any inputs and outputs that are connected to the application. Because the Application Linker automatically moves the application's inputs and outputs to the new host, the application will not be moved if a CLIO/S pipe with the same name as the pipe being moved already exists on the new host.

Changing the User ID and Password

You can change an application's user ID or password as long as the application is not running. The Application Linker automatically updates the user ID or password of any CLIO/S pipes connected to the application object to match new user ID or password.

Figure 16. The complex Application Linker configuration in list view

Changing Inputs and Outputs

You can delete an application's inputs and outputs provided the application is not running. If a CLIO/S pipe becomes unused as a result of deleting an input or an output, the pipe is not deleted. If a data object becomes unused as a result of deleting an input or an output, the object is deleted when you save the configuration.

You can also add inputs and outputs:

- When you can add an existing CLIO/S pipe as a new input or output, make sure the mode of the pipe is as follows:
  - Input pipes must be in read mode
  - Output pipes must be in write mode

  Also, the pipe must not already be in use by another application.
- You can add data objects (such as files and data sets).
- You can define a new CLIO/S pipe, regular pipe, file, data set, device, or raw volume as an input or an output.

Changing a CLIO/S pipe

The following are consideration and restrictions for changing a CLIO/S pipe.

Deleting a CLIO/S Pipe

You can always delete a CLIO/S pipe. If you delete a CLIO/S pipe that is attached to a pipe link, the Application Linker sets the local or remote host name (whichever applies) to null instead of deleting the link. To indicate that a pipe link has null local/remote data, the Application Linker identifies the pipe link using its assigned object ID but partial or no name. If you try to start the pipe link or save the configuration with the pipe link in this state, an error message will appear.

Changing the Pipe Name

You can change the name of a CLIO/S pipe provided there is not already a pipe with that name on the same host. If the pipe you change is connected to a pipe link, the Application Linker automatically changes the local or remote pipe link host name (whichever applies).

Changing the Host Name

You can change the host name of a CLIO/S pipe provided there is not already a pipe with that name on the same host, and provided the pipe is not connected to an application. If the pipe you change is connected to a pipe link, the Application Linker automatically changes the local or remote pipe link host name (whichever applies).

Changing the User ID and Password

You can change the user ID or password of a CLIO/S pipe provided the pipe is not connected to an application. If the pipe you change is connected to a pipe link, the Application Linker automatically changes the local or remote pipe link host name (whichever applies).

Changing the Direction (Mode)

You can change the direction or mode (read or write) of a pipe provided the pipe is not connected to an application. If a pipe is connected to only a pipe link, you can change the mode only if the link is not connected to a pipe that cannot change direction. Changing the direction of a pipe that is connected to an application changes the direction of the pipe link and the other pipe – if it exists – connected to the pipe link.

Changing a Pipe Link

The following are consideration and restrictions for changing a CLIO/S pipe link.

Deleting a CLIO/S Pipe Link

You can always delete a CLIO/S pipe link. If a link is attached to a pipe link and you delete the pipe link, the pipe becomes unattached on that side but is not deleted. If a pipe link is attached to a data object and you delete the pipe link, the data object becomes unused and is deleted when you save the configuration.

Changing the Local or Remote Data Name

You can change the local or remote name of a pipe link to a name that does not already exist. This causes the name of the associated pipe, if there is one, to change.

You can change the local or remote name of a pipe link to a name that does exist provided the existing object is either a data object or is a pipe that is not already used as an input or an output. If a pipe link was previously attached to a pipe or a data object, that connection is broken and replaced by the new pipe or the data object.

Changing the Local or Remote Host

You can change the local or remote host of a pipe link if the pipe link is connected to a data object on the side that is changing.

Also, you can change the local or remote host of a pipe link if there is no pipe on the new host with the same name, or if there is no existing pipe on the new host which is usable by the pipe link. If the existing pipe is also used by an application, the pipe is disconnected from the pipe link and a new pipe (or the existing pipe on the new host) is connected to the pipe link. If the existing pipe is not connected to an application, the host of the pipe is changed.

Changing the Local or Remote User ID and Password

You can change the local or remote user ID and password of a pipe link if the pipe link is not connected on the local (or remote) side to a pipe that is connected to an application. When you change the user ID or password of a pipe link, the user ID or password of the pipe attached to it on that side is changed as well.

Changing the Direction of a Pipe Link

You can change the direction of a pipe link if the pipe link is not connected to pipes that cannot change direction. If you change the direction of a pipe link, the direction of any attached pipes is also changed.

Changing the Local or Remote Site Options

You can change the local and remote name and host of a pipe link simultaneously provided one of the following is true:

- There is no existing pipe on the new host with the new name
- If there is an existing pipe by that name, it is usable by the pipe link.

If the pipe link is connected to a pipe that is connected to an application, the original pipe is disconnected from the pipe link before the name and host change is made, so neither the name nor the node of the original pipe is changed.

You can change the direction, local and remote name, and host of a pipe link simultaneously provided both of the following are true:

- The pipe link is not connected to a pipe that is connected to an application.
- The name and host of the pipe connected to the link can be changed. (That is, an existing pipe must be usable in the new direction of the pipe link.)

You can change the local and remote site string options. If you change them to illegal or incompatible values, the Application Linker will warn you both when you make the change and when the configuration is validated.

Changing Other Pipe Link Attributes and Options

You can change the global attributes of a pipe link (such as wait time, recovery, and blank conversion) provided the new values are consistent with the entire set of values.

Running Large Application Linker Configurations

If you wish to use the Application Linker to run configurations with more than 20 objects, ensure that the following system environment characteristics are set appropriately:

- Allow a sufficient number of processes per user on each node. A pipe link can spawn up to 4 processes on its local host and up to 3 processes on its remote host.
- Provide adequate paging space on each node. Each spawned process from the CLIO/S Application Linker can use up to 2.5 MB of memory. If you specify the Use Small Buffers option, for pipe links, memory usage for spawned pipe link processes is reduced from 2.5 MB to 1.5 MB or less.
- When running many MVS applications or pipe links that connect to MVS, ensure a sufficient number of MVS initiators

Part 3. Command Line Interfaces

Chapter 6. Accessing CLIO/S Tapes With AIX Commands and Subroutines 83

Chapter 7. Transferring Files Using the CLIO/S File Transfer Program . . 91

Chapter 8. The CLIO/S Cross System Pipe Link 141

© Copyright IBM Corp. 1993, 1995, 1996

Chapter 6. Accessing CLIO/S Tapes With AIX Commands and Subroutines

The CLIO/S tape command interface lets you access CLIO/S tapes using standard AIX commands or programs. Before you can use this interface, an administrator must install and configure the CLIO/S tape pseudo device driver and start the tape mount daemon, as described in Chapter 4, “Configuring and Managing the CLIO/S Tape Pseudo Device Driver” on page 45.

Allocating and Using Tapes

Before you can read from or write to a tape device, you must establish a connection between your host and a CLIO/S tape server on the remote host. These are the steps you perform to access CLIO/S tapes with UNIX commands and function calls:

1. **Use the cltalloc command to specify the tape device you want.**

   cltalloc establishes a connection with the remote host, allowing you to access it through the tape mount daemon. If delayed allocation was specified when a tape mount daemon was configured, the tape device will be allocated on a system-wide basis. Anyone on the system with appropriate file permissions can access that tape device. If immediate allocation had been specified, the device will be opened for the specific user who issues the **CLTALLOC** command. Only that user can access that tape device. See “Configuring and Starting the CLIO/S Tape Mount Daemon” on page 45 for more information about configuration.

   If the tape drive you want is physically attached to MVS, CLIO/S on your host talks to a PARSRV program on the remote MVS system. PARSRV on the remote MVS host starts a tape server on that same MVS system.

   If the tape drive you want is physically attached to a remote AIX host, CLIO/S on your host uses the **rexec** command to start a tape server on the remote host. If the tape drive is physically attached to your local AIX system, CLIO/S does not require a tape server.

2. **Use standard UNIX commands or function calls to access the tape device.**

   For example, you can use **tar** to backup a filesystem or you can write a UNIX application program that sends **ioctl** requests to the tape device.

   The tape mount daemon on your host intercepts the tape I/O requests that result from your command or program, and translates them into CLIO/S calls to the tape server on the remote host.

   The remote tape server sends tape I/O commands to the tape drive and returns status and data to the daemon.

   The daemon passes the response back to you.

3. **Issue additional AIX commands or I/O calls, repeating step 2.**

4. **Release the tape drive using the cltunall command.**

   When you no longer need access to the tape drive, use the **cltunall** command to unallocate the tape and stop the remote tape server if one was created for you.

© Copyright IBM Corp. 1993, 1995, 1996

See “Special Considerations for ANSI Standard Label (AL) Tapes” on page 247 for information about ANSI Standard Label (AL) tapes.

User Commands

cltalloc—Allocating a CLIO/S Tape Device for MVS

Description

The **cltalloc** command is used to allocate tapes on either MVS systems or AIX systems. Because there are significant differences in the way tapes are allocated on these systems, the **cltalloc** is described twice, once for each system.

Syntax

cltalloc [–**c**] [–**d** [**dsn**=]*dsname*] [–**e** [**ex**=]*yyddd*] [–**f** *device*] [–**l** **[nl** | **al** | **blp** | **sl**]][–**o** *"string"*] [–**p** [**new** | **old**]] [–**r** [**ret=]***nnnn*] [–**v** [**vol**=]*volser*]

Parameters

The person who configures the tape device daemon can specify default allocation parameters when the tape device daemon is configured. You can accept the defaults or use these flags to override them. Your system administrator can to tell you what defaults were set, or you can use SMIT to examine them.

–**c** If you specify –**c** with –**p** **new**, this parameter places an entry pointing to the data set in the system or user catalog by including the character string “CATLG” in the DD statement.

If you specify –**c** with –**p** **old**, then, assuming that an entry for the dataset exists in the catalog:

- You do not have to provide the tape volser with –**v** **volser**
- You must specify –**d** to provide the dataset name.

If the name is not known, use –**l** **blp** parameter to bypass label processing.

–**d** [**dsn**=]*dsname*
Specifies the name of a tape data set. If omitted, the default is taken from the allocation data specified when the device was configured with SMIT. If the data set name was not specified in SMIT, a name is created using the timestamp and job information determined by the MVS installation.

The value you specify for *dsname* is converted to upper case.

–**e** [**ex**=]*yyddd*
Specifies the expiration date for a **new** data set. *yyddd* is a Julian date.

Note: –**e** and –**r** are mutually exclusive. You can specify the longevity of the tape as either an expiration date (–**e** *yyddd*) or as a number of days from now (–**r** *nnn*).

–**f** *device* Specifies the tape device. This is the pseudo device associated with the desired physical device (for example, /**dev**/**pst0**).

This is a required parameter.

If the tape drive is physically attached to MVS and you are using cataloged MVS data sets, do not use the tape in no-rewind mode on AIX (that is, do not specify the device as /**dev**/**pst0.1**). Writing multiple files in this mode has unpredictable results. To write multiple files to a cataloged MVS data set using the tape pseudo device driver, use a separate **cltalloc** and **cltunall** for each file.

–**l [nl** | **al** | **blp** | **sl]**
Indicates the tape label type. If omitted, the value specified in the allocation data field in SMIT is used. If the label attribute was not specified when the device was configured, the default is **sl**.

You may type **nl**, **al**, **blp**, and **sl** in upper, lower, or mixed case; it is converted to upper case.

Notes:

1. **al** and **sl** tapes rewind automatically at the completion of the UNIX command but do not unload.
2. For **al** (ANSI standard label) tapes, CLIO/S supports ANSI version 3 standard labels.

–**o** *"string"* The **-o** flag is a general-purpose comprehensive way to provide all MVS tape allocation paramters. If the **-o** flag is used, the **-c**, **-d**, **-e**, **-l**, **-p**, **-r**, and **-v** flags are ignored. Any default parameters specified when the tape device was defined are also ignored. Therefore, if the **-o** flag is used, it must specify all of the desired tape allocation parameters.

"string" (note that the quotation marks are required) consists of MVS tape allocation parameters. This string is passed unchanged to the CLIO/S MVS tape server which uses it to specify the SVC99 dynamic allocation parameters. A wide range of parameters is supported. See “ctpalloc or tpalloc—Starting the Remote Tape Server” on page 227 and Appendix E, “Dynamic Device Allocation on MVS” on page 303 for details on supported parameters.

The parameters should be provided in the format specified in “ctpalloc or tpalloc—Starting the Remote Tape Server” on page 227 instead of the 'flag' format. For example, use **new** instead of **-p new**, and **ret=100** instead of **-r 100**.

CLIO/S requires a DD name for internal handling. If the **DD=** parameter is not specified, then CLIO/S will append **DD=TAPE01** to the allocation parameter string.

–**p [new** | **old]**
Describes the disposition status (DISP) of a data set. If omitted, the default is the value of the DISP= allocation parameter if specified in SMIT. If DISP= was not specified in SMIT, the default is **old**.

You may type **new** and **old** in upper, lower, or mixed case; it is converted to upper case.

–**r** [**ret**=]*nnnn*

Specifies the retention period for a **new** data set, in number of days.

Note: –**e** and –**r** are mutually exclusive. You can specify the longevity of the tape as either an expiration date (–**e** *yyddd*) or as a number of days from now (–**r** *nnnn*).

–**v** [**vol**=]*volser*

Specifies the volume serial number of the tape volume. If omitted, the default is the volser the system administrator specified when configuring the daemon, by default, **DUMMY**.

The value you specify for *volser* is converted to upper case.

Description

Use this command to specify which tape drive you want to use. The tape pseudo device driver uses the parameters from this command to set up internal control blocks and start the remote tape server. For MVS tapes, the parameters refer to the Data Definition (DD) statement used to describe tape devices on MVS.

The command waits until the operator has mounted the requested tape on the requested device. You must wait until the command completes and you receive a message that the **cltalloc** command is complete before you can issue commands that use the tape drive.

Usage Notes

CLIO/S supports using tapes as special files /**dev**/**pst***n* and /**dev**/**pst***n***.1** (no rewind mode). Do not use 'no rewind mode' when writing cataloged tapes. This results in creating multiple files on a tape with only a single catalog entry, and the results are unpredictable. Use a separate **cltalloc** and **cltunall** for each file.

You must have a **.netrc** file defined. The **.netrc** file is referenced at the time the CLIO/S tape daemon is started. See “The NETRC File” on page 34 for information about creating a **.netrc** file.

Some validity-checking is performed by the CLIO/S tape daemon before the allocation request is sent to the CLIO/S MVS tape server. However, in some cases the allocation request issued via the SVC99 call on MVS may fail due to incorrect parameters or other system considerations.

Messages and Responses

This command displays one of the following messages upon completion:

- `cltalloc: Allocation failed on volser for /dev/pstn: A tape is already mounted`
- `cltalloc: Allocation failed on volser for /dev/pstn: Dynamic allocation failed`

This message indicates that the command has failed because an error was detected on the adapter or the SCSI bus.

- `cltalloc: dsname cannot be more than 44 chars`
- `cltalloc: expiration date should be yyddd`
- `cltalloc: retention period should be nnnn`

The retention period must be entered as a decimal number of days.

- `cltalloc: Usage: cltalloc: [-v [vol=]volser] [-p new | old] [-c] [-l nl | al | blp | sl] [-e [ex=]yyddd] | -r [ret=]nnnn] | -o "MVS Allocation Parameter" -f device`
- `cltalloc: volser cannot be more than 6 chars`
- `cltalloc: volser is now mounted on /dev/pstn`

Examples

1. To allocate the MVS tape device associated with /**dev**/**pst0**, mounting a previously-created tape with volser **mytape** and data set name **backup.stuff**, issue:

   ```
   cltalloc -p old -v mytape -d backup.stuff -f /dev/pst0
   ```

   OR

   ```
   cltalloc -o "old vol=mytape dsn=backup.stuff" -f /dev/pst0
   ```

2. To allocate the same tape device as the previous example, back up the **$HOME**/**mydata** from AIX to MVS, then unallocate the tape device:

   ```
   cltalloc -p old -v mytape -d backup.stuff -f /dev/pst0
   tar -cvf /dev/pst0 $HOME/mydata
   cltunall -f /dev/pst0
   ```

3. To allocate the MVS tape device associated with /**dev**/**pst2**, extract a previously-archived directory without rewinding the tape, extract another directory with rewind, and lastly, unallocate the tape device:

   ```
   cltalloc -p old -v mytape -d backup.stuff -f /dev/pst2
   tar -xvf /dev/pst0.1 my_back_up
   tar -xvf /dev/pst0 another_back_up
   cltunall -f /dev/pst0
   ```

   The first instance of the **tar** command did not rewind the tape. Then, at the end of the second **tar** command, the tape is rewound.

4. The following command allocates the new cataloged data set '**galina.recent**', using the AIX tape device /**dev**/**pst1**. The tape uses standard labels and the data set expires on March 19, 1996 (Julian date 96078). Because –**v** is not specified, the default is the volume entered on the SMIT panel when the tape pseudo device driver was configured.

   ```
   cltalloc -p new -c -l sl -e 96078  \
            -d \'galina.recent\' -f /dev/pst1
   ```

5. The following command allocates a new cataloged data set '**spbackup.kn144n07.061496**', using the AIX tape device /**dev**/**pst1**. The tape uses standard labels and the data set will be retained for 100 days.

   Because **vol=** is not specified, and the default setting from the SMIT definition is not used with the –**o** flag, no volume parameter will be transmitted to the CLIO/S MVS tape server. In this case, CLIO/S will allow MVS (or a tape management system on MVS) to assign the volume identifier. Also, CLIO/S will handle end-of-volume occurences on MVS without passing this event back to AIX. See “Effects of MVS Tape Management on CLIO/S” on page 241 for information on tape management.

The flags1 setting tells the MVS system to wait for a tape drive to become available rather than terminating the tape server. See “The **FLAGS1** and **FLAGS2** Fields” on page 303 for information on the flags parameter.

```
cltalloc -o "new catalog sl exp=97078\
             dsn=spbackup.kn144n07.061496\
             recfm=fb blksize=32720 lrecl=80\
             flags1=xxxx\
             unit=ctape"\
          -f /dev/pst1
```

Files

$HOME/.netrc The **.netrc** file with password information.

Related Topics

Commands: **cltunall**

cltalloc—Allocating a CLIO/S Tape Device for AIX

Syntax

cltalloc [–**v** [**vol**=]*volser*] [–**d** [**dsn**=]*dsname*] –**f** *device*

Parameters

The person who configures the tape device daemon can specify default allocation parameters when the tape device daemon is configured. You can accept the defaults or use these flags to override them. Your system administrator can to tell you what defaults were set, or you can use SMIT to examine them.

–**f** *device* Specifies the tape device. This is the pseudo device associated with the desired physical device (for example, /**dev**/**pst0**).

This is a required parameter.

–**v** [**vol**=]*volser*

Specifies the volume serial number of the tape volume. If omitted, the default is the volser the system administrator specified when configuring the daemon, by default, **DUMMY**.

The value you specify for *volser* is converted to upper case.

Description

Use this command to specify which tape drive you want to use. The tape pseudo device driver uses the parameters from this command to set up internal control blocks and start the remote tape server. used to describe tape devices on MVS.

The command waits until the operator has mounted the requested tape on the requested device. You must wait until the command completes and you receive a message that the **cltalloc** command is complete before you can issue commands that use the tape drive.

Usage Notes
CLIO/S supports using tapes as special files /**dev**/**pst***n* and /**dev**/**pst***n***.1** (no rewind mode).

You must have a **.netrc** file defined. See "The NETRC File" on page 34 for information about creating a **.netrc** file.

Messages and Responses
This command displays one of the following messages upon completion:

- `cltalloc: Allocation failed on volser for /dev/pstn: A tape is already mounted`
- `cltalloc: Allocation failed on volser for /dev/pstn: Dynamic allocation failed`

  This message indicates that the command has failed because an error was detected on the adapter or the SCSI bus.
- `cltalloc: dsname cannot be more than 44 chars`
- `cltalloc: Usage: cltalloc: [-v [vol=]volser] [-d [dsn=]dsname] [-p new | old] [-c] [-l nl | al | blp | sl] [-e [ex=]yyddd] | -r [ret=]nnnn] -f device`
- `cltalloc: volser cannot be more than 6 chars`
- `cltalloc: volser is now mounted on /dev/pstn`

Examples
1. To allocate the AIX tape device /**dev**/**pst0** when no volser is required, issue:

   ```
   cltalloc -f /dev/pst0
   ```

Files
$HOME/.netrc The **.netrc** file with password information.

Related Topics
Commands: **cltunall**

cltunall—Unallocating a CLIO/S Tape Device

Syntax
cltunall –**f** *device*

Parameters
–**f** *device* Specifies the tape device being unallocated. For both AIX and MVS, this is the pseudo device associated with the desired physical device (for example, /**dev**/**pst0**).

Description
This command stops the remote tape server and unallocates the tape device.

Messages and Responses

This command displays one of the following messages upon completion:

- `cltunall: Error - no tape is allocated on /dev/pst`*n*

  A tape is already mounted.

- `cltunall: Tape on /dev/pst`*n*` is now unallocated`
- `cltunall: Usage: cltunall -f device`

Examples

To unallocate the CLIO/S tape device /**dev**/**pst1**, issue:

```
cltunall -f /dev/pst1
```

Related Topics

Commands: **cltalloc**

Chapter 7. Transferring Files Using the CLIO/S File Transfer Program

The CLIO/S File Transfer Program lets you transfer files between your local host and another host that supports CLIO/S. You transfer files using the CLFTP command and its subcommands, using an interface consistent with that provided by TCP/IP's FTP. Files can reside on disk, tape, or in BatchPipes/MVS. You can also use CLFTP to access raw devices on AIX, such as tape drives.

Specifically, CLFTP allows you to:

- Establish a connection to a remote host or to your local host
- Identify yourself to the host
- Obtain status information about local and remote hosts
- Work with directories and partitioned data sets on the local or remote host
- Transfer files on disk and tape to and from the host
- Send information to the local and remote hosts
- Transfer files on disk, tape, and in BatchPipes/MVS across intermediate systems
- Access raw devices on AIX
- Obtain assistance for the CLFTP subcommands
- Work interactively or in batch mode

If you are connecting to an MVS host, the CLIO/S server program, PARSRV, must be running on the remote system. One PARSRV can be shared among multiple users and by users on multiple hosts. “The CLIO/S Server—PARSRV” on page 34 describes how to start PARSRV.

This chapter describes how to use the CLFTP command and its subcommands.

How CLIO/S FTP Works

The CLFTP command is the client side of a set of client/server programs. When you invoke CLFTP, it uses TCP/IP to connect to the CLIO/S server, PARSRV, on the remote host. PARSRV on the remote host starts the file transfer server, FCFFSERV. FCFFSERV switches to a faster connection over a CTC, if one is available, to communicate with CLFTP.

When you exit from CLFTP, the FCFFSERV started by PARSRV stops automatically.

If There Are Multiple PARSRVs

If your installation runs multiple PARSRVs, set the environment variable CLIOPSPT to the port number on which your desired PARSRV is listening. The person who started PARSRV tells you what this is. Do this before invoking CLFTP; if you do not set CLIOPSPT, CLFTP connects to PARSRV on the default port, which is defined in **fcfconf.c**.

For example, to use a PARSRV listening on port number 1978, you would set CLIOPSPT to that value and export the environment variable. Korn shell users do this with the command:

© Copyright IBM Corp. 1993, 1995, 1996

```
export CLIOPSPT=1978
```

C shell users use this command:

```
setenv CLIOPSPT 1978
```

If PARSRV is registered in **/etc/services** on AIX or in **TCPIP.profile.TCPIP** on MVS, you must set the CLIOPSID environment variable to the name corresponding to the server/port combination you want to use.

The CLFTP Command

Description

Use the CLFTP command to transfer data between your local host and a remote host.

Syntax

clftp *remote_host*[**:***remote_host*]...

Note: On AIX, the CLFTP command is located in **/bin/clftp**. You must invoke the command as **clftp**, in lower case.

Parameters

remote_host Specifies the name of the host to which you are connecting. Specify the host by its host name, its IP address, or a nickname specified in the PARCONF file. See "The PARCONF File" on page 26 for information about creating and using the PARCONF file. The host can be a remote host or your local host.

If you do not specify a *remote_host* with the CLFTP command. you receive an error message. If you specify *remote_host* incorrectly or if the host is not accessible, you receive an error message after the timeout period.

*remote_host***:***remote_host* **...**

If you specify a colon-separated list of *remote_host*s, you are requesting file transfer through intermediate hosts. The first *remote_host* specifies the first intermediate host, using a name known to your local host, either an internet name or a nickname in the local PARCONF file. The second *remote_host* is the name of a machine as known to the preceding intermediate host.

For example, to get data from ***mvsnode*** using ***riscnode1*** as an intermediate host, use the command:

```
CLFTP riscnode1:mvsnode
```

The name ***mvsnode*** can be either an internet machine name or a nickname defined in the PARCONF file on ***riscnode1***.

See "Transferring Data Across Intermediate Hosts" on page 133 for more information.

Usage Notes

1. To invoke CLFTP from a TSO session, the CLIO/S load library must be allocated under DD name EDCMTF and added to your system STEPLIB concatenation. You can do this for your TSO session using the command:

   ```
   ALLOC DD(EDCMTF) DSN(SYS1.SFCFLOAD) SHR REUSE
   ```

   A system administrator can do this for all users, as described in “Enabling TSO Access to the CLFTP and CLPLINK Commands” on page 44.

2. TSO users need MOUNT permission to use CLFTP to read or write tapes on MVS. On systems using RACF, the user must be authorized to the resource class TSOAUTH, profile MOUNT, with at least READ permission.

3. By default, CLFTP has a 30 second timeout on connect. If the connection is not made within 30 seconds, the command returns an error message. You can change the time out value by setting the CLIOTMO environment variable before you invoke CLFTP.

4. CLFTP uses QSAM for file I/O. Files that cannot be processed by QSAM are not read or written by CLFTP.

Identifying Yourself to CLFTP

You can chose one of these ways to identify yourself to the remote host:

- Replying to prompts within the CLFTP command
- Placing user ID and password information in the NETRC file

Specifying User ID and Password at the CLFTP Prompt

If you correctly specify a remote host, the CLFTP command prompts you to identify yourself. Reply to the prompt with the user ID you want to use on the remote host. You are then prompted for the password associated with that user ID.

CLFTP does not display your password as you enter it. This protects the security of your passwords.

The following is an example of the information that is displayed after you successfully invoke CLFTP:

```
clftp mvsnode
Welcome to clftp Version 1
Name (mvsnode:jbecky): bidwell
Password:
Connecting to : mvsnode.hpssl.kgn.ibm.com
Remote system mvsnode connected with protocol ESCON.
clftp >
```

Placing User ID and Password in the NETRC File

If there is an entry for the remote host in the local NETRC file and the entry specifies both user ID and password, those values are used. If the NETRC entry specifies only the user's login name, CLFTP prompts for the password. If the NETRC file does not contain an entry for the remote host, CLFTP prompts for both a user ID and password.

For more information about NETRC files, see “The NETRC File” on page 34. Special NETRC considerations apply when using CLFTP to transfer data across one or more intermediate hosts; see “Specifying User and Password Information When Using Intermediate Hosts” on page 134.

Establishing Your Working Directory

When you establish a CLFTP connection, a default working directory is set for both the local and remote hosts. On MVS, you can control which data set is used as the default directory. The following sections describe the default directories and how you control them.

Your AIX Working Directory

The default working directory on a remote AIX host is the home directory of the user ID you use to connect to that host. To change the remote working directory on an AIX host for the duration of a CLFTP session, use the CD subcommand, described in “CD—Changing the Directory on the Remote Host” on page 102.

Your local working directory on an AIX host is set to the current working directory (Korn shell $PWD) in effect when you invoked CLFTP. To change the local working directory on an AIX host for the duration of a CLFTP session, use the LCD subcommand, described in “LCD—Changing the Local Working Directory” on page 107.

You can also set a working directory by putting a CD or LCD subcommand in the CLFTP defaults file. See “Customizing CLFTP Defaults” on page 95 for more information.

Your MVS Working Directory

On MVS, a “directory” is a common file name prefix for a group of sequential data sets or the qualifiers of a partitioned data set. When the “directory” is a partitioned data set, files are the members of the partitioned data set.

The default working directory on both local and remote MVS hosts is the data set named in the CLIO/S FTP information file, CLFTPINF, on that host. If this file doesn't exist, you do not have a default working directory; you can either:

- Define a working directory with the CD and LCD subcommands, described in “CD—Changing the Directory on the Remote Host” on page 102 and “LCD—Changing the Local Working Directory” on page 107
- Specify the fully-qualified data set name when you transfer files

Your CLFTPINF file is the MVS data set *user_ID*.**CLFTPINF**, which contains a single line of the form:

```
prefix=qualifier
```

For example, to set the MVS file name prefix to `SYS1`, user BRET puts this line in the **BRET.CLFTPINF** file:

```
prefix=SYS1
```

This lets you refer to the file `'sys1.output'` simply as `output`.

You can also set a working directory by putting a CD or LCD subcommand in the CLFTP defaults file. See “Customizing CLFTP Defaults” on page 95 for more information.

Customizing CLFTP Defaults

You can put personal defaults for CLFTP in a file, rather than typing your preferred settings each time you invoke the command. For example, if you most often use CLFTP to transfer binary files, you can change the default transfer mode for your user ID from ASCII to BINARY.

On MVS, the defaults file is *userid***.CLFTP.DATA**. On AIX, the defaults file is **$HOME/.clftprc**.

The CLFTP defaults file contains a list of CLFTP subcommands, entered as if you were typing them at the prompt inside CLFTP. Each subcommand goes on a separate line. SITE or LOCSITE subcommands should list all parameters on a single line. The following options can be changed in the CLFTP defaults file:

ASCII
BINARY
CD
HASH
LCD
LOCSITE
SITE
TRANSLATE

For example, the MVS user COLAO can set his default file transfer mode to binary by putting this line in the file COLAO.CLFTP.DATA:

```
binary
```

If an AIX user tires of typing the site information about binary files to be transferred from AIX to MVS, she can establish defaults by placing these lines in the file **$HOME/.clftprc**:

```
binary
site pri=10 sec=10
```

Input and Output Files for CLFTP

CLFTP for MVS supports DD INPUT and DD OUTPUT as specially recognized input and output files. Command input is read through the DD INPUT file, and CLFTP messages are written to the DD OUTPUT file. On AIX, these the same functions are available through environment variables CLFTPINPUT and CLFTPOUTPUT.

CLFTP Subcommands

Use the CLFTP subcommands to perform tasks related to transferring files between hosts. The following table lists these tasks and the subcommands associated with each.

Table 4. CLFTP Tasks

Task	Subcommands	Page
Displaying help	HELP	97
Obtaining status and system information	ECHO HASH LOCSTAT STATUS	98
Working with directories and partitioned data sets on the remote host	CD LS MKDIR PWD	101
Working with directories and partitioned data sets on the local host	LCD LMKDIR LPWD LLS	107
Preparing the environment	ASCII BINARY ENDONERROR LOCSITE SITE TRANSLATE	111
Transferring data	APPEND DELETE GET PUT	127
Disconnecting from the host	QUIT	132

The CLFTP subcommands are listed in Table 5. The minimum abbreviation, a description, and the page reference for each subcommand are also included. You use the CLFTP subcommands at the `clftp >` prompt after issuing the CLFTP command.

The subcommands may be entered in upper, lower, or mixed case. The minimum abbreviation for each subcommand is shown in uppercase letters.

Table 5 (Page 1 of 2). CLFTP Subcommands

Subcommand	Description	Page
APpend	Appends to a remote data set.	127
AScii	Sets the transfer type to ASCII. Use this to transfer text (non-binary) files between ASCII and/or EBCDIC hosts.	111
Binary	Sets the transfer type to IMAGE.	112
CD	Changes the working directory on the remote host.	102
DELETE	Deletes files and partitioned data sets on the remote host.	128
ECHO	Controls the redisplay of subcommands	98
ENDONERROR	Controls whether CLFTP quits after an error.	112
Get	Copies a file from the remote host to your local host.	129
HAsh	Controls the display of file transfer status messages.	98
HElp	Displays help information for CLFTP.	97

Table 5 (Page 2 of 2). CLFTP Subcommands

Subcommand	Description	Page
LCd	Changes the local working directory.	107
LLS	Lists the files in a local directory	108
LMkdir	Creates a directory on the local host.	109
LPwd	Displays the local current directory qualifier	111
LOCSIte	Specifies information that is used by the local host to provide service specific to that host system.	113
LOCSTat	Displays status information for the local host.	99
LS	Lists the files in a remote directory	104
MKdir	Creates a directory on the remote host.	104
PUt	Copies a file on your local host to the remote host.	131
PWd	Displays the current working directory.	106
QUIt	Leaves the CLFTP command environment.	132
SIte	Specifies information that is used by the remote host to provide service specific to that host system.	114
STatus	Displays status information for the remote host.	100
TRANSlate	Controls whether ASCII/EBCDIC translation is done on the local or the remote host.	126
?	Displays help information for CLFTP. Synonymous with HELP.	97

Displaying Help

HELP or ?—Displaying Help

Description

The HELP subcommand provides assistance with the CLFTP subcommands. This subcommand may also be entered as **?**.

Syntax

HElp [*subcommand*]

Parameters

subcommand Specifies the name of the subcommand.

Usage Notes

If you enter the HELP subcommand without a parameter, you see a menu which lists the subcommands and a description of the information available.

Obtaining Status and System Information

To display status information about the local and remote host, use the LOCSTAT and STATUS subcommands. Use the HASH subcommand to control the display of status messages during file transfer. To control the redisplay of subcommands, use the ECHO subcommand.

ECHO—Controlling the Redisplay of Subcommands

Description

The ECHO subcommand controls whether subcommands are redisplayed on the output device. This subcommand acts as a toggle; invoking it changes the setting. The default setting is OFF. When the toggle is set to ON, each subcommand you invoke is displayed on the output device. When running CLFTP as a batch job on MVS or when reading subcommands from a file or stdin on AIX, turn the ECHO toggle ON to write the subcommands to the output file. When the toggle is OFF, you see the output from a subcommand, but not the subcommand itself.

Syntax

ECHO

Parameters

The ECHO subcommand has no parameters.

Examples

See “Starting CLFTP as a Batch Job” on page 133 for sample JCL that uses the ECHO subcommand in a batch CLFTP job.

HASH—Controlling the Display of File Transfer Status Messages

Description

The HASH subcommand controls whether status messages are displayed during file transfer. This subcommand acts as a toggle; invoking it changes the setting. The default setting is OFF. When the toggle is set to ON, a status message is displayed at regular intervals during file transfer.

Syntax

HAsh

Parameters

The HASH subcommand has no parameters.

Usage Notes

To display the current setting of the HASH toggle, use the LOCSTAT subcommand.

Examples

The following example illustrates the use of the HASH subcommand. It demonstrates how to display the setting of the HASH toggle using the LOCSTAT subcommand and how to change the toggle with the HASH subcommand. It also shows the status messages that appear during the transfer of large files:

```
clftp > hash
Hash mark : Off
clftp > locstat

Connected to: mvsnode (MVS)
ascii: On               Status Freq.: 4
Last Site String: sec=27
Hash Mark Printing: Off
Maxbuflen = 500000      Firstbuflen = 100000    Secondbuflen = 100000
clftp > hash
Hash mark : On
clftp > put /u/bidwell/bigfile 'bidwell.bigfile'
42073723 bytes transferred, Elapsed time: 10.01 sec., 4.01 Meg/s
78088395 bytes transferred, Elapsed time: 20.09 sec., 3.71 Meg/s
114582993 bytes transferred, Elapsed time: 30.11 sec., 3.63 Meg/s
150577665 bytes transferred, Elapsed time: 40.19 sec., 3.58 Meg/s
187072263 bytes transferred, Elapsed time: 50.39 sec., 3.54 Meg/s
050 Transfer complete (200000000 bytes).
    Elapsed time: 53.50 sec.3.57 Meg/s
```

LOCSTAT—Retrieving Status Information From the Local Host

Description

To retrieve status information from the local host, use the LOCSTAT subcommand. The local status information includes:

- Whether data transfers are in ASCII or BINARY mode
- The last SITE string entered
- Whether intermediate transfer (HASH) messages are ON or OFF
- File allocation parameters, when the local host is MVS

Syntax

LOCSTat

Parameters

The LOCSTAT subcommand has no parameters.

Examples

The following example shows system status on a local MVS host:

```
clftp > locstat

Connected to: riscnode1 (AIX)
ascii: On               Status Freq.: 4
Last Site String:
Hash Mark Printing: Off
Maxbuflen = 500000      Firstbuflen = 100000    Secondbuflen = 100000

 Local file allocation values.

File Organization  : Physically Sequential.
Record Format      : Variable Blocked.
Record Length      : 260
Block Size         : 27648
Storage Unit Type  : Cylinders.
Primary Units      : 1
Secondary Units    : 1
```

STATUS—Retrieving Status Information From a Remote Host

Description

To retrieve status information from the remote host, use the STATUS subcommand. The remote status information includes:

- The transfer protocol
- Whether data transfers are in ASCII or BINARY mode
- The last SITE string entered
- The file allocation parameters, when the remote host is MVS

Syntax

STatus

Parameters

The STATUS subcommand has no parameters.

Examples

The following example shows system status as originally connected to the AIX host ***riscnode1***:

```
clftp > status

Connected to: riscnode1 (AIX)
ascii: On               Status Freq.: 4
Last Site String:
Transport Protocol: shared memory
Translation Site: Local
Maxbuflen = 500000      Firstbuflen = 100000    Secondbuflen = 100000
```

The following example shows system status as originally connected to MVS host ***mvsnode*** and after changing the data transfer type with BINARY and the MVS information with SITE:

```
clftp > status

Connected to: mvsnode (MVS)
ascii: On               Status Freq.: 4
Last Site String:
Transport Protocol: TCP/IP
Translation Site: Local
Maxbuflen = 500000      Firstbuflen = 100000    Secondbuflen = 100000

 Remote file allocation values.

File Organization  : Physically Sequential.
Record Format      : Variable Blocked.
Record Length      : 260
Block Size         : 27648
Storage Unit Type  : Cylinders.
Primary Units      : 1
Secondary Units    : 1
Directory Blocks   : 27
Volume             :
Dcbdsn             :
Unit               :

clftp > binary
000  -  File is IMAGE
clftp > site primary=2 lrecl=80
000  -  Command completed successfully
clftp > status

Connected to: mvsnode (MVS)
ascii: Off              Status Freq.: 4
Last Site String: primary=2 lrecl=80
Transport Protocol: TCP/IP
Translation Site: Local
Maxbuflen = 500000      Firstbuflen = 100000    Secondbuflen = 100000

 Remote file allocation values.

File Organization  : Physically Sequential.
Record Format      : Variable Blocked.
Record Length      : 80
Block Size         : 27648
Storage Unit Type  : Cylinders.
Primary Units      : 2
Secondary Units    : 1
Directory Blocks   : 27
Volume             :
Dcbdsn             :
Unit               :
```

Working With Directories and Partitioned Data Sets on the Remote Host

To work with directories on the remote host, use the following subcommands:

- CD
- LS

- MKDIR
- PWD

On MVS, a “directory” is a common file name prefix for a group of sequential data sets or the qualifiers of a partitioned data set.

The following sections describe these subcommands.

CD—Changing the Directory on the Remote Host

Description

To change the working directory on the remote host, use the CD subcommand.

Syntax

CD *directory*

Parameters

directory Specifies the name of a file directory, a fully qualified data set, or a prefix on the remote host.

You can specify the name of an MVS *directory* in upper, lower, or mixed case; it is converted to uppercase. Specify an AIX *directory* name exactly as it was created; case is significant.

Changing the Directory On a Remote AIX Host

When the remote host is AIX, you specify *directory* as a fully-qualified path name or a relative path name. For example, if the current working directory on the remote host is `/u/elvis` and you issue this CLFTP subcommand:

```
cd graceland
```

then the new working directory on the remote host becomes `/u/elvis/graceland`. If the directory `/u/elvis/graceland` does not exist, CD returns an error message. To set the remote current working directory to `/u/cheryl/gif`, issue this CD command:

```
cd /u/cheryl/gif
```

To back up one level in the directory hierarchy of the remote host, issue the CD subcommand with two periods (**..**). For example, if the current working directory is `/u/cheryl/gif`, the CD command:

```
cd ..
```

makes `/u/cheryl` the new working directory on the remote host.

Changing the Directory On a Remote MVS Host

If the remote host is MVS, *directory* specifies either a common prefix for a group of data sets or the qualifiers of a PDS. When the CD subcommand is issued, *directory* is appended to the current working directory. For example, if the current working directory is `BRET.TEST`, and you issue the CD subcommand:

```
CD FILES
```

then the new working directory becomes `BRET.TEST.FILES`.

To override the existing directory rather than append to the directory, issue the *directory* parameter in single quotation marks. For example, if the current working directory is BRET.TEST, and you issued the CD subcommand:

```
CD 'CLIO.FILES'
```

then the new working directory becomes CLIO.FILES.

If a PDS exists with the exact name of the current working directory, CLFTP considers the working directory to be that PDS. Otherwise, CLFTP considers the working directory to be a common prefix qualifier for sequential data sets.

If a PDS exists with the same name as the current working directory, but you want the current working directory to be treated as a common prefix for sequential data sets, specify the working directory with a period (.) at the end. For example, if a PDS named BRET.TEST exists, the subcommand:

```
CD 'BRET.TEST'
```

makes the PDS BRET.TEST the current working directory. A subsequent PUT of file NAME1 adds a member NAME1 to the BRET.TEST PDS. In contrast, the statement:

```
CD 'BRET.TEST.'
```

makes the current working directory, BRET.TEST., a prefix for sequential data sets. A subsequent PUT subcommand used to copy data set NAME1 would create the sequential data set BRET.TEST.NAME1.

To back up one level of the current working directory, issue the CD subcommand with 2 periods (..) at the end. For example, if the working directory is BRET.SOURCE, the statement:

```
CD ..
```

makes BRET. the working directory.

Examples

The following example demonstrates how to use the CD subcommand to change directories on a remote MVS host:

```
clftp > pwd
000  -  "BIDWELL." is current remote directory.
clftp > cd begin.
000  -  "BIDWELL.BEGIN." is current remote directory.
clftp > cd 'bidwell.jcl'
000  -  "BIDWELL.JCL" partitioned data set is current remote directory.
```

This example shows how to use the CD subcommand to change directories on a remote AIX host:

```
clftp > pwd
000  -  "/u/elvis" is current remote directory.
clftp > cd bigfiles
000  -  "/u/elvis/bigfiles" is current remote directory.
```

LS—Listing Files in a Remote Directory

Description
To get a list of the files in a remote directory, use the LS subcommand.

Syntax
ls [*–flags*] [*directory*]

Parameters
–flags When the remote system is AIX, you can use any flags accepted by the AIX **ls** command.

When the remote system is MVS, do not specify any flags.

directory Specifies the name of the file you want to list or the directory whose contents you want to list.

You can specify the name of an MVS *directory* in upper, lower, or mixed case; it is converted to uppercase. Case is significant in AIX *directory* names. If you omit *directory*, the current working directory is listed.

Usage Notes
On an AIX server all directory permission settings are observed. Only directories to which the remote user has the proper permission are listed.

On MVS, normal RACF permissions apply.

Examples
The following example shows sample output from displaying the contents of an MVS directory with LS:

```
clftp > ls
BECKY.ASCII.DATA
BECKY.BINARY.DATA
BECKY.CLFTP.DATA
BECKY.PLI.SRC
BECKY.TEST
000  -  Command completed successfully.
```

MKDIR—Creating a Directory on the Remote Host

Description
To create a directory or partitioned data set on the remote host, use the MKDIR subcommand. The MKDIR subcommand sends a request to the remote host's CLFTP server to create a directory with name *directory* in the current working directory.

On MVS, a PDS is created with the fully qualified name if *directory* is specified in single quotation marks. If *directory* is not specified in single quotations marks, CLFTP appends *directory* to the current working directory, if one is defined. If you

do not have a working directory because you don't have a **CLFTPINF** file and have not yet issued the CD subcommand, an error message is displayed. The new data set has the characteristics specified on the SITE subcommand.

On AIX, specify *directory* as either a fully-qualified or relative path name.

If a directory or PDS with the same name already exists, MKDIR issues an error message and does not overwrite it.

To verify that the directory was created by MKDIR, use the CD subcommand to change to the new directory, followed by the PWD subcommand to display its name.

Syntax

MKdir *directory*

Parameters

directory Specifies the name of the directory to be created.

You can specify the name of an MVS *directory* in upper, lower, or mixed case; it is converted to uppercase. Specify an AIX *directory* name exactly as you want it created; case is significant.

Examples

This example demonstrates how to create an AIX directory by specifying both a relative and fully-qualified path name:

```
clftp > pwd /tmp
000  -  "/tmp" is current remote directory.
clftp > mkdir newdir
000  -  Command completed successfully
clftp > cd newdir
000  -  "/tmp/newdir" is current remote directory.
clftp > mkdir /tmp/anotherdir
000  -  Command completed successfully
clftp > cd /tmp/anotherdir
000  -  "/tmp/anotherdir" is current remote directory.
```

The following example shows how to specify directory names on a remote MVS host with both relative and fully-qualified names:

```
clftp > pwd
000  -  "BILL." is current remote directory.
clftp > mkdir 'bill.clio.test'
000  -  Command completed successfully
clftp > cd clio
000  -  "BILL.CLIO." is current remote directory.
clftp > mkdir t2
000  -  Command completed successfully
clftp > cd t2
000  -  "BILL.CLIO.T2" partitioned data set is current remote directory.
clftp > cd 'bill.clio.test'
000  -  "BILL.CLIO.TEST" partitioned data set is current remote directory.
```

PWD—Displaying the Current Working Directory on the Remote Host

Description

To display the name of the current working directory on the remote host, use the PWD subcommand.

Syntax

PWd

Parameters

The PWD subcommand has no parameters.

Examples

When using the PWD subcommand to display the working directory on a remote AIX host, you see output similar to:

```
clftp > pwd
000  -  "/u/elvis/graceland" is current remote directory.
```

The following output results from using the PWD subcommand to display the working directory on a remote MVS host, when the directory is a sequential data set and a partitioned data set:

```
clftp > pwd
000  -  "BIDWELL." is current remote directory.
clftp > cd 'bidwell.jcl'
000  -  "BIDWELL.JCL" partitioned data set is current remote directory.
clftp > pwd
000  -  "BIDWELL.JCL" partitioned data set is current remote directory.
```

Working With Directories and Partitioned Data Sets on the Local Host

To work with directories on the local host, use the following subcommands:

- LCD
- LLS
- LMKDIR
- PWD

On MVS, a "directory" is a common file name prefix for a group of sequential data sets or the qualifiers of a partitioned data set.

The following sections describe these subcommands.

LCD—Changing the Local Working Directory

Description

To change the current working directory on the local host, use the LCD subcommand.

Syntax

LCd *directory*

Parameters

directory
: Specifies an AIX directory name or either a common prefix for a group of sequential data sets or the qualifiers of an MVS partitioned data set.

  You can specify the name of an MVS *directory* in upper, lower, or mixed case; it is converted to uppercase. Specify an AIX *directory* name exactly as it was created; case is significant.

Changing the Directory On a Local AIX Host

When the local host is AIX, you specify *directory* as a fully-qualified path name or a relative path name. For example, if the current local working directory is `/u/elvis` and you issue this CLFTP subcommand:

```
lcd graceland
```

then the new working directory on the local host becomes `/u/elvis/graceland`. If the directory `/u/elvis/graceland` does not exist, LCD returns an error message. To set the current working directory on the local host to `/u/cheryl/gif` issue this LCD command:

```
lcd /u/cheryl/gif
```

To back up one level in the local directory hierarchy, issue the LCD subcommand with two periods (**..**). For example, if the current working directory is `/u/cheryl/gif`, the LCD command:

```
lcd ..
```

makes `/u/cheryl` the new local working directory.

Changing the Directory On a Local MVS Host

When the LCD subcommand is issued, *directory* is appended to the current local working directory. For example, if the current local working directory is `BRET.TEST`, and you issue the LCD subcommand:

```
LCD FILES
```

the new working directory becomes `BRET.TEST.FILES`.

To override the existing directory rather than append to the directory, issue the *directory* in single quotation marks. For example, if the current local working directory is `BRET.TEST`, and you issued the LCD subcommand:

```
LCD 'CLIO.FILES'
```

then the new working directory is `CLIO.FILES`.

If a PDS exists with the exact name of the current local working directory, CLFTP considers the working directory to be that PDS. Otherwise, CLFTP considers the working directory to be a common prefix qualifier for sequential data sets.

If a PDS exists with the same name as the current local working directory, but you want the current local working directory to be treated as a common prefix for sequential data sets, specify the working directory with a period (.) at the end. For example, if a PDS named `BRET.TEST` exists, the subcommand:

```
LCD 'BRET.TEST'
```

makes the PDS `BRET.TEST` the current local working directory. A subsequent GET subcommand used to copy data set `NAME1` would add the member `NAME1` to the `BRET.TEST` PDS. In contrast, the statement:

```
LCD 'BRET.TEST.'
```

would make the current local working directory, `BRET.TEST.`, a prefix for sequential data sets. A subsequent GET subcommand used to copy data set `NAME1` would create the sequential data set `BRET.TEST.NAME1`.

To back up one level of the current local working directory, issue the LCD subcommand with 2 periods (..) at the end. For example, if the working directory is `BRET.SOURCE`, the statement:

```
LCD ..
```

makes `BRET.` the working directory.

LLS—Listing Files in a Local Directory

Description

To get a list of the files in a local directory, use the LLS subcommand.

Syntax

ls [*–flags*] [*directory*]

Parameters

–flags When the local system is AIX, you can use any flags accepted by the AIX **ls** command.

When the local system is MVS, do not specify any flags.

directory Specifies the name of the file you want to list or the directory whose contents you want to list.

You can specify the name of an MVS *directory* in upper, lower, or mixed case; it is converted to uppercase. Case is significant in AIX *directory* names. If you omit *directory*, the current working directory is listed.

Usage Notes

1. On an AIX client, all directory permission settings are observed. Only directories to which the local user has the proper permission are listed.

   On MVS, normal RACF permissions apply.

2. To use LLS on an MVS client CLFTP, add this statement to your *userid*.**BEGIN.CLIST**:

   ```
   ALLOC DD(REXXLST) DA('SYS1.SFCFSAMP') SHR
   ```

   SYS1.SFCFSAMP is the partitioned data set containing the IBM-supplied member FCFFLIST. FCFFLIST is a REXX program that implements the LLS function on MVS.

Examples

The following example shows sample output from displaying the contents of an AIX directory with LLS:

```
clftp > lls
tst.file1
tstdir
sarah7
clsamp
200 Command completed successfully.
```

LMKDIR—Creating a Directory on the Local Host

Description

To create a directory or partitioned data set on the local host, use the LMKDIR subcommand. On MVS, a partitioned data set is created with the fully qualified name if *directory* is specified in single quotation marks. If *directory* is not specified in single quotations marks, CLFTP appends *directory* to the current local working directory, if one is defined. If you do not have a local working directory because you don't have a **CLFTPINF** file and have not yet issued the LCD subcommand, an error message is displayed. The new data set has the characteristics specified on the LOCSITE subcommand.

On AIX, specify *directory* as either a fully-qualified or relative path name.

If a directory or PDS with the same name already exists, LMKDIR issues an error message and does not overwrite it. There is no CLFTP subcommand to display a list of local directory entries.

To verify that the directory was created by LMKDIR, use the LCD subcommand to change to the new directory, followed by the LPWD subcommand to display its name.

Syntax

LMKdir *directory*

Parameters

directory Specifies the name of the AIX directory or MVS PDS to be created.

You can specify the name of an MVS *directory* in upper, lower, or mixed case; it is converted to uppercase. Specify an AIX *directory* name exactly as you want it created; case is significant.

Examples

This example shows the result of creating directories on a local AIX host, using both fully-qualified and relative path names:

```
clftp > lmkdir /tmp/local
000  -  Command completed successfully
clftp > lcd /tmp
000  -  "/tmp" is current local directory.
clftp > lmkdir local2
000  -  Command completed successfully
clftp > cd /tmp/local2
000  -  "/tmp/local2" is current local directory.
```

This example shows the result of creating directories on a local MVS host, using both fully-qualified and relative path names:

```
 CLFTP >
lmkdir 'bidwell.local'
 000  -  Command completed successfully
 CLFTP >
lcd 'bidwell.local'
 000  -  "BIDWELL.LOCAL" partitioned data set is current local directory.
 CLFTP >
lcd 'bidwell'
 000  -  "BIDWELL." partitioned data set is current local directory.
 CLFTP >
lmkdir cltest
 000  -  Command completed successfully
 CLFTP >
lcd cltest
 000  -  "BIDWELL.CLTEST" partitioned data set is current local directory.
```

LPWD—Displaying the Current Working Directory on the Local Host

Description
To display the name of the current directory or working-level qualifier on the local host, use the LPWD subcommand.

Syntax
LPwd

Parameters
The LPWD subcommand has no parameters.

Examples
The following is a sample response that is displayed as a result of the LPWD subcommand on AIX:

```
clftp > lpwd
000  -  "/usr/local" is current local directory.
```

The following is a sample response that is displayed as a result of the LPWD subcommand on MVS:

```
 CLFTP >
lpwd cltest
 000  -  "BIDWELL.CLTEST" partitioned data set is current local directory.
```

Preparing the Environment

You can use the following subcommands to prepare the environment before working with data:

- ASCII
- BINARY
- LOCSITE
- SITE
- TRANSLATE

The following sections describe these CLFTP subcommands.

ASCII—Changing the Data Transfer Type to ASCII

Description
Use the ASCII subcommand to change the data transfer type to ASCII. ASCII mode indicates that you want the bytes in your file to be converted to the data representation of the receiving system, for example, when transferring text files between ASCII and EBCDIC hosts. Use the ASCII subcommand even when the receiving host is an EBCDIC system. (This contrasts with binary or image mode, which you use when data conversion should not be performed.)

ASCII is the default transfer type.

Syntax
AScii

Parameters
The ASCII subcommand has no parameters.

BINARY—Changing the Data Transfer Type to Image

Description
To change the data transfer type to image (binary), use the BINARY subcommand.

When using the image transfer type, data is sent as contiguous bits packed into 8-bit bytes. Use the image transfer type for efficient storage and retrieval of data sets or files between systems having the same data representation, and for the transfer of binary data. (This contrasts with ASCII mode, which you use when you want the bytes in your file to be converted to the data representation of the receiving host.)

Syntax
Binary

Parameters
The BINARY subcommand has no parameters.

ENDONERROR—Controlling Termination on Error

Description
The ENDONERROR subcommand controls whether the CLFTP command continues or terminates after a subcommand fails. This subcommand acts as a toggle; invoking it changes the setting. The default setting is OFF. When the toggle is OFF, CLFTP accepts subsequent subcommands after an error. When the toggle is set to ON, CLFTP terminates if a subcommand indicates an error. CLFTP sets a non-zero return code when terminating because of ENDONERROR.

ENDONERROR is most useful when running CLFTP in batch mode, letting you bail out if an error occurs.

Syntax
ENDONERror

Parameters
The ENDONERROR subcommand has no parameters.

Usage Notes
To display the current setting of the ENDONERROR toggle, use the LOCSTAT subcommand.

Examples
The following example illustrates the use of the ENDONERROR subcommand. It demonstrates how to display the setting of the ENDONERROR toggle using the LOCSTAT subcommand and how to change the toggle with the ENDONERROR subcommand. It also shows CLFTP terminating after an error, in this case, a LS subcommand specifying a non-existent name. The Korn shell command **echo $?** displays the non-zero value returned by CLFTP on an AIX system.

```
clftp > locstat

Connected to: kn1mvst (MVS)
ascii: On               Status Freq.: 4
Echo: Off
Last Site String:
Hash Mark Printing: Off
End on error: Off
Maxbuflen = 500000      Firstbuflen = 100000    Secondbuflen = 100000

 Local file allocation values.

Block size           : 0
Maximum block size   : 32768
Raw write permission : No
BLK write permission : No
200 Command completed successfully.
clftp > endonerror
End on error flag : On
clftp > ls bogus.dataset
452 Error in ls command function.
550 No Data Sets found.
671 Ending because 'end on error' flag is set.
221 GoodBye.
$ echo $?
59
```

LOCSITE—Specifying Site Information to the Local Host

Description
To specify information that is used by the local host to provide services specific to that host system, use the LOCSITE subcommand.

Syntax
LOCSIte *parameter* ...

Parameters
parameter Specifies parameters dependent on the local host system. Separate each of the parameters by a blank space.

"SITE and LOCSITE Parameters" on page 115 lists the parameters you can specify with LOCSITE.

Usage Notes
Issue the HELP LOCSITE subcommand to display a list of the types of services available on the local host.

The site-dependent information set with the LOCSITE subcommand remains active until you issue a new LOCSITE subcommand. The new LOCSITE subcommand adds to or changes the parameters established by previous LOCSITE subcommands.

If you specify one or more incorrect parameters with the LOCSITE subcommand, an error message specifying the incorrect parameter is displayed. All correct parameters are set, regardless of any incorrect parameters, and do not need to be reissued.

Note: To check the effect of the LOCSITE subcommand on the attributes at the local host, see "LOCSTAT—Retrieving Status Information From the Local Host" on page 99.

SITE—Sending Site Specific Information to a Remote Host

Description
To send information that is used by the remote host to provide services specific to that host system, use the SITE subcommand.

Syntax
SIte *parameter* ...

Parameters
parameter Specifies parameters dependent on the remote host system.

"SITE and LOCSITE Parameters" on page 115 lists the parameters you can specify with LOCSITE.

Usage Notes
Use the HELP SITE subcommand to display the types of services available on the remote host.

The site-dependent information sent with the SITE subcommand remains active until you issue a new SITE subcommand. The new SITE subcommand adds to, or changes, the attributes established by previous SITE subcommands.

If you specify one or more incorrect parameters with the SITE subcommand, an error message specifying the incorrect parameter is displayed. All correct parameters are set, regardless of any incorrect parameters, and do not need to be reissued.

Note: To check the effect of the SITE subcommand on the attributes at the remote host, see "STATUS—Retrieving Status Information From a Remote Host" on page 100.

SITE and LOCSITE Parameters

The SITE and LOCSITE parameters are used by both the CLFTP and CLPLINK commands.

The SITE and LOCSITE subcommands let you define the characteristics of the files and devices you want to access with CLFTP. Two sets of parameters are supported:

- Table 6 lists the parameters provided for defining MVS data sets
- Table 7 on page 121 lists the parameters provided for accessing AIX raw devices and block devices

On MVS, the characteristics of a data set must be defined before the data set is created. CLFTP provides defaults for many file characteristics. You override CLFTP's defaults by specifying parameters with the SITE and LOCSITE subcommands.

Table 6 lists the parameters of the SITE and LOCSITE subcommands. A description of each parameter and, where applicable, the defaults and possible values for each parameter are shown. The parameters can be specified in upper, lower, or mixed case. The minimum abbreviation for the parameter is shown in upper case.

You may specify more than one parameter per SITE or LOCSITE subcommand by separating values with the space character.

CLFTP provides many of the same SITE and LOCSITE parameters as TCP/IP's FTP on MVS; CLIO/S-specific parameters are noted in the table. See the *JCL Reference Manual*, GC28-1654, for more information about some of the SITE and LOCSITE parameters.

Table 6 (Page 1 of 6). SITE and LOCSITE Parameters for MVS Hosts

Parameters	Description
ADDZEROEOF	Adds a zero length record to the end of each transfer. This option is ignored if LENFIELDS, LENFIELDSADD or LENFIELDSSUB is not specified.
NOADDZEROEOF	Turns off the ADDZEROEOF flag.

Table 6 (Page 2 of 6). SITE and LOCSITE Parameters for MVS Hosts

Parameters	Description

ALLoc='*text*'
ALLoc+='*appended_text'*
	Specifies parameters to pass to dynamic device allocation. Use this parameter to allocate tape data sets and BatchPipes/MVS data sets. Specify *text* as a list of blank-separated allocation parameters inside single quotes (**'**), for example, ALLOC='UNIT=3480 VOLUME=TAPE01'. When the ALLoc parameter is specified, all allocation parameters must be specified in the ALLoc parameter *text* rather than as individual values. See Appendix E, "Dynamic Device Allocation on MVS" on page 303 for a list of supported parameters.
	
	If you specify `dsname=`*`name`* in the ALLOC= parameter text, you must specify a file name of `dd:` on the GET or PUT subcommand. If you specify a file name other than `dd:` on the GET or PUT subcommand, an allocation error results.
	
	The ALLOC parameter is specific to CLIO/S's CLFTP. It is not available in TCP/IP's FTP.
	
	Use ALLOC+='*appended_text'* to append allocation parameters to a previously-defined allocation string. If you want spaces between concatenated text strings, you must put them in the *text* or *appended_text* string. For example, to define the allocation string "UNIT=3480 VOLUME=TAPE01 MOD" in two parts, specify the allocation strings like this:
	
	`ALLOC='UNIT=3480 VOLUME=TAPE01'`
	`ALLOC+=' MOD'`
	
	Use ALLOC=*'allocation_data'* for the allocparm field of the **CTPALLOC()** call. See Appendix A, "User Specified I/O Exits for CLFTP and CLPLINK" on page 251 for more information.
	
	Note: If you use ALLOC= or ALLOC+= parameters, CLFTP ignores any other SITE or LOCSITE parameters.

BLocks	Specifies that primary and secondary space allocations are in blocks. If you do not specify one of BLOCKS, CYLINDERS, or TRACKS, space is allocated in CYLINDERS.
BLOCKSIze=*nnn* BLOCKSIze	Specifies the block size of a newly allocated data set. Valid range of values is 1 through 32,760. The default blocksize is 27,648 bytes. Specifying BLOCKSIZE=0 or BLOCKSIZE without a value indicates that no blocksize should be used when allocating the data set.
CONV=(*data1,data2*)	Specifies a user-written data conversion routine to be given control by CLIO/S as part of the data transfer process. Refer to Appendix B, "User Specified Data Translation for CLFTP and CLPLINK" on page 259.
CYlinders	Specifies that primary and secondary space allocations are in cylinders. If you do not specify one of BLOCKS, CYLINDERS, or TRACKS, space is allocated in CYLINDERS.

Table 6 (Page 3 of 6). SITE and LOCSITE Parameters for MVS Hosts

Parameters	Description
DCBdsn=*data_set_name* DCBdsn	Specifies the name of the data set to be used as a model for allocating new data sets. Specifying DCBDSN with no parameter value cancels the DCBDSN specification. The equal sign (=) is optional in this case. This parameter has no default value.
DIrectory=*nnn* DIrectory	Specifies the number of directory blocks to be allocated for the directory of a PDS. This parameter applies only to the MKDIR and LMKDIR subcommands. Valid range of values is 1 through 16,777,215. The default is DIRECTORY=27. Specifying DIRECTORY=0 specifies that no directory value should be used when allocating the data set. The equal sign (=) is optional in this case.
ENDOFFSET=*num*	Stops reading at a certain offset. The offset specified by ENDOFFSET will be treated as an end of file. Used with OFFSET. ENDOFFSET must be specified on the side of the transfer that is inputting from a file. The SITE subcommand must be used for a GET and the LOCSITE subcommand for a PUT.
ENDRECNUM=*num*	Specifies the last record to be read from a file that was started by the RECNUM parameter. ENDRECNUM must be specified on the side of the transfer that is inputting from a file. The SITE subcommand must be used for a GET and the LOCSITE subcommand for a PUT.
ENDVLIST=*num*	Specifies the ending line number to be processed in the VLIST file. ENDVLIST must be specified on the side of the transfer that is inputting from a file. The SITE subcommand must be used for a GET and the LOCSITE subcommand for a PUT.
IO=(*io_name,info*)	Specifies a user-written Input/Output routine to be given control by CLIO/S as part of a data transfer process. Refer to Appendix A, “User Specified I/O Exits for CLFTP and CLPLINK” on page 251.
LENFIELDS	Specifies that each record of a transferred file contains a 4 byte record length field before the record and a 4 byte record length field after the record. A zero length record signals end-of-file (unless NOZEROEOF is specified), although it is not required at the end of the file. On MVS, CLFTP or CLPLINK add these fields to each record as it is read from the file and strip them off as each record is written to the file. On AIX these fields are read or written as part of the file. If improper length fields are in the file, the file can not be correctly transferred. **This parameter is for FORTRAN only.**
NOLENFIELDS	Turns off the LENFIELDS flag.
LENFIELDSADD	Adds LENFIELDS lengths to an ASCII, binary, or RDW file. **This parameter is for FORTRAN only.**
NOLENFIELDSADD	Turns off the LENFIELDSADD flag.
LENFIELDSSUB	Removes LENFIELDS lengths from an ASCII, binary, or RDW file. **This parameter is for FORTRAN only.**

Table 6 (Page 4 of 6). SITE and LOCSITE Parameters for MVS Hosts

Parameters	Description
NOLENFIELDSSUB	Turns off the LENFIELDSSUB flag.
LRecl=*nnn* LRecl	Specifies the logical record length of a newly allocated data set. Valid values are 0 through 32,760. The default is LRECL=260. Specifying LRECL=0 or LRECL indicates that no LRECL should be used when allocating the data set. The equal sign (=) is optional in this case.
MAXLRECL=*nnn*	Specifies the maximum number of bytes in any record. When using the CLIO/S tape feature user I/O routine (FCFFCLTP), if LRECL=*nnn* is specified in the ALLOC=*string*, The LRECL value overrides the MAXLRECL value.
NULSTOBLANKS	Converts NULL characters to blanks as part of translation.
NONULSTOBLANKS	Turns off the NULSTOBLANKS flag.
OFFSET=*num*	The offset within the file at which CLFTP or CLPLINK should start reading data. The OFFSET option allows reading only the end part of a file. OFFSET must be specified on the side of the transfer that is inputting from a file. The SITE subcommand must be used for a GET and the LOCSITE subcommand for a PUT.
PRImary=*nnn* PRImary	Specifies the amount of tracks, blocks, or cylinders for primary allocation. For allocating partitioned data sets, this is the amount that is allocated for the primary extent. For allocating sequential data sets, this is the maximum amount that is allocated for the primary extent. If a lesser amount is needed to hold the data being transferred, only the amount actually needed to hold the data is allocated. Valid range of values is 1 through 16,777,215. The default is PRIMARY=1. Specifying PRIMARY=0 indicates that no primary value should be used when allocating the data set. The equal sign (=) is optional in this case.
RDW	Allows variable length records to be transferred from MVS to a workstation and back without being changed.
NORDW	Turns off the RDW flag. The following files are not to be treated as RDW files.
RDWADD	Adds RDW lengths to an ASCII, binary, or LENFIELDS file.
NORDWADD	Turns off the RDWADD flag.
RDWSUB	Removes RDW lengths from a file, producing an ASCII, binary, or LENFIELDS file.
NORDWSUB	Turns off the RDWSUB flag.

Table 6 (Page 5 of 6). SITE and LOCSITE Parameters for MVS Hosts

Parameters	Description
RECfm=*format* RECfm	Specifies the record format of a data set. Valid record formats are: F, FB, FBS, U, V, VB, VBS, and VS. The characters used to specify these record formats have the following meanings: **Code** **Description** **F** Fixed record length **V** Variable record length **U** Undefined record length **B** Blocked records **S** Spanned records The default record format is variable length blocked records (VB). Specify RECFM with no value to indicate that no record format should be used when allocating the data set. The equal sign (=) is optional in this case.
RECIO	Preserves record boundaries as a file is being transferred.
NORECIO	Do not preserve record boundaries as a file is transferred.
RECNUM=*nnn*	Specifies the record number at which to begin reading the file. Record boundaries are determined by the physical records on the disk. RECNUM must be specified on the side of the transfer that is inputting from a file. The SITE subcommand must be used for a GET and the LOCSITE subcommand for a PUT.
SECondary=*nnn* SECondary	Specifies the number of tracks, blocks, or cylinders for secondary allocation. Valid range of values is 0 through 16,777,215. The default is SECONDARY=1. Specifying SECONDARY with no value or SECONDARY=0 indicates that no secondary value should be used when allocating the data set. The equal sign (=) is optional in this case.
STRIDE=(*strideJump* *[,strideCount*])	Specifies: *strideJump* - Indicates how many records are to be skipped in between each record that is read. *strideCount* - Indicates how many records should be read for each read operation. The default is 1. STRIDE must be specified on the side of the transfer that is inputting from a file. The SITE subcommand must be used for a GET and the LOCSITE subcommand for a PUT.
STRIPBLANKS	Deletes blanks from the end of ASCII records as they are being transferred.
NOSTRIPBLANKS	Turns off the STRIPBLANKS flag.
TRacks	Specifies that primary and secondary space allocations are in tracks. If you do not specify one of BLOCKS, CYLINDERS, or TRACKS, space is allocated in CYLINDERS.

Table 6 (Page 6 of 6). SITE and LOCSITE Parameters for MVS Hosts

Parameters	Description
Unit=*unit_type*	Specifies the DASD unit type (for example, 3380) for allocation of new data sets. If *unit_type* is not specified, the unit type used for allocation is set back to the system default.
USERdata=*string*	Provides a mechanism to pass a string of random data to a user-written conversion or I/O routine. The string is passed to the user routine as part of the ALLOCP parameter.
UTILITY=*utility_data*	Specifies information to be passed to **CTPALLOC()** in the utility parameter. See Appendix A, “User Specified I/O Exits for CLFTP and CLPLINK” on page 251.
VLIST=*num*	Indicates that multiple tapes and files are to be concatenated using VLIST. If VLIST=0, the input file is treated as a normal input file, tape, pipe, or raw volume. If VLIST is a number other than 0, the input file is treated as a list of tapes and/or files that should be concatenated to form one logical input file. *num* indicates the offset into the list of tapes and/or files at which we should start reading. (i.e., 1 specifies to start with the first tape or file in the list, 2 the second, ...) By setting *num* to a value greater than 1, you can read a list of tapes starting at some offset into the list. The list of files and tapes can have three types of entries: • A comment line starts with "#". • The name of a file starts with the word "file". • Site parameters that are in effect for all following "file" entries start with "site". VLIST must be specified on the side of the transfer that is inputting from a file. The SITE subcommand must be used for a GET and the LOCSITE subcommand for a PUT. For VLIST examples, see “VLIST Sample Files” on page 136.
VOLume=*volume_serial* VOLume	Specifies the volume serial number for allocation of new data sets. If *volume_serial* is not specified, the volume serial number used for allocation is set back to the system default.
ZEROEOF	Zero length records are treated as an end of file. ZEROEOF is ignored if LENFIELDS, LENFIELDSADD, or LENFIELDSSUB is not specified.
NOZEROEOF	Specifies that when CLFTP (or CLPLINK) reads a LENFIELDS format file it should not treat 0 length records as an end of file.

The AIX LOCSITE and SITE parameters are shown in Table 7 on page 121.

Table 7 (Page 1 of 5). SITE and LOCSITE Parameters for AIX Hosts

Parameters	Description
ADDZEROEOF	Adds a zero length record to the end of each transfer. This option is ignored if LENFIELDS, LENFIELDSADD or LENFIELDSSUB is not specified.
NOADDZEROEOF	Turns off the ADDZEROEOF flag.

ALLoc='*text*'
ALLoc+='*appended_text'*
	Specifies parameters to pass to dynamic device allocation. Use this parameter to allocate tape data sets and BatchPipes/MVS data sets. Specify *text* as a list of blank-separated allocation parameters inside single quotes (**'**), for example, ALLOC='UNIT=3480 VOLUME=TAPE01'. See Appendix E, "Dynamic Device Allocation on MVS" on page 303 for a list of supported parameters. When the ALLoc parameter is specified, all allocation parameters must be specified in the ALLoc parameter *text* rather than as individual values.
	
	If you specify `dsname=`*`name`* in the ALLOC= parameter text, you must specify a file name of `dd:` on the GET or PUT subcommand. If you specify a file name other than `dd:` on the GET or PUT subcommand, an allocation error results.
	
	The ALLOC parameter is specific to CLIO/S's CLFTP. It is not available in TCP/IP's FTP.
	
	Use ALLOC+='*appended_text'* to append allocation parameters to a previously-defined allocation string. If you want spaces between concatenated text strings, you must put them in the *text* or *appended_text* string. For example, to define the allocation string "UNIT=3480 VOLUME=TAPE01 MOD" in two parts, specify the allocation strings like this:
	
	`ALLOC='UNIT=3480 VOLUME=TAPE01'`
	`ALLOC+=' MOD'`
	
	Use ALLOC=*'allocation_data'* for the allocparm field of the **CTPALLOC()** call. See Appendix A, "User Specified I/O Exits for CLFTP and CLPLINK" on page 251 for more information.
	
	Note: If you use ALLOC= or ALLOC+= parameters, CLFTP ignores any other SITE or LOCSITE parameters.

BLKwrite	By default, CLFTP and CLPLINK write only to tape and diskette block devices. BLKWRITE specifies that CLFTP and CLPLINK should write to different types of block devices such disk drives.
NOBLKwrite	Resets the supported block device types to the default: tape and diskette block devices only.
BLocksize=*num*	For reading from or writing to a raw device. Specifies the fixed block size that the device requires. The block size you specify must match the size for which the device is configured otherwise, I/O errors occur when you attempt to read or write to the device. If you specify a blocksize of 0, CLFTP and CLPLINK attempt to determine the correct block size from AIX; this is usually possible if the device is a disk drive. When the correct block size cannot be determined, the device is assumed to use variable length blocks.

Table 7 (Page 2 of 5). SITE and LOCSITE Parameters for AIX Hosts

Parameters	Description
CONV=(*data1,data2*)	Specifies a user-written data conversion routine to be given control by CLIO/S as part of the data transfer process. Refer to Appendix B, "User Specified Data Translation for CLFTP and CLPLINK" on page 259.
ENDOFFSET=*num*	Stops reading at a certain offset. The offset specified by ENDOFFSET will be treated as an end of file. Used with OFFSET. ENDOFFSET must be specified on the side of the transfer that is inputting from a file. The SITE subcommand must be used for a GET and the LOCSITE subcommand for a PUT.
ENDRECNUM=*num*	Specifies the last record to be read from a file that was started by the RECNUM parameter. ENDRECNUM must be specified on the side of the transfer that is inputting from a file. The SITE subcommand must be used for a GET and the LOCSITE subcommand for a PUT.
ENDVLIST=*num*	Specifies the ending line number in the VLIST file being processed. ENDVLIST must be specified on the side of the transfer that is inputting from a file. The SITE subcommand must be used for a GET and the LOCSITE subcommand for a PUT.
IO=(*io_name,info*)	Specifies a user-written Input/Output routine to be given control by CLIO/S as part of a data transfer process. Refer to Appendix A, "User Specified I/O Exits for CLFTP and CLPLINK" on page 251.
LENFIELDS	Specifies that each record of a file that is transferred will contain a 4 byte record length field before the record and a 4 byte record length field after the record. A zero length record signals an end of file (unless NOZEROEOF is specified), although it is not required at the end of the file. These fields are read or written as part of the file. If improper length fields are in the file, the file cannot be correctly transferred. **This parameter is for FORTRAN only.**
NOLENFIELDS	Turns off the LENFIELDS flag.
LENFIELDSADD	Adds LENFIELDS lengths to an ASCII, binary, or RDW file. **This parameter is for FORTRAN only.**
NOLENFIELDSADD	Turns off the LENFIELDSADD flag.
LENFIELDSSUB	Removes LENFIELDS lengths from an ASCII, binary, or RDW file. **This parameter is for FORTRAN only.**
NOLENFIELDSSUB	Turns off the LENFIELDSSUB flag.
MAXBLocksize=*num*	For reading from or writing to a raw device. Specifies the maximum block size for a raw device. The maximum block size represents the maximum amount of data that CLFTP or CLPLINK reads from or writes to the device in a single statement. The default MAXBLOCKSIZE is 32,768 bytes. When using the CLIO/S tape feature user I/O routine (FCFFCLTP), MAXBLOCKSIZE specifies the record length for binary files. If LRECL=*num* is specified in the ALLOC= parameter, the LRECL value overides the MAXBLKSIZE value.

Table 7 (Page 3 of 5). SITE and LOCSITE Parameters for AIX Hosts

Parameters	Description
MAXLRECL=*nnn*	Specifies the maximum number of bytes in any record. When using the CLIO/S tape feature user I/O routine (FCFFCLTP), if LRECL=*nnn* is specified in the ALLOC=*string*, The LRECL value overrides the MAXLRECL value.
MOUNT=(*mount_shell, data*)	*mount_shell* is a shell that mounts a tape or processes a file before it is read or written. *mount_shell* will also unmount the tape or file. *data* is a string of characters that is passed to the MOUNT shell containing information for mounting or unmounting. The format of the input to the shell script is: *mount_shell* ALLOC|DEALLOC *file_name data* Where: ALLOC - The tape being allocated DEALLOC - The tape being deallocated *file_name* - The name of the file that is being used. *data* - The data that is specified in the MOUNT site parameter. When MOUNT is specified on a LOCSITE command, *file_name* is the name of the local file. When MOUNT is specified on a SITE command, *file_name* is the name of the remote file.
NULSTOBLANKS	Converts NULL characters to blanks as part of translation.
NONULSTOBLANKS	Turns off the NULSTOBLANKS flag.
OFFSET=*num*	The offset within the file at which CLFTP or CLPLINK should start reading data. The OFFSET option allows reading only the end part of a file. OFFSET must be specified on the side of the transfer that is inputting from a file. The SITE subcommand must be used for a GET and the LOCSITE subcommand for a PUT.
RAWwrite	By default, CLFTP and CLPLINK write only to tape and diskette raw devices. RAWWRITE specifies that CLFTP and
NORAWwrite	Resets the supported raw device types to the default: tape and diskette raw devices only. CLPLINK should write to different types of raw devices such disk drives.
RDW	Allows variable length records to be transferred from MVS to a workstation and back without being changed.
NORDW	Turns off the RDW flag. The following files are not to be treated as RDW files.
RDWADD	Adds RDW lengths to an ASCII, binary, or LENFIELDS file.
NORDWADD	Turns off the RDWADD flag.
RDWSUB	Removes RDW lengths from a file, producing an ASCII, binary, or LENFIELDS file.
NORDWSUB	Turns off the RDWSUB flag.

Table 7 (Page 4 of 5). SITE and LOCSITE Parameters for AIX Hosts

Parameters	Description
STRIDE=(*strideJump* [, *strideCount*])	Specifies: *strideJump* - Indicates how many records are to be skipped in between each record that is read. *strideCount* - Indicates how many records should be read for each read operation. The default is 1. STRIDE must be specified on the side of the transfer that is inputting from a file. The SITE subcommand must be used for a GET and the LOCSITE subcommand for a PUT.
STRIPBLANKS	Deletes blanks from the end of ASCII records as they are being transferred from MVS to AIX.
NOSTRIPBLANKS	Resets the STRIPBLANKS parameters.
RECIO	Preserves record boundaries as a file is being transferred.
NORECIO	Do not preserve record boundaries as a file is transferred.
RECNUM=*nnn*	Specifies the record number at which to begin reading the file. • For ASCII files - Record boundaries are determined by new line characters. • For Binary files - Record boundaries are determined by the MAXLRECL site parameter. • For RDW files - Record boundaries are determined by the length fields stored in the records. • For LENFIELDS files - Record boundaries are determined by the length fields stored in the records. RECNUM must be specified on the side of the transfer that is inputting from a file. The SITE subcommand must be used for a GET and the LOCSITE subcommand for a PUT.
USERdata=*string*	Provides a mechanism to pass a string of random data to a user-written conversion or I/O routine. The string is passed to the user routine as part of the ALLOCP parameter.
UTILITY=*utility_data*	Specifies information to be passed to **CTPALLOC()** in the utility parameter. See Appendix A, “User Specified I/O Exits for CLFTP and CLPLINK” on page 251.

Table 7 (Page 5 of 5). SITE and LOCSITE Parameters for AIX Hosts

Parameters	Description
VLIST=*num*	Indicates that multiple tapes and files are to be concatenated using VLIST. If VLIST=0, the input file is treated as a normal input file, tape, pipe, or raw volume. If VLIST is a number other than 0, the input file is treated as a list of tapes and/or files that should be concatenated to form one logical input file. *num* indicates the offset into the list of tapes and/or files at which we should start reading. (i.e., 1 specifies to start with the first tape or file in the list, 2 the second, ...) By setting *num* to a value greater than 1, you can read a list of tapes starting at some offset into the list. The list of files and tapes can have three types of entries: • A comment line starts with "#". • The name of a file starts with the word "file". • Site parameters that are in effect for all following "file" entries start with "site". VLIST must be specified on the side of the transfer that is inputting from a file. The SITE subcommand must be used for a GET and the LOCSITE subcommand for a PUT.
ZEROEOF	Zero length records are treated as an end of file. ZEROEOF is ignored if LENFIELDS, LENFIELDSADD, or LENFIELDSSUB is not specified.
NOZEROEOF	Specifies that when CLFTP (or CLPLINK) reads a LENFIELDS format file it should not treat 0 length records as an end of file.

Record Conversion Support

When LENFIELDS and RDW are used with AIX to AIX transfers, source files that are LENFIELDS or RDW files produce destination files of the same format. CLFTP and CLPLINK provide ways to override this default.

The following parameters are used with ASCII and binary transfers as follows:

ascii + **LENFIELDSADD**
: Reads an ASCII file and transforms it into a LENFIELDS file.

ascii + **RDWADD**
: Reads an ASCII file and transforms it into an RDW file.

binary + **LENFIELDSADD**
: Reads a binary file and transforms it into a LENFIELDS file. The length of each record will be determined by the MAXLRECL= site parameter.

binary + **RDWADD**
: Reads a binary file and transforms it into an RDW file. The length of each record will be determined by the MAXLRECL= site parameter.

ascii + **LENFIELDSSUB**
: Reads a LENFIELDS file and transforms it into an ASCII file.

ascii + **RDWSUB**
: Reads an RDW file and transforms it into an ASCII file.

binary + LENFIELDSSUB
Reads a LENFIELDS file and transforms it into a binary file. The record structure of the file will be lost during this transformation.

binary + RDWSUB
Reads an RDW file and transforms it into a binary file. The record structure of the file will be lost during this transformation.

LENFIELDS + RDWADD
Reads a LENFIELDS file and transforms it into an RDW file.

RDW + LENFIELDSADD
Reads an RDW file and transforms it into a LENFIELDS file.

TRANSLATE—Controlling Which Host Performs Data Conversion

Description
The TRANSLATE subcommand controls whether ASCII/EBCDIC data conversion, when necessary, is performed on the local or the remote host. This subcommand acts as a toggle; invoking it changes the setting. By default, data conversion occurs on the AIX host. If both hosts are AIX systems, the conversion takes place on the local host.

Syntax
TRANSlate

Parameters
The TRANSLATE subcommand has no parameters.

Usage Notes
Use this subcommand to off-load work from the MVS mainframe to the AIX workstation. Performing data conversion on MVS is often faster. However, off-loading the conversion to the AIX workstation can reduce the load on your MVS system.

Examples
The following example demonstrates the toggle effect of the TRANSLATE subcommand, and shows how to display the current setting with the STATUS subcommand:

```
clftp > translate
Translation will be done remotely.
clftp > translate
Translation will be done locally.
clftp > status

Connected to: riscnode1 (AIX)
ascii: On               Status Freq.: 4
Last Site String:
Transport Protocol: shared memory
Translation Site: Local
Maxbuflen = 500000      Firstbuflen = 100000    Secondbuflen = 100000
```

Transferring Data

You can use the following subcommands to transfer files and data sets from disk, tape, BatchPipes/MVS, and AIX raw devices:

- APPEND
- DELETE
- GET
- PUT

The following sections describe these CLFTP subcommands.

Warning: CLFTP maintains the attributes of a data set that is transmitted between a client and a server. However, under certain conditions CLFTP can truncate data records. If this happens, you might lose data. CLFTP can truncate data records under the following conditions:

- When you are creating a new data set on an MVS local host and the value of LRECL, as shown by the LOCSTAT subcommand, is less than actual length of lines in the received file, CLFTP truncates the file.

  You can encounter this situation when using GET.

- When you are creating a new data set on a remote MVS hosts and the value of LRECL, as shown by the STATUS subcommand, is less than the actual length of lines in the sent file, CLFTP truncates the file.

  You can encounter this situation when using PUT.

APPEND—Appending to a Remote File or Data Set

Description

To append a file on your local host to the end of a file on the remote host, use the APPEND subcommand. Both the file you append and the file to which you append must be sequential files, namely AIX files or MVS sequential data sets. APPEND does not support raw devices or MVS partitioned data sets.

Syntax

APpend *local_file* [*remote_file*]

Parameters

local_file Specifies the name of the file or sequential data set on your local host being appended to the file on the remote host.

remote_file Specifies the name of the file or sequential data set on the remote host to which the file is being appended. If the *remote_file* name is not specified, the *remote_file* name is the same as the *local_file* name.

You can specify an AIX file either by its fully qualified name (for example, /**u**/**colao**/**myfile**) or by a path relative to the current working directory (for example, **myfile**).

You can specify an MVS file name one of these ways:

'*fully***.***qualified***.***dsn***'** Enclose fully qualified data set names in single quotes.

qualifier.name Relative file names are qualified by the current working directory. For more information about working directories on MVS, see "Your MVS Working Directory" on page 94.

dd:*ddname* Indicates that the file can be read or written through *ddname*. *ddname* is the name of a DD statement known to CLFTP. You must define the DD statement and allocate the file, tape, or MVS BatchPipe in the JCL that starts CLFTP as a batch job or before invoking CLFTP from the TSO command line.

dd: Indicates that the file is allocated using the SITE or LOCSITE parameter ALLOC='dsname=*filename* ...'.

Usage Notes

If the remote host already has a file with the name specified by *remote_file*, the remote host overwrites the existing file. If the remote host does not have a file with the same name specified by *remote_file*, the remote host creates a new file.

To append a file to a file on the remote host, you must have defined a working directory on the remote host and you must have write privileges to the working directory and to the files in this directory.

Note: Under certain conditions CLFTP can truncate data records. For information about these conditions, see the warning on page 127.

Examples

Following are several examples of using the APPEND subcommand to append an AIX file to an MVS sequential data set.

```
clftp > append aixfile  bidwell.sds
226 Transfer complete(1044480 bytes).
    Elapsed time: 1.73 sec.      0.57 Meg/s
clftp > append big1 'bidwell.seqds'
226 Transfer complete(1044480 bytes).
    Elapsed time: 1.66 sec.      0.60 Meg/s
```

DELETE—Deleting Files On the Remote Host

Description

To delete files and partitioned data sets on the remote host, use the DELETE subcommand.

Syntax

DELETE *remote_file*

Parameters

remote_file Specifies the name of the file, sequential data set, or partitioned data set member to be deleted from the remote host.

You can specify an AIX file either by its fully qualified name (for example, /**u**/**colao**/**myfile**) or by a path relative to the current working directory (for example, **myfile**).

You can specify an MVS file name one of these ways:

'*fully.qualified.dsn***'**	Enclose fully qualified data set names in single quotes.
qualifier.name	Relative file names are qualified by the current working directory. For more information about working directories on MVS, see “Your MVS Working Directory” on page 94.
dd:*ddname*	Indicates that the file can be read or written through *ddname*. *ddname* is the name of a DD statement known to CLFTP. You must define the DD statement and allocate the file, tape, or MVS BatchPipe in the JCL that starts CLFTP as a batch job or before invoking CLFTP from the TSO command line.
dd:	Indicates that the file is allocated using the SITE or LOCSITE parameter ALLOC='dsname=*filename* ...'.

Usage Notes
To delete a file from the remote host, you must have write privileges to the remote working directory.

GET—Copying Files From the Remote Host

Description
To copy a file from the remote host to your local host, use the GET subcommand.

Syntax
Get *remote_file* [*local_file*] [**(Replace**]

Parameters
remote_file Specifies the name of the file, sequential data set, or partitioned data set member to be retrieved from the remote host.

If you specify an invalid name for the remote file and use the replace parameter, the local file is erased. Repeat the GET subcommand with a valid *remote_file*.

local_file Specifies the name of the local file created as a result of the GET subcommand. If *local_file* is not specified, the *local_file* name is the same as the *remote_file* name.

If the name specified for *local_file* is not acceptable to your local host, the file is not transferred.

Replace Causes a file on your local host to be overwritten, if it already exists. If the file already exists, and you do not use the replace parameter, the existing file is not overwritten. A message informing you of this is displayed.

If the file already exists and you specify replace, CLFTP overwrites the existing data set and retains the original characteristics.

You can specify an AIX file either by its fully qualified name (for example, /**u**/**colao**/**myfile**) or by a path relative to the current working directory (for example, **myfile**).

You can specify an MVS file name one of these ways:

'*fully.qualified.dsn***'**	Enclose fully qualified data set names in single quotes.
qualifier.name	Relative file names are qualified by the current working directory. For more information about working directories on MVS, see "Your MVS Working Directory" on page 94.
dd:*ddname*	Indicates that the file can be read or written through *ddname*. *ddname* is the name of a DD statement known to CLFTP. You must define the DD statement and allocate the file, tape, or MVS BatchPipe in the JCL that starts CLFTP as a batch job or before invoking CLFTP from the TSO command line.
dd:	Indicates that the file is allocated using the SITE or LOCSITE parameter ALLOC='dsname=*filename* ...'.

Usage Notes

To get a file from the remote host, you must have a defined working directory on that host and you must have read privileges to the files in this working directory.

Notes:

1. Under certain conditions CLFTP can truncate data records. For information about these conditions, see the warning on page 127.
2. A GET subcommand performed on an empty data set erases the contents of the existing local data set.

Examples

The following portion of a CLFTP session demonstrates using the GET subcommand to copy files. The first file copied is a text file that is a member of a partitioned data set on MVS; the file is transferred in ASCII mode. The second file is a data file contained in an MVS sequential data set; it is transferred in IMAGE mode to prevent translation from EBCDIC to ASCII.

```
clftp > ascii
000  -  File is ASCII
clftp > get 'bidwell.myprogs(servant)' servant.c
Transfer complete(538 bytes). Elapsed time: 0.27 sec.    0.00 Meg/s
clftp > binary
000  -  File is IMAGE
clftp > get 'bidwell.mydata' mydata
8250351 bytes transferred
Transfer complete(10606960 bytes). Elapsed time: 51.14 sec.   0.20 Meg/s
```

PUT—Copying Files to the Remote Host

Description

To copy a file from your local host to the remote host, use the PUT subcommand.

Syntax

Put *local_file* [*remote_file*]

Parameters

local_file Specifies the name of the file, sequential data set or PDS member on your local host being sent to the remote host.

remote_file Specifies the name that the delivered file is given on the remote host. If the *remote_file* name is not specified, the *remote_file* name is the same as the *local_file* name.

If the name specified for *local_file* is not acceptable, the file is not transferred but *remote_file*, if it exists, is erased.

You can specify an AIX file either by its fully qualified name (for example, /**u**/**colao**/**myfile**) or by a path relative to the current working directory (for example, **myfile**).

You can specify an MVS file name one of these ways:

'*fully***.***qualified***.***dsn***'** Enclose fully qualified data set names in single quotes.

*qualifier***.***name* Relative file names are qualified by the current working directory. For more information about working directories on MVS, see "Your MVS Working Directory" on page 94.

dd:*ddname* Indicates that the file can be read or written through *ddname*. *ddname* is the name of a DD statement known to CLFTP. You must define the DD statement and allocate the file, tape, or MVS BatchPipe in the JCL that starts CLFTP as a batch job or before invoking CLFTP from the TSO command line.

dd: Indicates that the file is allocated using the SITE or LOCSITE parameter ALLOC='dsname=*filename* ...'.

Usage Notes

If the remote host already has a file with the name specified by *remote_file*, the remote host overwrites the existing file. If the remote host does not have a file with the same name specified by *remote_file*, the remote host creates a new file.

To send a file to the remote host, you must have defined a working directory on the remote host and you must have write privileges to the working directory and to the files in this directory.

Note: Under certain conditions CLFTP can truncate data records. For information about these conditions, see the warning on page 127.

Examples

Following is an example of using the PUT subcommand. The tasks being performed include:

- Copying two text files (servant.c and master.c) from the workstation, creating two new members of a partitioned data set on MVS
- Copying a text file (fcfconf.c) from the workstation, creating a sequential data set on MVS
- Copying a file containing binary data (mydata) from the workstation, creating a sequential data set on MVS

```
clftp > ascii
000  -  File is ASCII
clftp > put servant.c 'bidwell.myprogs(servant)'
Transfer complete(538 bytes). Elapsed time: 0.28 sec.    0.00 Meg/s
clftp > put master.c 'bidwell.myprogs(master)'
Transfer complete(538 bytes). Elapsed time: 0.24 sec.    0.00 Meg/s
clftp > put /clio/bin/fcfconf.c 'bidwell.fcfconf'
Transfer complete(538 bytes). Elapsed time: 0.65 sec.    0.00 Meg/s
clftp >
clftp > binary
000  -  File is IMAGE
clftp > put /bbidwell/mydata 'bidwell.mydata'
Transfer complete(10606960 bytes). Elapsed time: 4.37 sec.    2.32 Meg/s
```

Disconnecting from the Host

To disconnect from the host, use the QUIT subcommand.

QUIT—Leaving the CLFTP Environment

Description

To disconnect from the remote host and end the CLFTP session, use the QUIT subcommand.

Syntax

QUIt

Parameters

The QUIT subcommand has no parameters.

Description

The QUIT subcommand ends the CLFTP session with the remote host and exits CLFTP on the local host. To establish a new session, use the CLFTP command as described in “The CLFTP Command” on page 92.

Starting CLFTP as a Batch Job

To submit CLFTP as a batch job, specify the CLFTP subcommands as input to the command. CLFTP reads input using the SYSIN DD statement and writes output using the SYSPRINT DD statement.

Place the CLFTP subcommands either inline following the SYSIN DD statement, as shown in Figure 17, or in a data set pointed to by the SYSIN DD statement. Specify the subcommands in the order you want them to happen.

```
//FCFTP1    JOB 1,PP.NUMBER.5648.129,MSGLEVEL=1,REGION=4096K
//FCFTP     EXEC PGM=CLFTP,PARM='riscnode3'
//SYSUDUMP  DD SYSOUT=4
//STEPLIB   DD DSN=SYS1.SFCFLOAD,DISP=SHR
//SYSPRINT  DD SYSOUT=4
//SYSIN     DD *
userid
password
echo
binary
locsite pri=10 sec=10
get /u/galina/datafile 'ds.name'
quit
//
```

Figure 17. JCL for Submitting CLFTP as a Batch Job

By default, the output data set shows the result of the subcommand's execution, but not the subcommand itself. Use the ECHO subcommand if you want the subcommands displayed.

Be sure to specify the QUIT subcommand as the last item in your input stream. This allows CLFTP to terminate normally.

CLFTP sets return code 0 in most cases. A return code of 0 indicates that the CLFTP command terminated normally; it does not reflect the success or failure of individual subcommands in your job stream. Examine your batch job's output data sets to determine whether individual subcommands were successful.

You can specify the user ID and password either in the JCL, as shown in Figure 17, or in your *user_ID*.**NETRC** file. If you place them in your NETRC file omit them from the JCL.

Transferring Data Across Intermediate Hosts

You can use CLIO/S FTP to transfer data across multiple hosts. Each host must have CLIO/S installed. If any of the hosts are MVS, PARSRV must be running on each target host.

For example, in the configuration of hosts in Figure 2 on page 18, ***riscnode3*** does not have a direct CTC connection to ***mvsnode***. However, users on ***riscnode3*** can use ***riscnode1*** as an intermediate CLIO/S host for transferring files from ***mvsnode***. To place a copy of **remotefile** from ***mvsnode*** in **localfile** on ***riscnode3***, using ***riscnode1*** as an intermediate host, riscuser3 issues these commands:

```
CLFTP riscnode1:mvsnode
clftp > get remotefile localfile
```

Specifying User and Password Information When Using Intermediate Hosts

The following table shows the different ways you can specify your user ID and password when you are crossing intermediate hosts. You can have different user IDs on the different systems.

If you want to...	Make sure that...
Bypass prompting for user ID and password on all hosts	The NETRC file on each host contains an entry for the downstream host, specifying both: • user ID • password
Supply all user IDs and passwords at the CLFTP prompt	The NETRC file on the local host must not contain an entry for the next host downstream. NETRC files on intermediate hosts, if they exist, are ignored.
Supply user ID and password for one or more intermediate hosts, while bypassing prompting for hosts farther downstream	The NETRC file on the local host must not contain an entry for the next host downstream. When CLFTP prompts for user IDs and passwords and you don't supply as many as there are hosts, the user IDs/passwords are applied from left to right (first intermediate through remote) across the list of hosts. Any host for which you do not provide a user ID and password must be listed in the NETRC file of the upstream host.

When working with intermediate hosts, any user ID whose password is not supplied either at the prompt or in the NETRC file must be defined to require no password; you can do this on AIX hosts, although it is not recommended by IBM.

CLFTP prompts once for all user IDs and once for all passwords. Specify multiple values as a colon-separated list, starting with the first intermediate host and proceeding through the remote host. When entering a list of passwords, you do not see either the passwords or the separating colons.

Note: Because CLFTP uses the colon (**:**) to separate passwords, a password that contains the colon character is best handled by placing all passwords in NETRC files.

Bypassing All Prompting

To bypass CLFTP prompting on all hosts in the previous example (`clftp riscnode1:mvsnode`), create these NETRC files:

riscnode3: **/u/riscuser3/.netrc**

```
machine riscnode1 login riscuser1 password secret1
```

riscnode1: **/u/riscuser1/.netrc**

```
machine mvsnode login mvsuser password secret
```

Each NETRC file contains both login and password.

Supplying All User IDs and Passwords at the CLFTP Prompt

Using the configuration set up in the previous example, the **/u/riscuser3/.netrc** file on ***riscnode3*** cannot contain any entries specifying machine ***riscnode1***. Because you supply two user IDs and passwords at the CLFTP prompt, the **/u/riscuser1/.netrc** file on ***riscnode1***, if it exists, is ignored.

The following example demonstrates how to supply multiple user IDs and passwords for all hosts:

```
$ clftp riscnode1:mvsnode
Welcome to clftp Version 1
Name (riscnode1:riscuser3): riscuser1:mvsuser
Password: secret1:secret           (note:  passwords are not echoed)
Connecting to : riscnode1
Connecting to : mvsnode
Remote system riscnode1:mvsnode connected with protocol TCP/IP:BMCA.
```

Supplying Some User IDs and Passwords at the CLFTP Prompt

The following example demonstrates how to supply multiple user IDs and passwords for the intermediate host, while using the NETRC file for information about the remote host:

```
$ clftp riscnode1:mvsnode
Welcome to clftp Version 1
Name (riscnode1:riscuser3): riscuser1
Password: secret1                   (note:  password is not echoed)
Connecting to : riscnode1
Connecting to : mvsnode
Remote system riscnode1:mvsnode connected with protocol TCP/IP:BMCA.
```

Using the configuration set up in the previous example, the **/u/riscuser3/.netrc** file on ***riscnode3*** cannot contain any entries specifying machine ***riscnode1***. However, the NETRC file on the intermediate host must contain both login and password, as shown here:

riscnode1: **/u/riscuser1/.netrc**

```
machine mvsnode login mvsuser password secret
```

CLFTP MVS Return Codes and AIX Exit Values

If you exit from CLFTP with the QUIT subcommand, the program returns an MVS return code (or AIX exit value) of 0. If the program terminates for another reason, it returns a positive integer, along with messages describing why it terminated. Return code 0 indicates that the CLFTP command terminated normally; it does not reflect the success or failure of individual subcommands.

CLFTP Error Messages

If you make an error when entering a CLFTP subcommand or if an internal error occurs during the execution of a CLFTP subcommand, you receive an error message explaining the problem.

The following examples show some representative error messages. The first command attempts to change the directory on an AIX host when the target directory does not exist. To correct this error, specify the name of a directory that does exist, or create the target directory if you need it.

```
 CLFTP >
cd /tmp/not_there
 301 Remote cd error.     errno : 2   No such file or directory
 43  Directory could not be changed.
```

The command in the following example attempts to create a file on the MVS host that is larger than the current data set extents. You can display the data set extents using the LOCSTAT subcommand. To recover from this error, use the LOCSITE subcommand to set larger PRIMARY and SECONDARY extents.

```
 CLFTP >
get /bbidwell/big100 'bidwell.big100'
 382 Local put buffer error.      errno : 33  Attempt to extend an
          non-extendable file
 12  Error while writing to file.
 621 Transfer Aborted.
```

CLFTP MVS Abends

MVS user abend code 768 occurs when a CLIO/S FTP process on MVS is terminated. Examine other messages that appear at the same time to determine why the process stopped. Possible causes include:

- You interrupted the AIX client CLFTP program using <**ctrl-c**>, halting the MVS server.
- A subtask of CLIO/S FTP (either client or server) has abended.

VLIST Sample Files

The following are examples of the VLIST parameter.

VLIST Sample File 1

The following file, **test1.vlist**, is an example of a file that can be used with the VLIST option.

```
--------Start file test1.vlist-----
# This file concatenates file f1, f2, f3, f4, and f5 together so that
# they can be transferred into a single file.
file f1
file f2
file f3
file f4
file f5
-------End file--------
```

Figure 18. A file that can be used with VLIST

Using CLFTP, this VLIST file can be used with the following subcommands.

```
>   site VLIST=1
>   get test1.vlist outFile
```

CLFTP concatenates remote files f1 through f5 into local file, *outFile*.

VLIST Sample File 2

The following file, **test2.vlist**, is an example of a VLIST file that contains site commands.

```
--------Start file test2.vlist-----
# This file concatenates several files and uses the SITE command
# to determine how the files should be read.
#
# Start reading file f1 at record number 10. Read until record number
# 200 and then ignore the rest of the records in the file. If there are
# less than 200 records in the file, get the remaining records from f2
# until 200 records have been processed (ie. 190 returned). If f1 has
# 200 or more records, no records will be read from file f2.
site recnum=10 endrecnum=200
file f1
file f2
# Reset recnum and endrecnum so we can start reading from file f3.
# Use a stride jump of 10 (skip 9 records between every record that
# is returned) and a stride count of 2 (return two records every time
# we return records). The stride will carry through files f4 and f5 as
# if they were part of file f3.
site recnum endrecnum stride=(10,2)
file f3
file f4
file f5
# Read files f6 and f7 with a stride of 100. Restart the stride
# count and jump before reading each file so that the first record
# returned from each file is record number 1.
site stride=(100)
file f6
site stride=(100)
file f7
# Use a stride of 100 on f8, but start at record number 200.
site recnum=200 stride=(100)
file f8
# Reset stride and recnum, and use conversion routine fcffcnv1 to
# convert the next records.
site stride recnum conv=(fcffcnv1)
file f9
# Continue to convert using fcffcnv1, but read using I/O
# routine fcffcltp.
site io=(fcffcltp) alloc="/dev/rmt0"
file /dev/rmt0
# Use normal I/O and conversion, and strip blanks from the input files.
site stripblanks io conv alloc
file f10
file f11
-------End file--------
```

Figure 19. An example of a VLIST file with site commands

- Using CLFTP, this VLIST file can be used with the following subcommands.

  ```
  >   site VLIST=1
  >   get test2.vlist outFile
  ```

  CLFTP concatenates the remote files into local file *outFile*.

- To start reading at file f6 the following subcommands can be used.

  ```
  >   site VLIST=6
  >   get test2.vlist outFile
  ```

CLFTP concatenates the remote files starting from f6 into local file *outFile*.

- To start reading at file f3 and end reading at file f9 the following subcommands can be used.

```
>   site VLIST=3 ENDVLIST=9
>   get test2.vlist outFile
```

  CLFTP will concatenate the remote files starting from f3 and ending at f9 into local file *outFile*.

Resolving Problems With CLIO/S File Transfer Program

If CLFTP is unable to establish a CLIO/S connection or create a server process on the remote system, you receive error messages similar to these when invoking CLFTP:

```
201 Unable to start server.
120 Clio_create detected an error.
```

To correct this problem, make sure that:

- The PARSRV process is running on the remote host.

  If your installation uses multiple PARSRV servers, make sure that you are using the appropriate one, as described in “If There Are Multiple PARSRVs” on page 91.
- On AIX, your user ID has permission to run the CLFTP program, and the executable is in a directory in your $PATH.
- On MVS, the CLFTP executable is resident and runnable.
- You can **ping** the remote hosts and any intermediate hosts.
- The user has an account on all target hosts.
- Your NETRC file is correct. Errors in the NETRC file can prevent login.
- There is a free port and the user is permitted to use it.
- Logins are currently allowed on the remote and intermediate machines.
- The PARCONF and PARCTC files are correctly defined.
- You are not having hardware errors related to your ESCON or BMCA/PCA adapter.

Chapter 8. The CLIO/S Cross System Pipe Link

This chapter describes the CLIO/S Cross System Pipe Link. It provides a basic overview of the pipe concept and describes scenarios in which the CLIO/S Cross System Pipe Link is used. It explains how to create CLIO/S pipes with the **clpmk** command and how to use the CLPLINK command to connect to remote data.

This chapter contains guide and reference information about the AIX **clpmk** command and the AIX and MVS command CLPLINK. The reference information about CLPLINK contains examples showing how to use CLIO/S Cross System Pipe Link in a variety of ways.

What Is a Pipe?

A pipe is a mechanism that allows two programs to communicate with each other. To one program, the pipe appears to be an output file; to the other it appears to be an input file. As the first program writes to the pipe, the second program reads from the pipe. The data written to the pipe is not written to disk or tape. Instead it passes from one program to the next through memory.

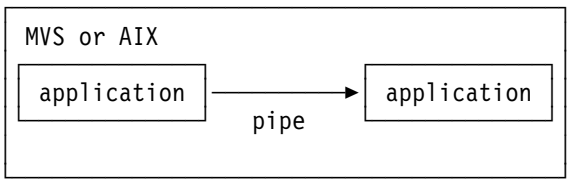


Figure 20. Piping eliminates intermediate output files.

Pipes let you overlap the execution of two programs when the output of the first is the input to the second. Without pipes, programs like this must run sequentially. The first program must finish writing its entire output file before the second program can begin reading that file as input. A pipe allows the programs to run in parallel. Every time the first program writes to the file, the data becomes available for the second program to read. This parallel processing cuts down on the total amount of time it takes for the two programs to finish as well as saving disk or tape space by avoiding the intermediate file.

Pipes are used with data that is read or written sequentially. If a program jumps back and forth through a file as it reads or writes it, it can't use pipes.

Transparent Access Across a CLIO/S Network

An important benefit of the CLIO/S Cross System Pipe Link is that it extends the pipe concept to remote data. Your application can, without change, read or write its data as if it were sequentially reading or writing a local file, while the CLIO/S Cross System Pipe Link transparently accesses the data on a remote system. For example, an application running on AIX can read data from MVS files or tapes and write output to MVS files or tapes. This lets you off-load processing to your workstation or SP, while exploiting the data management capabilities of MVS.

© Copyright IBM Corp. 1993, 1995, 1996

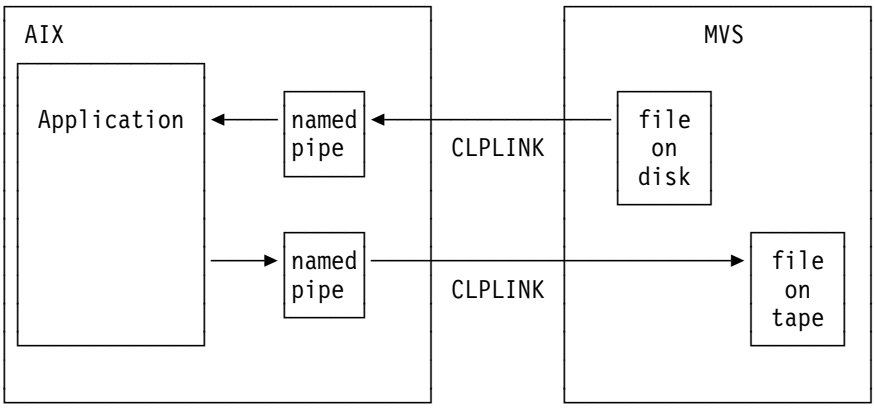


Figure 21. An AIX application transparently accessing files on a remote MVS system. You can use the CLIO/S Cross System Pipe Link with both named (as shown) and unnamed AIX pipes.

You can connect an AIX named pipe or MVS BatchPipe on your local system to an AIX named pipe or an MVS BatchPipe on a remote system. This lets you distribute your processing workload without relocating your data.

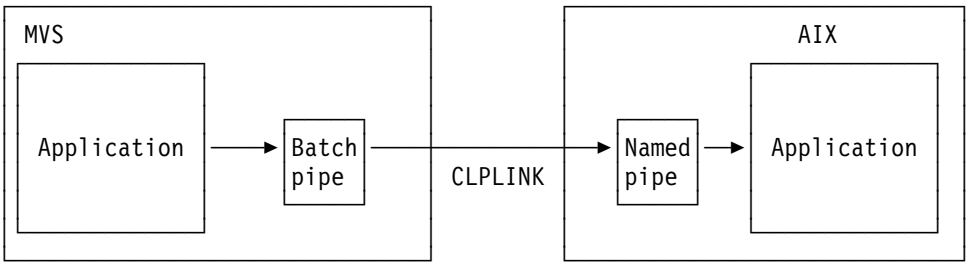


Figure 22. Piping across a CLIO/S network. The MVS application pipes data to an AIX application using named pipes.

CLPLINK can make the job of porting programs easier. Could this scenario happen at your company? Two applications run sequentially on MVS with the output of the first application being the input to the second. The second application is ported to the SP. Using the CLIO/S Cross System Pipe Link, the two applications can run in parallel. The output of the first (running on MVS) is piped to the second (running on AIX) and then the output of the second is piped back to an MVS file. Neither application needs to know whether it is working with local or remote data.

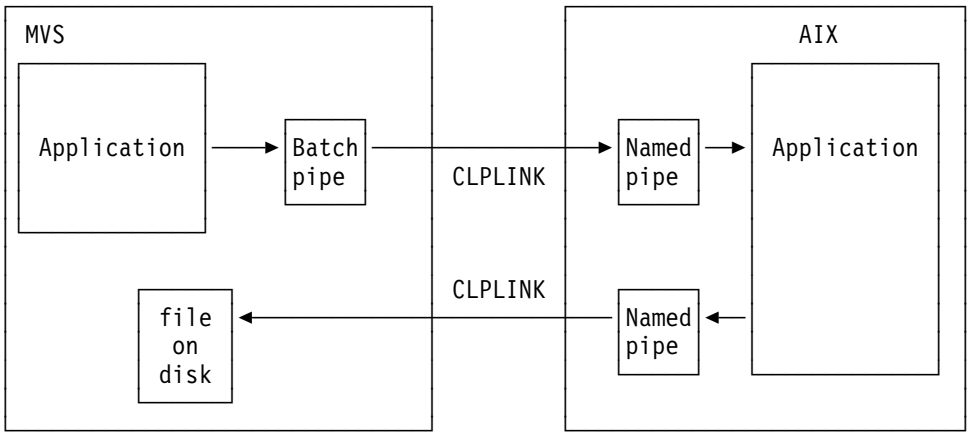


Figure 23. Piping across a CLIO/S network. The MVS application pipes data to an AIX application. Output created on AIX is saved in a file on MVS.

Connections can be established between these types of files and pipes:

MVS		AIX
BatchPipe		Unnamed pipe
File	CLPLINK	Named pipe
Tape		File
		Tape
		Raw volume

Using the CLIO/S Cross System Pipe Link

This section describes the tasks you perform to use the CLIO/S Cross System Pipe Link for:

- Copying files to and from remote hosts
- Piping remote files into an application or piping output to remote files
- Connecting pipes between remote hosts

This book assumes you known how to create and use BatchPipes/MVS. For more information, refer to *BatchPipes/MVS User's Guide and Reference*, GC28-1215.

If your installation runs multiple PARSRV servers on one host, you must set the CLIOPSPT environment variable to specify a PARSRV listening on other than the default port. If your installation runs multiple PARSRVs and has registered PARSRV in **/etc/services** (AIX) or **TCPIP.profile.TCPIP** (MVS), you must set the CLIOPSID environment variable to the name corresponding to the server you wish to use. See “Running Multiple PARSRVs” on page 41 for more information.

CLPLINK commands in this chapter are shown with the options listed vertically to help you read them; the AIX command line continuation character (\) is shown as appropriate. For example, this command line:

```
clplink -h remote_host -x remote_file -r local_file
```

is shown in this equivalent format:

```
clplink -h remote_host \
        -x remote_file \
        -r local_file
```

MVS commands are also shown using this convention, although the command line continuation syntax is not available on MVS.

Copying Files To and From Remote Hosts

To copy files to (–**r**) or from (–**w**) a remote host, simply use the CLPLINK command as shown here:

```
clplink -h remote_host \
        -x remote_file \
        -r local_file
```

For complete syntax of the CLPLINK command, see “clplink–Connecting To a CLIO/S Named Pipe” on page 148.

Accessing Remote Files From an Application

An application can pipe data to or from a remote host using one of:

- AIX unnamed pipes
- AIX named pipes
- BatchPipes/MVS

AIX Unnamed Pipes

You can use the CLIO/S Cross System Pipe Link with unnamed pipes on AIX by specifying *local_file* as stdin (–**w** –) or stdout (–**r** –). Consider this AIX command line that pipes data from the local file **inputdata** into the application **myprog**:

```
cat inputdata | myprog
```

To modify this to read a file from a remote host, use:

```
clplink -h remote_host -x remote_file -w - | myprog
```

AIX Named Pipes and BatchPipes/MVS

You can use the CLIO/S Cross System Pipe Link with AIX named pipes or BatchPipes/MVS to enable an application to pipe data to or from a remote host. These are the step you follow:

1. Create the pipe on the local host
2. Connect the pipe to the remote file
3. Start your application

You do these steps only on the local host. You don't need to log onto the remote host at all.

On MVS, create the pipe using BatchPipes/MVS. On AIX, create a CLIO/S pipe using the **clpmk** command. When you *read* from the pipe, CLPLINK *writes* the data. When you *write* to the pipe, CLPLINK *reads* the data.

This example shows the commands that pipe data from a remote file to an application on your local system:

```
clpmk -r local_pipe
clplink -h remote_host \
        -x remote_file \
        -w local_pipe
```

Now you start the application that reads from *local_pipe*.

Creative use of –**x dd:** with –**S "alloc='***allocation_parms***'"** can extend this example to connect to BatchPipes/MVS without logging onto MVS.

Note: You may get a slight response benefit on AIX when using –**w** or –**A** if you start the application *before* you issue the CLPLINK command. This eliminates a short initial delay required for pipe access synchronization.

Connecting Pipes Between Remote Hosts

You can connect pipes across a CLIO/S network by logging onto both hosts and performing these steps:

1. Create a pipe on each host
2. Connect each pipe to its data (connect the server side first)
3. Start the applications

You must start CLPLINK on the server side before starting CLPLINK on the client side.

This example shows the commands you use to make an AIX application pipe data into an MVS application. In this example AIX is the client and MVS is the server. On MVS, create an MVS BatchPipe named batchpipe. In the example, BatchPipes/MVS is the subsystem “BP01.” Select the port number on which the client and server will communicate (our example uses 12401), then start the CLPLINK server with this command:

```
clplink -p 12401     \
        -w batchpipe \
        -s "alloc='subsys=bp01 lrecl=260 recfm=vb'"
```

On AIX, issue these commands to create the CLIO/S pipe and start the CLPLINK client:

```
clpmk -w aixpipe
clplink -h mvshost   \
        -p 12401     \
        -r aixpipe
```

Now you start the applications on both hosts. It doesn't matter which application you start first; the one that reads from the pipe blocks until the other writes data into the pipe.

Creating a CLIO/S Pipe With clpmk

This section describes the complete syntax for using **clpmk** to create a CLIO/S pipe. You need to do this if you want to use the CLIO/S Cross System Pipe Link to pipe data into or out of an application on AIX using named pipes. If you are using unnamed pipes on AIX, as described in “AIX Unnamed Pipes” on page 144, or if you are using the CLIO/S Cross System Pipe Link for file transfer between systems, you do not need the **clpmk** command. If your application runs on MVS, use an MVS BatchPipe.

clpmk–Creating a CLIO/S Pipe

Syntax

clpmk –**r** *pipename*
clpmk –**w** *pipename*

Parameters

pipename Specifies the name of the pipe you want to create. You may specify the name as either a full or relative path name.

Note: You cannot create a pipe (for CLIO/S or any other purpose) in an AFS file system. The command appears to work but creates a regular file instead of the pipe special file.

–**r** Specifies that the local application reads from the pipe. Specify only one of –**r** or –**w**.

–**w** Specifies that the local application writes to the pipe. Specify only one of –**r** or –**w**.

Description

The **clpmk** command creates a CLIO/S pipe on AIX. The **clpmk** command performs the same function as the AIX **mkfifo** command, but **clpmk** additionally sets the appropriate read or write permissions on the special file. Only the user who issued the **clpmk** command has permission to access the file.

The **clpmk** command is available only on AIX. For information about the corresponding feature on MVS, BatchPipes/MVS, see the *BatchPipes/MVS User's Guide and Reference*, GC28-1215.

Use the CLIO/S pipe by connecting it to a remote system using the **clplink** command. The CLIO/S pipe remains after the connection established by **clplink** is broken. You can reuse it by issuing another **clplink** command to establish another connection.

When you no longer need the CLIO/S pipe special file, remove it using the AIX **rm** command, for example:

```
rm pipename
```

Usage Notes

You can use the **chmod** command to give others access to CLIO/S pipes you create with **clpmk**. Change only which users have access; **do not** change the read or write status of the pipe. For example, if you used **clpmk -r mypipe** to create this CLIO/S pipe for reading data:

```
pr-x------ 1 bbidwell cliodev      0 Feb 20 16:36 mypipe
```

then you can use this command to allow read access by members of the group **cliodev**:

```
chmod g+r mypipe
```

Warning

Do not change the read or write permissions for a CLIO/S pipe. The results are unpredictable.

Examples

1. To create a CLIO/S pipe represented by the file **inputpipe** (in the current directory) for reading read data from a remote system, issue this command:

   ```
   clpmk -r inputpipe
   ```

   The pipe is created with these default UNIX permission settings:

   ```
   pr-x------ 1 bbidwell cliodev      0 Feb 20 16:36 inputpipe
   ```

2. To create a CLIO/S pipe represented by the file **/tmp/outputpipe** for writing data to a remote system, issue this command:

   ```
   clpmk -w /tmp/outputpipe
   ```

   The pipe is created with these default UNIX permission settings:

   ```
   p-wx------ 1 bbidwell cliodev      0 Feb 20 16:36 /tmp/outputpipe
   ```

Files

/usr/lpp/fcf/bin/clpmk The path name of the executable for the command on AIX.

Related Topics

Commands: **chmod**, **clplink.**

Connecting to a CLIO/S Pipe With CLPLINK

Use the CLPLINK command to connect pipes, files, and tapes across a CLIO/S network. The CLIO/S Cross System Pipe Link uses an instance of the CLPLINK command on each host involved in the transfer, as shown in Figure 24. However, you don't always have to manually start both instances.

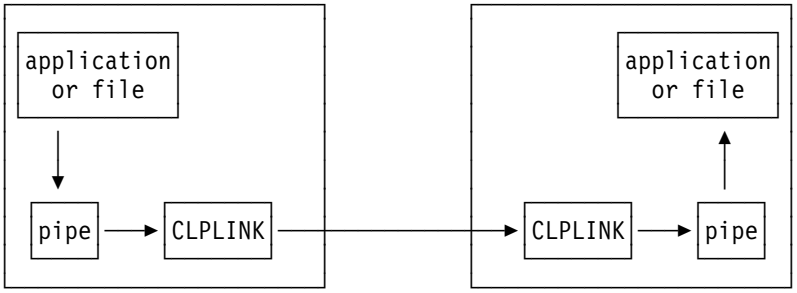


Figure 24. A CLPLINK instance on each host involved in a transfer.

In many common uses of CLPLINK, you invoke the CLPLINK command on the local system and it automatically starts the other instance on the remote host. Manually starting CLPLINK on both hosts lets you connect on a port of your choosing. This lets you do data set allocation using JCL on MVS.

The connection is automatically broken and the CLPLINK program terminates normally on both systems when either application closes the pipe. Normally,

whichever application is writing to the pipe closes first, sending a normal end-of-file indication to the other side. If you are using a CLIO/S pipe on AIX, the pipe remains after CLPLINK's connection is broken. You may reuse the CLIO/S pipe by issuing a CLPLINK command to connect the pipe to another location.

Pausing clplink

CLPLINK has the ability to pause a transfer. When a transfer is paused, the server side of the transfer is stopped, freeing all resources at the server, until the user restarts the transfer. At that time, the server is restarted. Pausing requires that the –**R** option is active on the CLPLINK.

The Application Linker has monitor functions that can pause and resume CLPLINK clients that are being managed by the Application Linker.

A **clsig30** shell script is provided to pause and resume a CLPLINK.

clplink–Connecting To a CLIO/S Named Pipe

Syntax

On the server:

clplink –**p** *port* –**rwA** *pipename* [*other_options*]

On the client:

clplink –**h** *host* [**:***host* ...] [–**U** *userid* [**:***userid* ...] [–**P** *password* [**:***password* ...]]]
–**p** *port*
–**rwA** *pipename*
[–**a** | –**b**] *other_options*

clplink –**h** *host* [**:***host* ...] [–**R**] [–**U** *userid* [**:***userid* ...] [–**P** *password* [**:***password* ...]]]
–**nx** *remote_data* [
–**x** *remote_data* [[–**S** **"***remote_site_string***"**] [–**T** *seconds*] [–**X** *string*]]
–**rwA** *pipename*
[–**a** | –**b**] *other_options*

Other options:

[–**f**] [–**O** *logname* | –**o**] [–**s** **"***local_site_string***"**] [–**v**]

Parameters

–**a** The file transferred over the pipe is an ASCII file. This option affects the files named on the –**A** *pipename*, –**r** *pipename*, or –**w** *pipename* option. This is the default file transfer mode. You can use this option to override a BINARY setting in the *userid***.CLPLINK.DATA** or **$HOME/.clplinkrc** defaults file.

By default, data conversion takes place on the AIX host. If both client and server are AIX systems, the conversion takes place on the local host. To change this, use the TRANSLATE statement in the **.clplinkrc** file as described in “Customizing CLPLINK Defaults” on page 161.

You may specify –**a** only on the client host (that is, the host on which you code the –**h** option).

Use only one of –**a** or –**b**. The default is –**a**.

–**A** *pipename* The name of the local file, tape, pipe, or raw volume that CLPLINK appends to. If *pipename* is on AIX, you cannot append to a tape or raw volume. Note that while CLPLINK *writes* when appending to the pipe, your application *reads* from it.

On AIX, *pipename* can specify a CLIO/S pipe previously created with the `clpmk -r pipename`. You can specify *pipename* as one of:

- A full path name (for example, /**tmp**/**datafile**)
- A name relative to the current directory (for example, **myfile**). The default working directory can be modifed using **$HOME/.clplinkrc**.

On AIX, CLPLINK can also write to stdout. To do this, specify *pipename* as a hyphen (–), for example, `clplink -A -`.

When appending MVS data with –**A**, you can specify *pipename* one of these ways:

'*fully.qualified.dsn***'**
: Enclose fully qualified data set names in single quotes. On AIX, escape the single quotes enclosing fully qualified data set names to get them past the shell.

dsname
: Specifies a file or data set name qualified by the file name prefix defined in *userid***.CLFTPINF** or as set in *userid***.CLPLINK.DATA**.

dd:*ddname*
: Indicates that the file can be read or written through *ddname*. *ddname* is the name of a DD statement known to CLPLINK. You must define the DD statement and allocate the file, tape, or MVS BatchPipe in the JCL that starts CLPLINK as a batch job or by invoking CLPLINK from the TSO command line.

  To append on MVS, the data set must have been allocated with DISP=MOD.

dd:
: Indicates that the file is allocated using the SITE parameter ALLOC='dsname=*filename* ...'. To append on MVS, specify SITE parameter ALLOC='dsname=*filename* MOD ...'.

You must specify exactly one of –**r**, –**w**, or –**A**.

–**b** The file transferred over the pipe is a binary file. If –**b** is not specified, the file specified with –**A** *pipename*, –**r** *pipename*, or –**w** *pipename* is read or written as an ASCII file.

You may specify –**a** only on the client host (that is, the host on which you code the –**h** option).

Use only one of –**a** or –**b**. The default is –**a**.

–**f** On AIX, specifies that the CLPLINK program should run in the foreground. CLPLINK normally runs in background without a controlling terminal.

This flag is ignored on MVS.

–**h** *host*
–**h** *host*[**:***host* ...]
Specifies the remote *host* where your data resides. Specifying a single *host* indicates that your local host has a direct connection to the remote system.

You can let CLIO/S route your request through an intermediate CLIO/S system by specifying a colon-separated list of host names ([**:***host*...]). Use this to access hosts in your CLIO/S network that do not have a direct connection to your local host. The first *host* in the list is the intermediate node directly connected to your local node; the last *host* is the remote node on which the data resides.

If you omit –**h**, you must specify –**p**. If you use –**h**, you must specify one of –**n**, –**p**, or –**x**.

–**o** Indicates that messages should be written to the file *pipename***.log** (where *pipename* is the name specified in the –**r**, –**w**, or –**A** option). If neither –**o** nor –**O** is specified, messages are printed to stdout and stderr. You may specify only one of –**o** nor –**O**.

–**n** *remote_data*
Indicates that the local instance of CLPLINK should start the remote instance of CLPLINK for appending to a remote file. *remote_data* is the name of the file to which the remote instance of CLPLINK appends. Use –**n** only when reading a local file with –**r**.

If the local file specified by –**r** is a pipe, CLPLINK does not start the remote instance of CLPLINK until the local application reads from or writes to the pipe.

You must specify exactly one of –**n**, –**p**, or –**x**.

When you use –**n**, CLPLINK directs the PARSRV server on the remote host to start the remote instance of CLPLINK. If your installation has multiple PARSRVs on the remote host, you must set the CLIOPSPT environment variable to the appropriate PARSRV port number before invoking the CLPLINK command. See “Running Multiple PARSRVs” on page 41 for more information.

If the remote PARSRV is registered in /**etc**/**services** (AIX) or in **TCPIP.profile.TCPIP** (MVS) you must set the CLIOSPID environment variable to the server/port combination you want to use.

–**O** *logname* Indicates that messages should be written to a file with the specified name. If neither –**o** or –**O** is specified, messages are printed to stdout and stderr. You may specify only one of –**o** or –**O**.

–**p** *port*
: If –**h** is specified, *port* is the port number of the CLPLINK instance on another host. The local instance of CLPLINK connects to the remote instance on this port.

  If –**h** is not specified, *port* is the port for the local instance of CLPLINK. Another instance of CLPLINK can contact the local instance of CLPLINK on this port.

  If you specify –**p** on one system, you must explicitly start a CLPLINK program with the same port number on both the local and remote host. Intermediate hosts, if any, are started automatically using a dynamically generated port number.

  You can omit –**p** if you use –**h** and want CLPLINK on your local system to automatically start the CLPLINK program on the remote system. In this case, you must use –**n** or –**x**.

  You must specify exactly one of –**p**, –**n** or –**x**.

  Note: CLPLINK's port number, specified with –**p** *port* is the port over which client and server instances of CLPLINK communicate; you select the port number each time you start a client and server pair. This is a different port from PARSRV's port number, which you may have to set with the CLIOPSPT environment variable if PARSRV is not using the default. PARSRV's port number is determined by the person who started the PARSRV server.

–**P** *password*
–**P** *password*[**:***password* ...]
: A string of passwords associated with the user IDs specified in the –**U** option. The passwords should be separated by colons (:). This flag is valid only with –**U**. If you do not specify the password with the –**P** option, CLPLINK prompts for it.

–**r** *pipename*
: The name of the local file, tape, pipe, or raw volume that CLPLINK reads from. Note that while CLPLINK *reads* from the pipe, your application *writes* to it.

  On AIX, *pipename* can specify a CLIO/S pipe previously created with the `clpmk -w pipename`. You can specify *pipename* as one of:

  - A full path name (for example, /**tmp**/**datafile**)
  - A name relative to the current directory (for example, **myfile**). The default working directory can be modifed using **$HOME**/**.clplinkrc**.

  On AIX, CLPLINK can also read from stdin. To do this, specify *pipename* as a hyphen (–), for example, `clplink -r -`.

  When reading MVS data with –**r**, you can specify *pipename* one of these ways:

  '*fully.qualified.dsn***'**
  : Enclose fully qualified data set names in single quotes. On AIX, escape the single quotes enclosing fully qualified data set names to get them past the shell.

dsname Specifies a file or data set name qualified by the file name prefix defined in *userid*.**CLFTPINF** or as set in *userid*.**CLPLINK.DATA**.

dd:*ddname* Indicates that the file can be read or written through *ddname*. *ddname* is the name of a DD statement known to CLPLINK. You must define the DD statement and allocate the file, tape, or MVS BatchPipe in the JCL that starts CLPLINK as a batch job or by invoking CLPLINK from the TSO command line.

To append on MVS, the data set must have been allocated with DISP=MOD.

dd: Indicates that the file is allocated using the SITE parameter ALLOC='dsname=*filename* ...'. To append on MVS, specify SITE parameter ALLOC='dsname=*filename* MOD ...'.

You must specify exactly one of –**r**, –**w**, or –**A**.

–**R** This option indicates that recovery should be done if the server side of a transfer goes down. Recovery consists of restarting the server side of the transfer and restarting the transfer from the point at which it stopped. The following restrictions apply to recovery:

- CLPLINK can recover only if the client side (the side which specified –**h**) of the transfer remains running. If the client goes down for any reason, the server side can not start the client side.
- CLPLINK can recover only if the –**x** option is being used.
- If CONV=(*conv_name...*) conversion of data is being done at the server, and the conversion routine returns **FCFFNoRecov** the first time it is called, recovery can not be used. This is because there is no way to make sure that at recovery time the conversion routine will be able to start its conversion at a random point in the middle of the file.
- If the server is calling an IO=(*io_name...*) function to do I/O, and that I/O routine does not support recovery, CLPLINK can not support recovery on that transfer.
- CLPLINK can recover only if the client is running on an AIX system.
- Recovery will not be successful when the server is reading from or writing to a pipe or an AIX tape.

–**s "***local_site_string***"**

local_site_string is used to set file characteristics for the local side of the pipe. Enclose *local_site_string* in double quotation marks (**"**) if it contains the space character. For example:

```
-s "primary=10 sec=10"
```

See “SITE and LOCSITE Parameters” on page 115 for information on the valid values for *local_site_string*.

This is an optional parameter.

–**S "***remote_site_string***"**
remote_site_string is be passed to the remote CLPLINK server to set file characteristics on the remote side of the pipe. Enclose *remote_site_string* in double quotation marks (**"**) if it contains the space character. For example:

```
-S "primary=10 secondary=10"
```

See “SITE and LOCSITE Parameters” on page 115 for information on the valid values for *remote_site_string*. This flag is valid only with –**n** or –**x**.

–**T** *seconds* Indicates the number of seconds the local instance of CLPLINK should wait for the remote instance to start. The default time-out value of 30 seconds can be modified using the CLIOTMO environment variable.

This parameter is valid only for client instances of CLPLINK. Use –**T** only when you specify –**n** or –**x** to let CLPLINK start the remote instance of CLPLINK, or when you are using intermediate hosts. Otherwise, –**T** is ignored.

–**U** *userid*
–**U** *userid* [**:***userid* ...]
The user ID or user IDs associated with the host or hosts specified on the –**h** flag. Each user ID is used to start a CLPLINK instance on the corresponding host in the list of *host*s. User IDs should be separated by colons (**:**). If –**U** is not specified and there is no entry in the **.netrc** file for the first host in the chain, CLPLINK prompts for user IDs and passwords.

–**v** Indicates that verbose messages should be written. This lets you generate detailed information about what is going on within the CLPLINK program. See the description of –**o** and –**O** for information about capturing the output in a file.

–**w** *pipename* The name of the local file, tape, pipe, or raw volume that CLPLINK writes to. Note that while CLPLINK *writes* to the pipe, your application *reads* from it.

On AIX, *pipename* can specify a CLIO/S pipe previously created with the `clpmk -r pipename`. You can specify *pipename* as one of:

On AIX, CLPLINK can also write to stdout. To do this, specify *pipename* as a hyphen (–), for example, `clplink -w -`.

When writing MVS data with –**w**, you can specify *pipename* one of these ways:

'*fully.qualified.dsn***'**
Enclose fully qualified data set names in single quotes. On AIX, escape the single quotes enclosing fully qualified data set names to get them past the shell.

dsname Specifies a file or data set name qualified by the file name prefix defined in *userid***.CLFTPINF** or as set in *userid***.CLPLINK.DATA**.

dd:*ddname* Indicates that the file can be read or written through *ddname*. *ddname* is the name of a DD statement known to CLPLINK. You must define the DD statement and allocate the file, tape, or MVS BatchPipe in the JCL that starts CLPLINK as a batch job or by invoking CLPLINK from the TSO command line.

To append on MVS, the data set must have been allocated with DISP=MOD.

dd: Indicates that the file is allocated using the SITE parameter ALLOC='dsname=*filename* ...'. To append on MVS, specify SITE parameter ALLOC='dsname=*filename* MOD ...'.

You must specify exactly one of –**r**, –**w**, or –**A**.

–**x** *remote_data*

Indicates that the local instance of CLPLINK should start the remote instance of CLPLINK. *remote_data* is the name of the file, tape, pipe, or raw volume that the remote instance of CLPLINK is to access.

If the local file specified by –**r**, –**w**, or –**A** is a pipe, CLPLINK does not start the remote instance of CLPLINK until the local application reads from or writes to the pipe.

When you use –**x** with –**r**, if *remote_data* already exists, it is overwritten. To append to a remote file, use the –**n** option.

You must specify exactly one of –**n**, –**p** or –**x**.

When you use –**x**, CLPLINK directs the PARSRV server on the remote host to start the remote instance of CLPLINK. If your installation has multiple PARSRVs on the remote host, you must set the CLIOPSPT environment variable to the appropriate PARSRV port number before invoking the CLPLINK command. See “Running Multiple PARSRVs” on page 41 for more information.

If the remote PARSRV is registered in /**etc**/**services** (AIX) or in **TCPIP.profile.TCPIP** (MVS) you must set the CLIOSPID environment variable to the server/port combination you want to use.

When accessing MVS data with –**x**, you can specify *remote_data* one of these ways:

'*fully.qualified.dsn***'**

Enclose fully qualified data set names in single quotes. On AIX, escape the single quotes enclosing fully qualified data set names to get them past the shell.

dsname Specifies a file or data set name qualified by the file name prefix defined in *userid***.CLFTPINF** or as set in *userid***.CLPLINK.DATA**.

dd:*ddname* Indicates that the file can be read or written through *ddname*. *ddname* is the name of a DD statement known to CLPLINK. You must define the DD statement and allocate the file, tape, or MVS BatchPipe in the JCL that starts CLPLINK as a batch job or by invoking CLPLINK from the TSO command line.

To append on MVS, the data set must have been allocated with DISP=MOD.

dd: Indicates that the file is allocated using the SITE parameter ALLOC='dsname=*filename* ...'. To append on MVS, specify SITE parameter ALLOC='dsname=*filename* MOD ...'.

–**X** *string* Specifies a string that is passed in the CLPLINKX environment variable to the JCL generation function of PARSRV. This variable can be examined during JCL generation on MVS to allow generation of special purpose JCL based on the value of CLPLINKX. (See Appendix D, “Controlling How CLIO/S Servers Are Started” on page 279 for more information.)

You can specify this option only with –**n** or –**x**. It is ignored if the remote instance is not running on MVS.

Usage Notes

1. To invoke CLPLINK from a TSO session, the CLIO/S load library must be allocated under DD name EDCMTF and added to your system STEPLIB concatenation. You can do this for your TSO session using the command:

   ```
   ALLOC DD(EDCMTF) DSN(SYS1.SFCFLOAD) SHR REUSE
   ```

   A system administrator can do this for all users, as described in “Enabling TSO Access to the CLFTP and CLPLINK Commands” on page 44.

2. CLPLINK automatically breaks the CLIO/S connection between hosts when either the client or server side closes the pipe. Normally, the side that writes to the pipe closes first, sending an end-of-file indication to the reading side.

3. If either the client or server is running on MVS, there must be a PARSRV server running on the MVS system. If both client and server are AIX systems, the remote instance of CLPLINK is started with **rexec** and a PARSRV server is not required on the remote host.

   If your installation uses more than one PARSRV server, you must set the CLIOPSPT environment variable to the appropriate port number. For example, to use PARSRV running on port number 1955 for establishing connectivity between two CLPLINK programs, set the CLIOPSPT environment variable to the value 1955. The port number used by PARSRV is different from the port number specified with the –**p** parameter.

   If your installation runs multiple PARSRVs and has registered PARSRV in /**etc**/**services** (AIX) or **TCPIP.profile.TCPIP** (MVS), you must set the CLIOPSID environment variable to the name corresponding to the server you wish to use.

4. You can put personal defaults for CLPLINK in the file **$HOME/.clplinkrc** on AIX or *userid***.CLPLINK.DATA** on MVS. For more information, see “Customizing CLPLINK Defaults” on page 161.

5. TSO users need MOUNT permission to use CLPLINK to read or write tapes on MVS. On systems using RACF, the user must be authorized to the resource class TSOAUTH, profile MOUNT, with at least READ permission.
6. CPLINK uses QSAM for file I/O. Files that cannot be processed by QSAM are not read or written by CPLINK.

Examples

1. Here's an easy way for an AIX user to copy the MVS tape data set TAPE.FILE to the AIX file /**data**/**file**. MVS dynamically allocates a tape drive for the tape data set, using allocation parameters passed in the –**S** option. Detailed informational messages about the file transfer are collected in the file **msgout** in the current directory in AIX. On the AIX system, issue this command:

   ```
   clplink -h mvshost \
           -x tape.file \
           -S "alloc='vol=vol1 unit=3480 sl'" \
           -w /data/file \
           -O msgout \
           -v
   ```

   You don't issue any commands on the remote system.

   Because –**U** and –**P** are not specified, CLPLINK looks for mvshost in **.netrc** file. If the host isn't listed there, CLPLINK prompts for user ID and password.

2. To cause the AIX application **myprog** to read its data from the 'FERN.DATA' data set on the MVS system **mvshost**, issue this command:

   ```
   clplink -h mvshost -x \'fern.data\' -w - | myprog
   ```

   You don't issue any commands on the remote system.

3. To pipe the MVS data set 'FERN.DATA' into the AIX application **myprog** using an AIX named pipe, issue these commands on the AIX system:

   ```
   # Create the CLIO/S pipe
   clpmk -r pipe1

   # Connect the pipe to MVS file
   clplink -h mvshost \
           -x \'fern.data\' \
           -w pipe1

   # Start your application
   myprog < pipe1
   ```

   You don't issue any commands on the remote system.

   Contrast this with example 2, which uses unnamed pipes on AIX, and therefore doesn't need the **clpmk** command.

4. This example shows how you run the AIX program **myprog**, reading input from the MVS tape data set '*userid*.INFILE' and writing output to the MVS tape data set '*userid*.OUTFILE'. On the AIX system, issue these commands:

```
# Create the CLIO/S pipes
clpmk -r /tmp/input
clpmk -w /tmp/output

# Connect the pipe to MVS data
clplink -h mvshost \
        -x infile \
        -S "alloc='vol=vol1 unit=3480 sl'" \
        -w /tmp/input

clplink -h mvshost \
        -x outfile \
        -S "alloc='vol=vol2 unit=3480 sl'" \
        -r /tmp/output

# Start your application
myprog < /tmp/input  > /tmp/output
```

You don't issue any commands on the remote system.

5. An MVS application using BatchPipes/MVS can communicate with an AIX application using named pipes, as shown in this example. The output of APPL1 (running on MVS) is piped to **appl2** (running on an SP). The output of **appl2** is piped back to the MVS data set '*userid*.APPL2(OUT)'. The local site parameters specified in –**s** enable CLPLINK to connect to the BatchPipes/MVS subsystem (BP01); additional parameters may be required. Neither application needs to know whether its data is local or remote.

   It doesn't matter which application you start first. The program reading from the pipe blocks until the other program has written data to the pipe.

```
————————AIX side————————          ————————MVS side————————

# Create the CLIO/S pipes
clpmk -r aixpipe
clpmk -w output

                                    # Pipe listens on port 50000
                                    clplink -r batchpipe \
                                            -p 50000     \
                                            -s "alloc='subsys=bp01
                                                       lrecl=260
                                                       recfm=fb'"

# Connect the input pipe to MVS
clplink -h mvshost \
        -p 50000   \
        -w aixpipe

# Connect output pipe to MVS
clplink -h mvshost    \
        -x appl2(out) \
        -r output

# Start second application          # Start application
appl2 < aixpipe                     appl1
```

6. In this example, the AIX program **myprog** sends its output to the MVS data set 'COLAO.OUT', explicitly passing the MVS user ID “colao” and its password

"figaro" on the command line (make sure no one is watching over your shoulder!).

```
# Create the CLIO/S pipe
clpmk -w pipe1

# Connect the pipe to MVS
clplink -h mvshost \
        -x \'colao.out\' \
        -S "pri=10 sec=10" \
        -r pipe1 \
        -U colao -P figaro

# Start your application
myprogram > pipe1
```

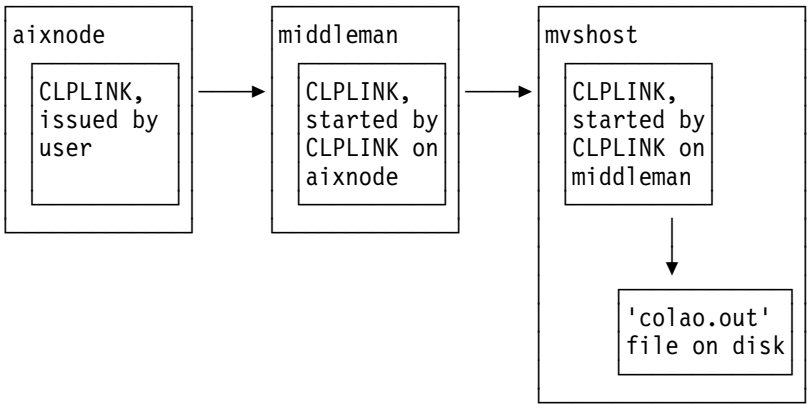
You don't issue any commands on the remote system.

7. This example is identical to example 6 on page 157 except that CLIO/S routes the request through an intermediate machine. CLIO/S connects to middleman, the intermediate host, using user ID "jbecky" and password "secret." The intermediate host connects to the host mvshost using user ID "colao" and password "figaro." On AIX, issue these commands:

```
# Create the CLIO/S pipe
clpmk -w pipe1

# Connect the pipe to MVS through
# an intermediate system
clplink -h middleman:mvshost \
        -x \'colao.out\' \
        -S "pri=10 sec=10" \
        -r pipe1 \
        -U jbecky:colao \
        -P secret:figaro

# Start your application
myprogram > pipe1
```

You don't issue any commands on the remote or intermediate system.

The following figure shows how each of the three instances of CLPLINK in this example are started. Notice that the user only issues one command, on the local host (AIX). The connection between both pairs of hosts happens over a port selected by CLPLINK and not known to the user who issues the original CLPLINK command. Contrast this with example 8 on page 159.



8. This example has the same end result as example 7 on page 158, but differs in how it is started. In both cases, the command is routed through the intermediate node middleman. In this example, an instance of CLPLINK is started manually on mvshost using port 50000 (for example, it could send output to an MVS BatchPipe instead of the data set 'COLAO.OUT').

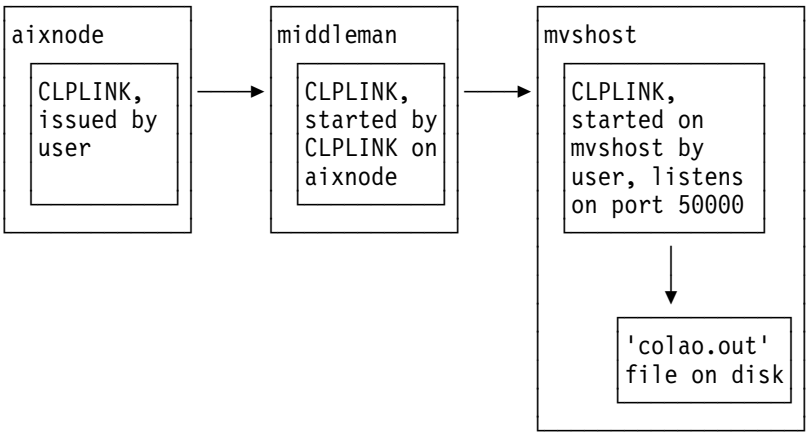
   You provide only the user ID for the intermediate host. You don't need a user ID on the remote host in this case, because the CLPLINK command (started by yourself or someone else) is listening at port 50000.

```
————————AIX side————————                    ————————MVS side————————

# Create a CLIO/S pipe
clpmk -w pipe1


                                             # Listen on port 50000
                                             # for someone to connect
                                             clplink -p 50000  \
                                                     -s "pri=10 sec=10" \
                                                     -w 'colao.out'

# Connect to MVS through
# intermediate system
clplink -h middleman:mvshost \
        -p 50000             \
        -U jbecky -P secret  \
        -r pipe1

# Start the application
myprogram > pipe1
```

   The following figure shows how each of the three instances of CLPLINK in this example is started. Notice that the user uses the CLPLINK command on both the local and remote systems. The intermediate CLPLINK is started automatically.

   Note: The user never manually starts the CLPLINK instance on an intermediate system.

   The connection between aixnode and middleman happens over a port selected by CLPLINK and not known to the user. In contrast, the connection between middleman and mvshost is made using port 50000, as specified by the **–p** option in both commands.



Contrast this example with example 7 on page 158.

9. To take advantage of the CLPLINK recovery feature, use the -R option of CLPLINK. When recovery is specified, if the server process is stopped for any reason, the client will continue running and attempt to restart the server and finish the transfer.

   For the command:

   ```
   >clplink -h charger -w zzz -x big10 -v -R
   ```

   The following is displayed.

   ```
   Daemon initialization is now set
   This instance of clplink can be paused and resumed with the
       'clsig30 26552' command.
   Connecting to : charger
   Remote system charger connected with protocol TCP/IP.
   Opening remote file or pipe big10.
   Remote file or pipe big10 is open. Starting to transfer data.
   Opening local file or pipe zzz.
   Local file or pipe zzz is open.
   1,872,440 bytes transferred, Elapsed time: 10.21 sec., 0.17 Meg/s
   ```

   At this point the CLPLINK server is stopped and the following is displayed.

   ```
   The clplink server has gone down.
   Recovery will be attempted every 60 seconds.
   Attempting clplink recovery.
   Connecting to : charger
   Remote system charger connected with protocol TCP/IP.
   Opening remote file or pipe big10.
   Remote file or pipe big10 is open. Starting to transfer data.
   clplink recovery was successful.
   2,117,935 bytes transferred, Elapsed time: 76.64 sec., 0.18 Meg/s
   5,018,351 bytes transferred, Elapsed time: 88.59 sec., 0.23 Meg/s
   8,655,281 bytes transferred, Elapsed time: 98.95 sec., 0.28 Meg/s
   All data transferred. Closing files.
   226 Transfer complete(10,444,810 bytes).
       Elapsed time: 124.19 sec.    0.17 Meg/s
   221 GoodBye.
   ```

Files

/usr/lpp/fcf/bin/CLPLINK The path name of the executable for the command on AIX.

SYS1.SFCFSAMP(FCFFCLPR)
SYS1.SFCFSAMP(FCFFCLPW) Sample JCL for starting CLPLINK on MVS.

Related Topics

Commands: **clpmk.**

Customizing CLPLINK Defaults

You can put personal defaults for CLPLINK in a file rather than using command line options each time you invoke the command. For example, if you frequently use CLPLINK to transfer binary files, you can change the default transfer mode for your user ID from ASCII to BINARY.

To establish personal defaults, create a file on the client host (the side where you issue **clplink -h** *hostname*). On MVS, the defaults file is *userid***.CLPLINK.DATA**. On AIX, the defaults file is **$HOME/.clplinkrc**.

The file contains a list of CLFTP subcommands. The syntax used in this file is identical to that used in the CLFTP default file, described in "Customizing CLFTP Defaults" on page 95. Each subcommand goes on a separate line. For SITE and LOCSITE, include all parameters on a single line. The following table lists the transfer characteristics that can be changed in the CLPLINK defaults file and shows the corresponding CLPLINK option:

Default Statement	Corresponding CLPLINK Option	Notes
ASCII	–**a**	Sets file transfer mode to ASCII for text data. For more information, see "ASCII—Changing the Data Transfer Type to ASCII" on page 111.
BINARY	–**b**	Sets file transfer mode to BINARY to prevent data conversion on transfers between machine types. For more information, see "BINARY—Changing the Data Transfer Type to Image" on page 112.
CD	(none)	Using CD to set the default remote directory makes it easier to specify file names for the –**x** option. For more information, see "CD—Changing the Directory on the Remote Host" on page 102.
HASH	–**v**	Controls the display of file transfer messages. For more information, see "HASH—Controlling the Display of File Transfer Status Messages" on page 98.
LCD	(none)	Using LCD to set the default local directory makes it easier to specify file names for the –**A**, –**r**, and –**w** options. For more information, see "LCD—Changing the Local Working Directory" on page 107.

Default Statement	Corresponding CLPLINK Option	Notes
LOCSITE	–**s**	Sets local site parameters. For more information, see “LOCSITE—Specifying Site Information to the Local Host” on page 113.
SITE	–**S**	Sets remote site parameters. For more information, see “SITE—Sending Site Specific Information to a Remote Host” on page 114.
TRANSLATE	(none)	Determines which host performs data translation. For more information, see “TRANSLATE—Controlling Which Host Performs Data Conversion” on page 126.

For example, the MVS user COLAO can set his default file transfer mode to binary by putting this line in the file COLAO.CLPLINK.DATA:

```
binary
```

If an AIX user tires of typing the site information about binary files to be transferred from AIX to MVS, she can establish defaults by placing these lines in the file **$HOME/.clplinkrc**:

```
binary
site pri=10 sec=10
```

Part 4. Application Programming Interfaces

Chapter 9. Using the CLIO/S Sockets Interface 165

Chapter 10. Using the Communication Manager Interface (CMI) 171

Chapter 11. Using the Distributed Processing Interface (DPI) 187

Chapter 12. Using the CLIO/S Tape Programming Interface and Support 225

© Copyright IBM Corp. 1993, 1995, 1996

Chapter 9. Using the CLIO/S Sockets Interface

The CLIO/S sockets interface provides a standards-based interface to the communication functions of CLIO/S.

CLIO/S provides high-speed, low-overhead communications between RS/6000 workstations and S/390 mainframes through a proprietary interface. The CLIO/S sockets interface provides a "front end" to CLIO/S in the form of the standard AF_INET reliable data stream interface. This data stream interface simplifies application development and improves code portability.

The CLIO/S sockets interface provides C subroutines. The design is optimized for large-volume data transmission. The CLIO/S sockets interface runs on both RS/6000 workstations and MVS.

Using the CLIO/S Sockets Interface

Applications access the CLIO/S sockets interface function by redefining specific socket calls. This is done by including the **fcfsock.h** header file, which transparently redefines socket calls. Applications must then be recompiled and linked with the CLIO/S sockets interface libraries.

The **fcfsock.h** header is located in the MVS data set **SYS1.SFCFHDRS(FCFSOCK)** and in the AIX file **/usr/lpp/fcf/inc/fcfsock.h**. The CLIO/S sockets interface library is **SYS1.SFCFLIB** on MVS/ESA; on AIX, the libraries are in the directory **/usr/lpp/clnettape/lib**.

The redefined socket calls are then part of the application, although they are not visible to it. They run in user application space. The redefined calls run a combination of TCP/IP and CLIO/S CMI calls to provide socket function.

For an example of a CLIO/S sockets interface application, see the sample programs FCFSERV (server) and FCFCLNT (client). These are located on MVS in in the partitioned data set **SYS1.SFCFSAMP**, and on AIX in the directory **/usr/lpp/fcf/sample**.

Figure 25 on page 166 illustrates how CLIO/S presents the socket interface to an application. In this example, the client application sends a request for data to the server application, which responds accordingly.

© Copyright IBM Corp. 1993, 1995, 1996

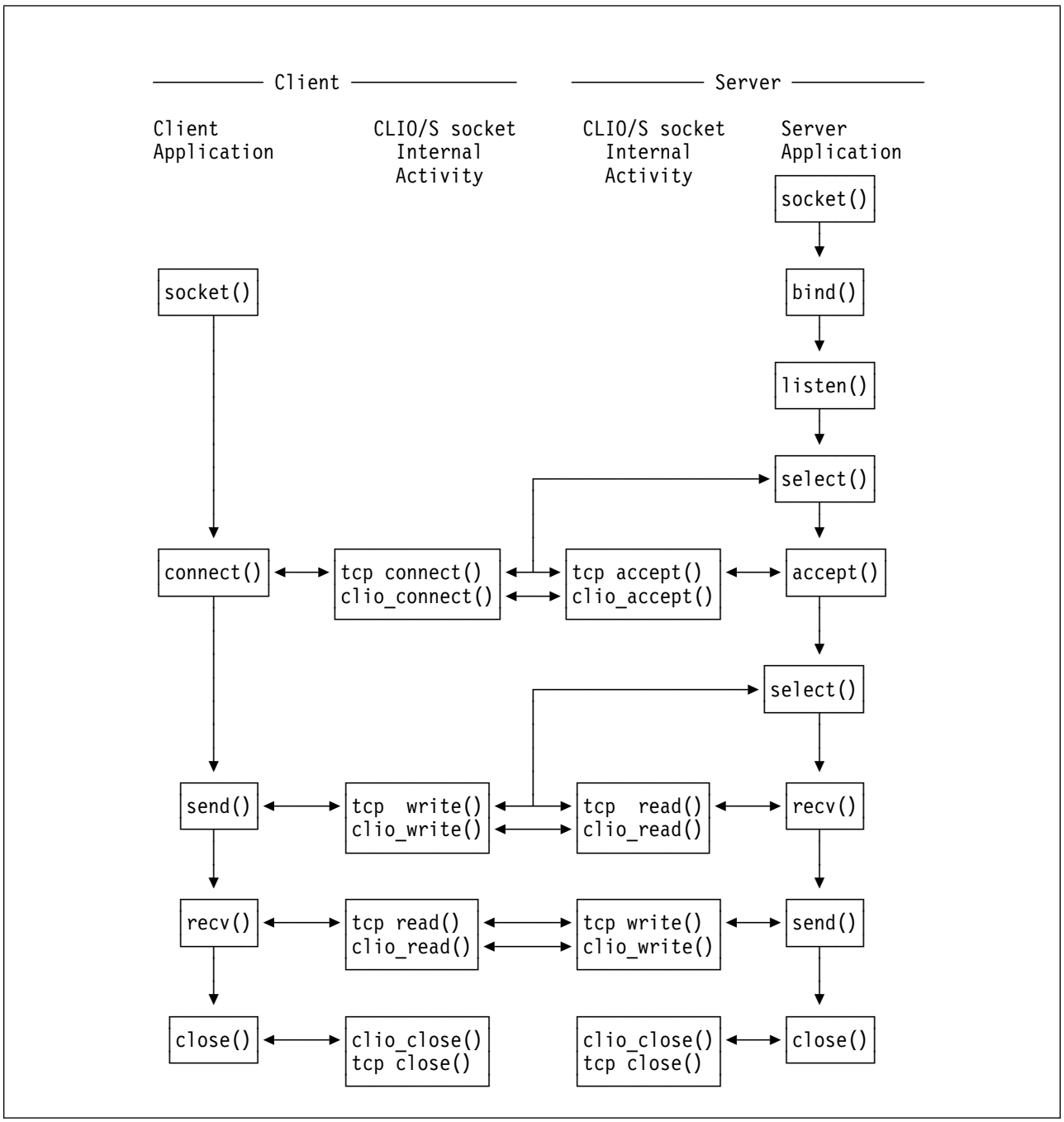


Figure 25. CLIO/S Sockets Interface Flow

Socket Calls Provided by the CLIO/S Sockets Interface

The following socket calls are redefined by the CLIO/S sockets interface during the compilation process. No changes are made in the parameters passed to the calls, with the exception of **recv** and **send**, in which the flag values are ignored. (Thus, for the CLIO/S sockets interface, **read** is identical to **recv**, and **write** is identical to **send**.) The returned values correspond to the values returned by TCP/IP for the same calls.

Socket Call	Redefined Call
accept()	fs_accpt()
close()	fs_close()
connect()	fs_conn()
read()	fs_read()
recv()	fs_read()
select()	fs_selct()
send()	fs_write()
write()	fs_write()
shutdown()	fs_shutd()

You can rename the functions in the application code, instead of including **fcfsock.h**. The advantage of renaming is it allows the same source file to use both TCP/IP sockets and CLIO/S sockets interface. However, the disadvantage is that it makes the code less portable.

If the CLIO/S sockets interface **select**, **read**, **recv**, **send**, **write**, and **close** calls are presented with a socket number that has not been previously passed to the CLIO/S sockets interface **accept** or **connect** calls, the call is transferred to TCP/IP, bypassing CLIO/S. For example, a connected socket that had been obtained by calling a function from a different source file is transferred to TCP/IP. This offers another way for processes to use both TCP/IP sockets and CLIO/S sockets interface.

Using Socket Calls Not Provided by the CLIO/S Sockets Interface

All other socket calls are ignored by the CLIO/S sockets interface and are handled by TCP/IP as usual. In some cases, the results of these calls may be unexpected unless careful consideration is given to the consequences of the changes caused by CLIO/S sockets interface. For example, the **setsockopt** and **getsockopt** calls work correctly. However, because the CLIO/S sockets interface does not use the application's socket for data transfer, the results of these calls may not be useful. The **socket**, **bind**, and **listen** calls work as expected.

Portability

This section describes how the CLIO/S sockets interface works with previously developed socket applications and socket applications in other environments.

Using Preexisting Socket Applications with the CLIO/S Sockets Interface

Most preexisting socket applications work with the CLIO/S sockets interface, but there may be exceptions. It is necessary to compare the particular functions and requirements of the application with the limitations noted in “Limitations of the CLIO/S Sockets Interface” on page 168. Inclusion of the **fcfsock.h** header and recompiling is also necessary.

Using the CLIO/S Sockets Interface in Other Environments

In general, socket applications that work with the CLIO/S sockets interface should work without the CLIO/S sockets interface; that is, with TCP/IP. Removal of the **fcfsock.h** header file, followed by recompiling, is necessary.

Limitations of the CLIO/S Sockets Interface

The CLIO/S sockets interface has some limitations. These limitations are:

1. The CLIO/S sockets interface provides only a socket, not the full suite of TCP/IP functions, such as FTP and NFS.
2. It is recommended that for corresponding **read** and **write** and for **send** and **recv**, the **read** and **recv** length should be greater than or equal to the length values of **write** and **send**. The CLIO/S sockets interface can buffer data if the **send** length is greater than the **recv** length, but there is a performance degradation for this buffering.
3. The **givesocket**, the **takesocket**, and the **selectex** calls are not supported on MVS.
4. CLIO/S supports socket numbers 0 through 255.
5. The **send** and **write** calls cause the issuing application to be blocked pending corresponding **recv** or **read** calls. Blocking does not occur on “short” transmissions. “Short” is installation-dependent when the underlying transmission protocol is TCP/IP and is less than or equal to 96 bytes otherwise. Similarly, the **recv** and **read** calls cause blocking pending corresponding **send** or **write** calls.
6. **read** and **write** calls must be synchronized between two parts of an application. When one part of an application includes a **write**, the other part of the application must include a **read**. CLIO/S does not support an application where both parts of the application **write** then both parts of the application **read**.

   The following example calls the **read** and **write** in the order supported by CLIO/S:

Application Part 1	Application Part 2
write(sock, ...);	read(sock, ...);
read(sock, ...);	write(sock, ...);

7. Flags on the **send** and **recv** commands are ignored.
8. The **shutdown** call has no effect on the CLIO/S sockets interface.
9. The maximum size for CLIO/S data transfer is 240 megabytes.
10. CLIO/S does not use the CLIO/S fast transfer for the following calls:
    - **sendto** and **recvfrom**
    - **writex** and **readx**
    - **writev** and **readv**
    - **writevx** and **readvx**
11. For CLIO/S socket inheritance, the child process must close the socket received from the **accept** call, using a CLIO/S **close**.

    If the parent wants to free the socket number for reuse in the parent process, it must close the socket using a TCP/IP **close** rather than a CLIO/S **close**. To do this, call TCP/IP **close** from code that was compiled without the **fcfsock.h** header file. Using a CLIO/S **close** in the parent process breaks the CLIO/S connection in use by the child process, resulting in possible data loss.
12. If a **read**, **write**, **send**, or **recv** call to a socket fails, subsequent **read**, **recv**, and **close** calls also fail.

13. The CLIO/S Sockets Interface does not support the User Datagram Protocol (UDP).

CLIO/S Sockets Interface Performance Hints

These hints help get the maximum performance from the CLIO/S sockets interface:

- The CLIO/S sockets interface is designed to transfer large blocks of data.
- In general, the larger the buffer size, the better your performance.
- Messages of less than 96 bytes are sent with TCP/IP and do not give the same performance benefit.

Linking and Compiling Sockets

On MVS, the **SYS1.SFCFSAMP(FCFSAMP)** program contains sample JCL that shows how to compile and link programs written in the CLIO/S sockets interface, the CMI, or the DPI. You can copy this file and modify it to compile and link your own programs.

On AIX, you can use the **make** command to help compile and link your programs. You can copy the sample makefile, **/usr/lpp/fcf/sample/Makefile**, and modify it to compile and link your own sockets programs.

Chapter 10. Using the Communication Manager Interface (CMI)

The CLIO/S Communication Manager Interface (CMI), like the CLIO/S sockets interface, is a single high-level interface to several high-speed communication technologies. This section describes how to use the CLIO/S CMI functions to perform tasks such as starting processes, transferring data, and performing data conversion. You can write complete distributed applications using just the CMI routines. You can also call the CMI routines from an application that uses the Distributed Processing Interface (DPI), which is described in Chapter 11, “Using the Distributed Processing Interface (DPI)” on page 187.

CMI Header File fcfcmi.h

The CMI functions are defined in the **fcfcmi.h** header file. The CMI interface is available only as C subroutines. These functions are not available in FORTRAN. The following list shows where to find the header file on MVS and AIX:

MVS **SYS1.SFCFHDRS(FCFCMI)**

AIX **usr/lpp/clnettape/inc**

The header file also defines common data types, including these important structures:

CLIO_t Contains information defining a CLIO/S process
MINFO Describes an entry in the PARCONF file
Hid_t Describes a process's internet address

About CLIO/S Data Conversions

When data is transferred between machines that have different data representations, CLIO/S automatically performs any necessary data conversions. The conversions are performed to and from a network format, which currently is IEEE, Big Endian, and ASCII. The conversions are always done on MVS. Between RISC systems, data conversion is not required.

When you call a CMI data transfer or conversion routine, you supply an argument (*dt*) defining the type of data you are transferring. This argument has one of the following values, as defined in **fcfcmi.h**:

0	**OPAQ**	Opaque. No data conversions are performed
1	**CHAR**	Character data
2	**INTE**	Integer data
3	**REAL**	Single-precision floating point
4	**DOUB**	Double-precision floating point
5	**LOGI**	Logical integer data

© Copyright IBM Corp. 1993, 1995, 1996

Using the CMI Functions

The following table lists the CMI subroutines and describes the function provided by each.

Subroutine	Description	Page
clio_accept	Accepting a CLIO/S Connection from Another Process	172
clio_async	Setting the Asynchronous Data Flag	173
clio_check	Checking the Number of Bytes Transmitted	173
clio_close	Closing a CLIO/S Connection	174
clio_connect	Requesting a CLIO/S Connection to Another Process	174
clio_create	Starting a Process on a Node	175
clio_end	Ending a CLIO/S Connection	176
clio_env	Getting the Environment	176
clio_exp	Setting the Environment	177
clio_h2i	Locating Machine Information Using an Internet Address	177
clio_init	Initializing CLIO/S	178
clio_kill	Stopping a Process on a Node	178
clio_l2n	Converting Data Representations, Local to Network	179
clio_n2l	Converting Data Representations, Network to Local	179
clio_m2i	Locating Machine Information Using a Machine Number	180
clio_map	Mapping From One Nickname Chain to Another	180
clio_n2h	Getting an Internet Address	181
clio_n2i	Locating a Machine by Symbolic Name	181
clio_n2m	Getting the Machine Information ID	181
clio_own	Allowing a Process to Change Ownership of a Connection	182
clio_pid	Returning a Process ID	182
clio_read	Reading Data over a CLIO/S Connection	183
clio_write	Writing Data over a CLIO/S Connection	184
tcp_accept	Accepting a TCP/IP Connection	184
tcp_connect	Establishing a TCP/IP Connection	185

clio_accept—Accepting a CLIO/S Connection from Another Process

Description

The **clio_accept** function establishes a CLIO/S connection between two processes. The other process must issue a corresponding **clio_connect**, and, therefore, must have an active TCP/IP socket connection between them. The request fails if nothing happens after *tmo* seconds.

Syntax

Language	Programming Syntax
C	**CLIO_t *clio_accept (int** *sd*, **int** *tmo* **)**

Where:

sd Is an input integer representing the socket descriptor.

tmo Is an input integer representing the timeout value in seconds.

Return Values

This function returns a pointer to a structure containing data for this connection. It returns a null pointer if the connection is not established.

clio_async—Setting the Asynchronous Data Flag

Description

The **clio_async** function sets or resets the asynchronous data flag. Asynchronous I/O is available only with the CTC protocols (BMCA/PCA or ESCON). For the other protocols, the I/O is always synchronous. See “clio_check—Checking the Number of Bytes Transmitted” for information.

When *asyncflag* is 1 (on), the function calls to **clio_read** or **clio_write** are asynchronous; the calls may return before the data has been completely transmitted.

Syntax

Language	Programming Syntax
C	**int clio_async (CLIO_t *** *cp*, **int** *asyncflag* **)**

Where:

cp Is a pointer to the **CLIO_t** structure

asyncflag Is a integer flag with the following possible values:

1 Sets the asynchronous flag on
0 Sets the asynchronous flag off

Return Values

This function returns 0.

clio_check—Checking the Number of Bytes Transmitted

Description

The **clio_check** function returns the number of bytes transmitted in an asynchronous I/O operation. **clio_check** waits for the last I/O on *cp* to complete. If more than one asynchronous I/O is performed to a given connection, **clio_check** is called automatically, and the number of bytes transmitted is lost.

Syntax

Language	Programming Syntax
C	**int clio_check (CLIO_t *_cp_)**

Where:

cp Is a pointer to the **CLIO_t** structure.

Return Values
The function returns the number of bytes transmitted.

clio_close—Closing a CLIO/S Connection

Description
The **clio_close** function closes a CLIO/S connection. The TCP/IP connection is not affected. If one of the CTC protocols is used both sides should call **clio_close** to ensure the required “handshaking” is done when closing the connection.

Syntax

Language	Programming Syntax
C	**int clio_close (CLIO_t *_cp_)**

Where:

cp Is a pointer to the **CLIO_t** structure.

Return Values
This function returns 0. If *cp* is null, –1 is returned.

clio_connect—Requesting a CLIO/S Connection to Another Process

Description
The **clio_connect** function establishes a CLIO/S connection between two processes. The other process must issue a corresponding **clio_accept** and there must be an active TCP/IP socket connection between them.

You can request a specific CLIO/S protocol by giving *proto* a suitable value. The suggested protocol is not necessarily honored; if an invalid or unsupported value, or zero, is supplied, the fastest protocol available is selected.

If the two processes execute on the same RS/6000 node, **clio_connect** attempts to use shared segments, if available.

If the protocol turns out to be TCP/IP, a new socket connection is established. If there is more than one TCP/IP network available to access a given node, you can select a particular one using the *hid* argument. To do this, the node must have multiple internet addresses. Then the PARCONF file can have multiple entries, one for each internet address, each specifying a different protocol.

Syntax

Language	Programming Syntax
C	**CLIO_t *clio_connect (int** *sd*, **int** *proto*, **Hid_t** *hid* **)**

Where:

sd Is an input integer representing a socket descriptor.

proto Is an input integer representing a protocol value as follows:

0 Dynamic
1 TCP/IP
2 SOC
3 BMCA/PCA
4 ESCON

hid Is a structure of type **Hid_t** that represents the internet address of the host. Nodes can be connected by more than one network. These nodes normally have a different internet address for each network. *hid* specifies the desired network. When *hid* is 0, the internet address is obtained from the socket.

Return Values

This function returns a pointer to a structure containing data for this connection. It returns a null pointer if the connection is not established.

clio_create—Starting a Process on a Node

Description

The **clio_create** function starts a new process. If both processes run under UNIX, CLIO/S uses **rexec** to start the process. Otherwise, a start request is sent to the PARSRV daemon running on the target node.

Calling **clio_create** to create a servant process does not do a TCP/IP **connect** to its master.

The servant can determine the master's port number and internet address by calling **clio_env("MstPort")** and **clio_env("MstHid")**, respectively.

Syntax

Language	Programming Syntax
C	**int clio_create (char** **node*, **char** **cmd*, **int** *sd* **)**

Where:

node Is a pointer to a symbolic name of a machine. See "clio_n2i—Locating a Machine by Symbolic Name" on page 181 for details on node values.

cmd Is a command string with which the process is invoked. This is the AIX command line or the name of the MVS JCL that starts the servant.

Note: The names FCFSERV, TAPESRV, CLPLINK, CLFTP, and C*A*N have special meaning to PARSRV. Do not give your servant program these names.

sd Is an input integer representing a socket descriptor. If *sd* is greater than zero, it is assumed to be a TCP/IP socket. Further, if it is not bound to a port, it is dynamically bound.

Return Values

Upon successful completion a zero or higher is returned. This value is a CLIO/S process ID that is used by other calls. A negative return code means failure; the process ID of the new process is not available until either **clio_accept** or **clio_connect** is called.

clio_end—Ending a CLIO/S Connection

Description

The **clio_end** function ends a CMI connection.

Syntax

Language	Programming Syntax
C	**void clio_end (void)**

Return Values

There is no return code or value for this call.

clio_env—Getting the Environment

Description

The **clio_env** function returns the value of environment variables. On AIX systems, this is identical to the **getenv()** call. On MVS, **clio_env** tries to obtain the value from the file represented by the DD card CLIOENV. If the DD card does not exist, it looks in the file *user_ID*.**CLIOENV**. Each record in this file has the format: *name=value*

clio_env can also be used to read variables added to the internal CLIO/S environment variable pool by **clio_exp**. “Setting CLIO/S Environment Variables” on page 36 discusses the environment variables used by CLIO/S. “clio_exp—Setting the Environment” on page 177 describes the CLIO/S subroutine you use to set these variables from within your CLIO/S program.

Syntax

Language	Programming Syntax
C	**char *clio_env (char *** *name* **)**

Where:

name Is a pointer to the name of an environment variable.

Return Values

This function returns a pointer to the string containing the value of the specified variable or null if the variable has no value.

clio_exp—Setting the Environment

Description

The **clio_exp** function can be used to add a value to a CLIO/S internal pool of environment variables. These internal environment variables can then be obtained by the **clio_env** function. "Setting CLIO/S Environment Variables" on page 36 discusses the environment variables used by CLIO/S.

"clio_env—Getting the Environment" on page 176 describes the routine you use to read CLIO/S environment variables. **clio_env** examines the pool of internal environment variables before examining the external environment variables. This allows **clio_exp** to not only add new environment variables, but also to override existing ones.

Variables created by **clio_exp** on MVS are passed to PARSRV when **clio_create** is called to start a process. On MVS, these environment variables are available within PARSRV's JCL generation (see Appendix D, "Controlling How CLIO/S Servers Are Started" on page 279 for more information). In effect, this allows the process calling **clio_create** to pass information to JCL generation.

When you use **clio_create** to start a process on MVS and the JCL for that program includes the .CLIOENV section, the variables defined by **clio_exp** are passed to the started process through the CLIOENV DD statement. In contrast, when you use **clio_create** to start processes on AIX, the internal variables created or changed by **clio_exp** are not available to these processes.

Syntax

Language	Programming Syntax
C	**int clio_exp (char ****key***, char ****value***)**

Where:

key Is the name of the internal environment variable.

value Is the character string value assigned to this internal environment variable.

Return Values

Upon successful completion a zero is returned. A value of –1 indicates that the internal pool of environment variables is full. A value of –2 indicates that a NULL key was specified.

clio_h2i—Locating Machine Information Using an Internet Address

Description
The **clio_h2i** function finds more information about a machine given the machine's internet address (*hid*). The PARCONF file is scanned to build a list of MINFO structures. If *hid* is not found in this list, the function checks TCP/IP to see if it is a valid internet address. If the address is valid, an MINFO structure is added to the list.

Syntax

Language	Programming Syntax
C	**MINFO *clio_h2i (Hid_t** *hid* **)**

Where:

hid Is the internet address.

Return Values
This function returns a pointer to MINFO or returns a null pointer if it was unsuccessful.

clio_init—Initializing CLIO/S

Description
The **clio_init** function initializes CLIO/S. This is an optional routine; if you don't invoke it, then CLIO/S will perform the equivalent function for you.

Syntax

Language	Programming Syntax
C	**int clio_init (void)**

Return Values
This function returns the number of valid entries in PARCONF.

clio_kill—Stopping a Process on a Node

Description
The **clio_kill** function kills processes started by **clio_create**.

Syntax

Language	Programming Syntax
C	**void clio_kill (int** *pid,* **Hid_t** *hid* **)**

Where:

pid Is an input integer representing a process ID returned by **clio_create**.

On UNIX systems, *pid* is the UNIX process ID. On MVS it is the JES job number. The new process may obtain its *pid* by calling **clio_pid**. It may also be obtained from the **CLIO_t** structure returned by **clio_accept** or **clio_connect** when the connection was established.

hid Is the internet address of the process.

Return Values
There is no return code or value.

clio_l2n or clio_n2l—Converting Data Representations

Description
When CLIO/S is running on two machines with different data representations, CLIO/S can perform the conversion from one data representation to another using **clio_n2l** or **clio_l2n**. **clio_n2l** converts data from the network representation to the local representation. **clio_l2n** converts data from the local representation to the network representation. Network format is the same data representation used by the RS/6000, for example IEEE, Big Endian, or ASCII. The data representation of the local machine is called “local representation.”

When a CLIO/S connection is established, the two processes exchange and compare predefined values of the various data types. The conversion routines use this to determine whether conversion is needed.

Syntax

Language	Programming Syntax
C	**int clio_n2l (CLIO_t** ******cp*, **int** *dt*, **void** ******src*, **void** ******dest*, **int** *nbytes* **)**
C	**int clio_l2n (CLIO_t** ******cp*, **int** *dt*, **void** ******src*, **void** ******dest*, **int** *nbytes* **)**

The following macros in **fcfcmi.h** assist in starting up **clio_n2l** and **clio_l2n**:

ntrans = LN_char_a (void ******src*, **void** ******dest*, **int** *nbytes* **)**
ntrans = NL_char_a (void ******src*, **void** ******dest*, **int** *nbytes* **)**
ntrans = LN_long_a (void ******src*, **void** ******dest*, **int** *nel* **)**
ntrans = NL_long_a (void ******src*, **void** ******dest*, **int** *nel* **)**
ntrans = LN_float_a (void ******src*, **void** ******dest*, **int** *nel* **)**
ntrans = NL_float_a (void ******src*, **void** ******dest*, **int** *nel* **)**
ntrans = LN_double_a (void ******src*, **void** ******dest*, **int** *nel* **)**
ntrans = NL_double_a (void ******src*, **void** ******dest*, **int** *nel* **)**

Where:

cp Is a pointer to the **CLIO_t** structure. If *cp* is NULL, the conversion is performed from ASCII, IEEE, or Big Endian.

dt Is an input integer representing a data type. See “About CLIO/S Data Conversions” on page 171 or the **fcfcmi.h** header file for acceptable values.

src Is a pointer to the source address of the data. The source and destination addresses may be the same.

dest Is a pointer to the destination address of the data. The source and destination addresses may be the same.

nbyte Is an input integer representing the number of bytes.

nel Is an input integer representing the number of elements being converted.

Return Values
These functions return the number of bytes converted.

clio_m2i—Locating Machine Information Using a Machine Number

Description
The **clio_m2i** function searches the chain of MINFO structures to find the structure whose machine number matches *machno*. The address of that MINFO structure is returned.

Syntax

Language	Programming Syntax
C	**MINFO *clio_m2i (int** *machno* **)**

Where:

machno Is an input integer representing a machine number.

Return Values
If successful, this function returns a pointer to the MINFO structure with the specified *machno*. If unsuccessful, it returns a pointer to null.

clio_map—Mapping from One Nickname Chain to Another

Description
The **clio_map** function takes as input a nickname chain and produces another nickname chain based on the mappings defined in the PARCONF file.

Syntax

Language	Programming Syntax
C	**int clio_map(char * input_chain, char * output_chain)**

Where:

input_chain Is the nickname chain being passed in.

output_chain Is the nickname chain produced when all of the nicknames in *input_chain* are mapped to new nickname chains. If no mappings are defined for the nicknames in *input_chain*, *output_chain* is a copy of *input_chain*

Return Values
A value of zero is returned upon successful completion. A return of –1 indicates that a mapping of the nicknames is defined recursively.

clio_n2h—Getting an Internet Address

Description

The **clio_n2h** function obtains the the internet address based on a name. The name can be a nickname defined in the PARCONF file, a nodename, or a dotted decimal address, for example 129.40.64.28.

Syntax

Language	Programming Syntax
C	**Hid_t clio_n2h (char *** *name* **)**

Where:

name Is a pointer to a symbolic name.

Return Values

The function returns the internet address, or it returns INADDR_NONE (–1) if it is unsuccessful.

clio_n2i—Locating a Machine by Symbolic Name

Description

The **clio_n2i** function finds the MINFO structure containing *name*. The name can be a nickname defined in the PARCONF file, a nodename or a dotted decimal address, for example 129.40.64.28.

The PARCONF file is used to build a list of MINFO structures. If the node is not found in this list, the function tries to obtain a valid internet address. If successful, the node is added to the list.

Syntax

Language	Programming Syntax
C	**MINFO *clio_n2i (char *** *name* **)**

Where:

name Is a pointer to a symbolic name.

Return Values

This function returns a pointer to the MINFO structure or a null pointer if it is unsuccessful.

clio_n2m—Getting the Machine Information ID

Description

The **clio_n2m** function obtains a machine number based on the symbolic name. The machine number is the entry number of *name* in PARCONF.

Note: This routine is primarily used by DPI applications that are also accessing CMI routines.

Syntax

Language	Programming Syntax
C	**int clio_n2m (char** ***name*)**

Where:

name is a pointer to a symbolic machine name.

Return Values

This function returns the MINFO ID (*machno*) field if it is successful, otherwise it returns a –1.

clio_own—Allowing a Process to Change Ownership of a Connection

Description

The **clio_own** function lets a process change ownership of a connection. Ownership is important when closing a connection, in particular for CTC protocols. When a child process (one that has been created by a **fork()** from a parent process) wants to close a connection that was opened by the parent process, it must call **clio_own** to gain ownership before closing the connection.

Syntax

Language	Programming Syntax
C	**int clio_own (CLIO_t** ***cp*, int** *ownflag* **)**

Where:

cp Is a pointer returned by **clio_connect** or **clio_accept** when the connection is made. *cp* can also be obtained by calling the DPI call, **cdpi_s2c**.

ownflag Is an input integer flag as follows:

0 Release ownership
1 Take ownership

Return Values

This function returns the process ID of the previous owner.

clio_pid—Returning a Process ID

Description

The **clio_pid** function returns a unique process ID. In UNIX the equivalent call is **getpid( )** . In MVS a value derived from the SSIBJBID field is returned. The value is five digits where the last four digits are the JES job ID and the first digit indicates the environment of the caller. The first digit can be one of the following: 2 - batch job, 3 - started task, and 4 - TSO user.

Syntax

Language	Programming Syntax
C	**int clio_pid (void)**

Return Values

This function returns the process ID of the caller.

clio_read—Reading Data over a CLIO/S Connection

Description

The **clio_read** function reads data over a CLIO/S connection. The number of bytes requested and datatype specification must be the same as in the corresponding **clio_write** request. No buffering is performed, but when data conversion is necessary, the request may be split into a number of smaller requests internally.

Specify the data type, *dt*, to ensure appropriate conversion of your data. See “About CLIO/S Data Conversions” on page 171 for possible values.

If a previous call to **clio_async** has set *asyncflag* to 1 (on), calls to **clio_read** or **clio_write** are asynchronous; the calls may return before the data has been completely transmitted.

Syntax

Language	Programming Syntax
C	**int clio_read (CLIO_t ****cp*, **int** *dt*, **void ****dest*, **int** *nbyte* **)**

Where:

cp Is a pointer to an identification structure.

dt Is an input integer specifying the type of data to be read. See “About CLIO/S Data Conversions” on page 171 for possible values.

dest Is a pointer to the destination of the data.

nbyte Is an input integer representing the number of bytes to be read.

Return Values

This function returns the actual number of bytes read or a –1 if unsuccessful.

clio_write—Writing Data over a CLIO/S Connection

Description

The **clio_write** function writes data over a CLIO/S connection. The number of bytes requested and datatype specification must be the same as in the corresponding **clio_read** request. No buffering is performed, but when data conversion is necessary, the request may be split into a number of smaller requests internally.

If a previous call to **clio_async** has set *asyncflag* to 1 (on), calls to **clio_read** or **clio_write** are asynchronous; the calls may return before the data has been completely transmitted.

Specify the data type, *dt*, to ensure appropriate conversion of your data. See "About CLIO/S Data Conversions" on page 171 for more information.

Syntax

Language	Programming Syntax
C	**int clio_write (CLIO_t ****cp***, int** *dt***, void ****src***, int** *nbyte* **)**

Where:

cp Is a pointer to an identification structure.

dt Is an input integer specifying the type of data to be written. See "About CLIO/S Data Conversions" on page 171 for possible values.

src Is a pointer to the data to be written.

nbyte Is an input integer representing the number of bytes to be written.

Return Values

This function returns the actual number of bytes written or a –1 if it is unsuccessful.

tcp_accept—Accepting a TCP/IP Connection

Description

The **tcp_accept** function executes the TCP/IP **accept** call. This call is the opposite side of **tcp_connect**. An application may bypass this call and accept the connection by itself, if desired. **tcp_accept** does not issue a TCP/IP **listen** call.

Syntax

Language	Programming Syntax
C	**int tcp_accept (int** *sd***, int** *tmo***, char ****text* **)**

Where:

sd Is an input integer representing the socket descriptor.

tmo Is an input integer representing the timeout value in seconds.

text Is a pointer to a character string which may accompany any messages produced.

Return Values

The new socket number is returned. The function returns –1 if no connection is established in *tmo* seconds.

tcp_connect—Establishing a TCP/IP Connection

Description

The **tcp_connect** function gets a TCP/IP socket and then connects it to the specified port. An application may bypass this call and get the socket and establish the connection by itself, if desired.

Syntax

Language	Programming Syntax
C	**int tcp_connect (int** *port*, **Hid_t** *hid*, **char ***text* **)**

Where:

port Is an input integer representing the TCP/IP port number.

hid Is a datatype representing the internet address of the host.

text Is a pointer to a character string which may accompany any messages produced.

Return Values

This function returns a socket descriptor or –1 if the connection is not established.

Chapter 11. Using the Distributed Processing Interface (DPI)

This chapter includes a description of the Distributed Processing Interface (DPI) and defines the calls and functions available in the DPI. The DPI provides calls and functions that allow you to start processes, transfer data, and perform data conversion. In addition, the DPI interface gives you more control over the distributed processes than is available with the CMI interface. You can also mix CMI subroutines (described in Chapter 10, "Using the Communication Manager Interface (CMI)" on page 171) and DPI subroutines in your program.

The DPI Header File fcfdpi.h

The DPI functions are defined in the **fcfdpi.h** header file. The DPI interface is available as both C and FORTRAN subroutines. The following lists where to find the header file on MVS and AIX:

MVS **SYS1.SFCFHDRS(FCFDPI)**

AIX **/usr/lpp/clnettape/inc**

The header also defines macros that assist you in calling the data conversion routines. These macros are described along with the associated data conversion routine.

The DPI interface uses the structures created by the CMI interface routines. **fcfdpi.h** includes the CMI header file, **fcfcmi.h**, which is described in "CMI Header File fcfcmi.h" on page 171.

About CLIO/S Data Conversions

When data is transferred between machines that have different data representations, CLIO/S automatically performs any necessary data conversions. The conversions are performed to and from a network format, which currently is IEEE, Big Endian, and ASCII. The conversions are always done on MVS. Between RISC systems, data conversion is not required.

When you call a DPI data transfer or conversion routine, you supply an argument (*dt*) defining the type of data you are transferring. This argument has one of the following values, as defined in **fcfcmi.h**, which is included by the **fcfdpi.h** header.

0	**OPAQ**	Opaque. No data conversions are performed
1	**CHAR**	Character data
2	**INTE**	Integer data (INTEGER*4)
3	**REAL**	Single-precision floating point
4	**DOUB**	Double-precision floating point
5	**LOGI**	Logical integer data (LOGICAL*4)

© Copyright IBM Corp. 1993, 1995, 1996

Overview of the DPI

The DPI allows applications to start and control processes on other systems, perform data conversions, and provides a way to control multiple processes. DPI processes use high-speed communication functions to exchange data. The communication functions are similar to the CMI, though the interface is different. The DPI has two types of functions: master and servant control functions and data transfer functions. DPI functions are available in two programming languages: C and FORTRAN. The functions available in the DPI are defined in the **fcfdpi.h** file.

The C language DPI subroutines whose names are composed of upper case letters (for example, ALERT) are C *macros*.

The calls and functions that provide control over master and servant processes are listed in Table 8. The calls and functions that perform data conversion and transfer are listed in Table 9 on page 189. These tables provide a brief description of the task performed by each subroutine.

Table 8 (Page 1 of 2). DPI Master and Servant Control Subroutines

Subroutine			
C	**FORTRAN**	**Description**	**Page**
ALERT	ALERT	Setting an Alert Group	194
ALERTGRP	ALRTGRP	Adding a Process ID to an Alert Group	195
cdpi_async	CDPIASY	Setting an Asynchronous Flag By Group	195
cdpi_check	CDPICHK	Checking Number of Bytes Transmitted by a Group	196
cdpi_s2c	N/A	Returning a CLIO/S Pointer	196
CLIOBLK	CLIOBLK	Setting the Buffer Bypass Limit	200
CLIOBUF	CLIOBUF	Setting the Size of the Internal Buffer	200
CLIODEB	CLIODEB	Masking Printed Status Messages	201
CLIOENV	CLIOENV	Getting Environment Information	201
CLIOMST	CLIOMST	Telling OPNSRV to Start Independent Masters	202
CLIOSOCK	N/A	Returning the Socket Descriptor	202
CLIOTBS	CLIOTBS	Changing Send and Receive Buffer Sizes	203
CLIOTMO	CLIOTMO	Setting the Time-out Period	203
CLSFAM	CLSFAM	Stopping All Processes in a Family	204
CLSPAR	CLSPAR	Closing the DPI Environment	204
CLSSRV	CLSSRV	Stopping the Servant Process	205
FNDMACH	FNDMACH	Locating a Machine by Symbolic Name	205
NMACHS	NMACHS	Counting Valid Entries in PARCONF	208
OPNFAM	OPNFAM	Opening a Family of Processes	208
OPNPAR	OPNPAR	Initializing the DPI Environment	209
OPNSRV	OPNSRV	Starting a Servant Process	209
QMACH	N/A	Getting Machine Information	210
QSLAVE	N/A	Getting Servant Machine Information	211
SCHEDUL	SCHEDUL	Scheduling a Task to Start	213

Table 8 (Page 2 of 2). DPI Master and Servant Control Subroutines

Subroutine			
C	**FORTRAN**	**Description**	**Page**
SETFAM	SETFAM	Setting the Family ID	214
SYNCHRO	SYNCHRO	Waiting Until All or Selected Tasks are Completed	218
TSKEXIT	N/A	Informing a Master the Servant Process is Finished	219
WAITFOR	WAITFOR	Waiting Until an Alert is Available	219
WAITREQ	WAITREQ	Indicating a Servant is Waiting for Work	220

Table 9. DPI Data Transfer and Conversion Subroutines

Subroutine			
C	**FORTRAN**	**Description**	**Page**
CLFLUSH	CLFLUSH	Clearing Buffers Written Using SPANY	197
clio_l2n	LN*xxxx*	Converting Data to Network Format	198
clio_n2l	NL*xxxx*	Converting Data to Local Format	199
MSANY	MS*xxxx*	Transferring Data from Master to Servant	206
MS_flush	MSFLSH	Confirming Data Was Sent from Master to Servant	207
RPANY	RP*xxxx*	Receiving Data from a Process	211
rw_dpi	N/A	Reading and Writing Data	212
SETSRV	SETSRV	Forcing Data Transfer to the Servant ID	215
SMANY	SM*xxxx*	Transferring Data from Servant to Master	215
SM_flush	SMFLSH	Confirming Data Was Sent from Servant to Master	216
SPANY	SP*xxxx*	Sending Data to a Process	217

The CLIO/S DPI Master/Servant Model

The CLIO/S DPI uses a master/servant model. The master starts a number of servants, which all wait for tasks to perform. The servants may be compute servants or data servants, or a combination. Compute servants and data servants are likely to have different use characteristics. Data servant requests are normally synchronous, while compute servant requests correspond to tasks in a parallel program; for example, asynchronous tasks. For asynchronous tasks, CLIO/S provides a synchronism mechanism. When an eligible servant becomes available, the task is dispatched to that servant. When the task is complete, the servant notifies the master, which retrieves the results. Pure compute servants can be pooled. This provides some dynamic load balancing.

A servant can start other servants, enabling linear chains to be built. An example of when this is helpful is in remote tape access, assuming fast communications between the RS/6000 workstations, when only some of the workstations have fast host communication.

The CLIO/S DPI has some similarities to Sun Remote Procedure Call (RPC**) but also some important differences. Like RPC, CLIO/S sends requests to a remote process and uses special routines to send arguments or retrieve results, with transparent data conversions. Unlike RPC, CLIO/S requests are asynchronous and

are not really remote procedure calls. They are tasks that are dispatched and waited for. Also, CLIO/S servants are started and stopped by the application.

How the CLIO/S DPI Operates

The DPI's method of operation is illustrated in Figure 26 on page 191. A master process initiates a servant process over a standard TCP/IP link. The TCP/IP link is necessary for initialization and some control functions, but does not have to be used for data transfer. Both the servant and the master can run on either RISC or MVS systems. If the link is RS/6000-to-RS/6000, the servant is started by calling **rexec**. If MVS is being used, a continuously running task (similar to a UNIX daemon), PARSRV, is needed to start the servant.

Note: The CLIO/S DPI requires a standard TCP/IP interface. However, data transfer rates are better when a high speed connection such as ESCON or BMCA/PCA is used.

Data can be transferred using:

TCP/IP	Transmission Control Protocol/Internet Protocol
BMCA	Channel-to-Channel between MVS and a workstation with a Block Multiplexer Channel Adapter (BMCA) (BMCA to RS/6000).
ESCON	Channel-to-Channel between MVS and a workstation with ESCON (ESCON to RS/6000).

Notes:

1. A master process can communicate with multiple servant processes on one or more systems.
2. Master and servant processes can run on the same system.
3. Multiple master processes can run on the same system.
4. A servant process can also act as a master process.

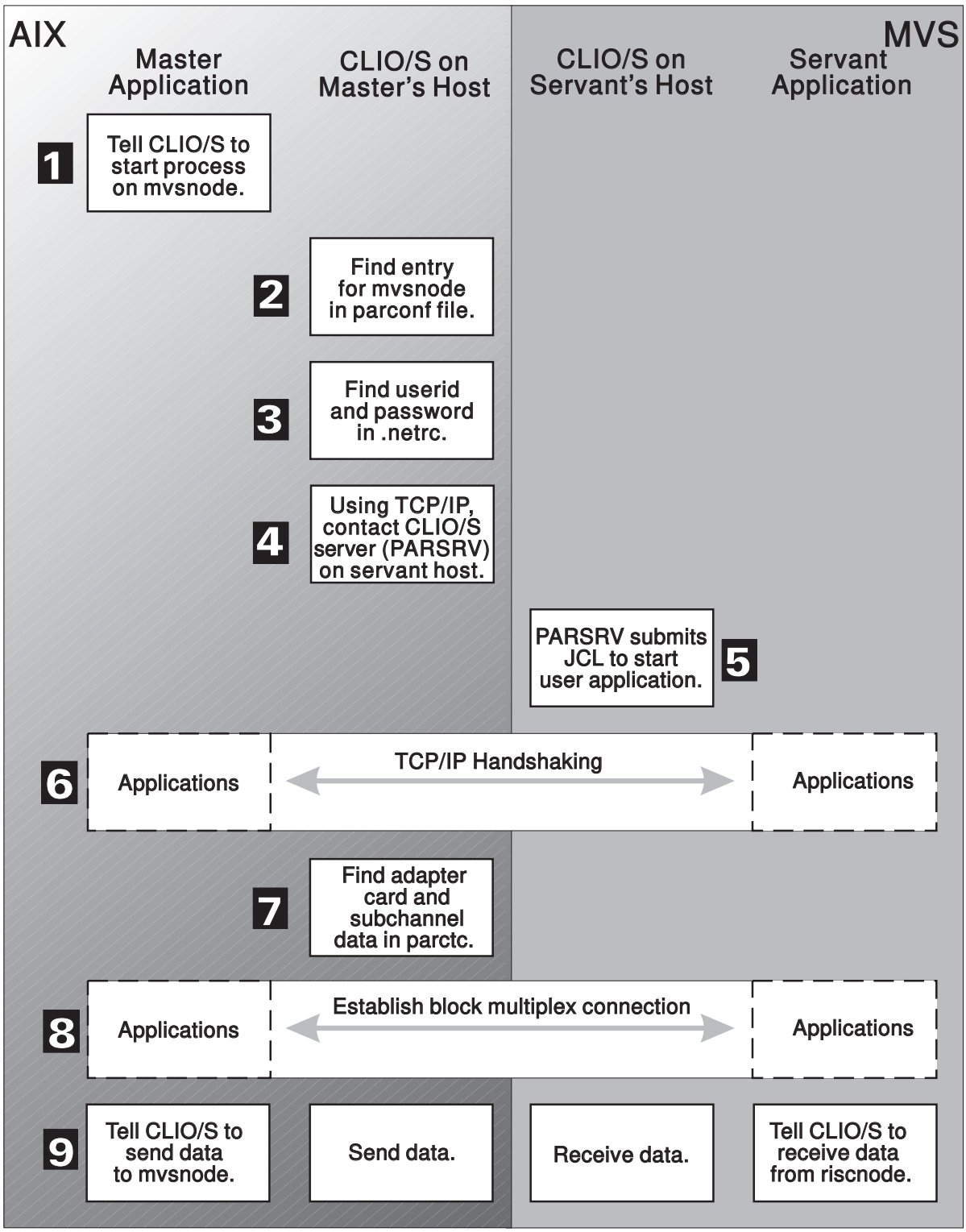


Figure 26. How the DPI Works

The CLIO/S DPI Sample Program

Many components interact when the CLIO/S DPI functions are used. The following sample programs, **fcfsampm** and **fcfsamps**, show how the CLIO/S DPI interface works.

Note: Line numbers have been added for reference in these examples.

```
 1 #include <fcfdpi.h>             /* CLIO/S declarations             */
 2
 3 main()
 4 {
 5   long myid, mach, isrv;
 6   int lmsg;
 7   char message[18] = "Master to servant";
 8
 9   myid = OPNPAR(0);              /* Initialize CLIO/S environment  */
10   mach = FNDMACH("mvsnode");     /* Locate node in parconf         */
11   isrv = OPNSRV(mach,1,"fcfsamps",0); /* Start servant             */
12   lmsg = strlen(message) + 1 ;
13   MS_long(&lmsg);                /* Master sends, servant receives */
14   MS_char_a(message,lmsg);       /* Master sends, servant receives */
15   MS_flush();                    /* Flush buffers                  */
16   CLSSRV(isrv);                  /* Close servant                  */
17   CLSPAR();                      /* Close CLIO/S environment       */
18   exit(0);
19 }
```

Figure 27. fcfsampm, a Sample CLIO/S Master Application.

```
 1 #include <fcfdpi.h>             /* CLIO/S declarations             */
 2 #include <stdio.h>
 3
 4 main()
 5 {
 6   int lmsg, myid ;
 7   char message[18];              /* Same size, both programs       */
 8
 9   myid = OPNPAR(0);              /* Initialize CLIO/S environment  */
10   lmsg = strlen(message) + 1 ;
11   MS_long(&lmsg);                /* Master sends, servant receives */
12   MS_char_a(message,lmsg);       /* Master sends, servant receives */
13   MS_flush();                    /* Flush buffers                  */
14   printf("Message: %s\n", message);
15   CLSPAR();                      /* Close CLIO/S environment       */
16   exit(0);
17 }
```

Figure 28. fcfsamps, a Sample CLIO/S Servant Application.

Starting the Master

When **fcfsampm** (the master) is run, it starts **fcfsamps** (the servant) on another machine called ***mvsnode***. The master sends a message to the servant, the servant receives the message, and both master and servant end. In this example, assume that **fcfsampm** is run on ***riscnode1***, an AIX node. Also assume that ***mvsnode***, where **fcfsamps** runs, is an MVS/ESA machine.

Let's look at **fcfsampm** in more detail:

Line 1, `#include <fcfdpi.h>` Includes the definitions supporting the calls to CLIO/S. Every application using the DPI must include this file.

Line 9, `myid = OPNPAR(0);` Initializes the CLIO/S environment for this program. The returned value, `myid`, equals 0 because the master program makes the call. The servant program **fcfsamps** makes a corresponding call in line 9 of that program, receiving the servant id.

Line 10, `mach = FNDMACH("mvsnode");` Uses the configuration files that were created as part of the installation process. `FNDMACH` searches the PARCONF file on ***riscnode1*** for an entry for ***mvsnode***. That entry looks something like this:

```
mvsnode  2 3 kn1mvst
```

The `2` in this entry tells CLIO/S that ***mvsnode*** is an MVS system. The `3` indicates the connection between the ***riscnode1*** and ***mvsnode*** is a parallel channel adapter. The internet address is required for CLIO/S to initialize communications between the nodes.

PARCONF is found in **$HOME/parconf** or **/etc/parconf** on AIX and in *user_ID*.**PARCONF** on MVS.

Starting the Servant

Line 11 in **fcfsampm** (Figure 27 on page 192),
`isrv = OPNSRV(mach,1,"fcfsamps",0);`, Starts the servant process on the remote node. This is a key operation for the DPI, and many steps are involved.

The PARCONF entry for ***mvsnode*** indicated that a parallel channel adapter is being used to communicate between ***riscnode1*** and ***mvsnode***. For details on this connection, CLIO/S looks in the PARCTC file, which is found in **$HOME/parctc** or in **/etc/parctc** on ***riscnode1***. The entry for ***mvsnode*** looks something like this:

```
kn1mvst  hpssl1  /dev/cat0  20   660-66F
```

/dev/cat0 is the adapter name for the parallel channel adapter on ***riscnode1***. 20 is the adapter ID, which must match the corresponding esoteric name in the MVS IOCP statement defining the parallel channel adapter as a CTC device. 660-66F indicates which MVS subchannels are to be used to communicate with this adapter card.

After reading PARCTC, **OPNSRV** has enough information to establish a channel-to-channel link between ***riscnode1*** and ***mvsnode***. However, **OPNSRV** still needs to know the user ID and password to use to start the new process. This information is in the NETRC file.

From the earlier call to **FNDMACH**, CLIO/S already knows that ***mvsnode*** has internet address ***kn1mvst.kgn.ibm.com***. In the NETRC entry for that address it finds something like this:

```
machine kn1mvst.kgn.ibm.com login mvsuser password dummy
```

fcfsamps is then started on mvsnode using user ID mvsuser.

The NETRC file is not CLIO/S-specific; it is also used by **rexec** and similar programs. NETRC is found in **$HOME/.netrc** on AIX and in *user_ID*.**NETRC** on MVS.

If ***mvsnode*** were an AIX system, CLIO/S would use **rexec** to start the process. Because ***mvsnode*** is running MVS, the PARSRV functions of CLIO/S on ***mvsnode*** dynamically generate Job Control Language (JCL) to start the program. See Appendix D, “Controlling How CLIO/S Servers Are Started” on page 279 for more information. JCL generation can be modified to suit the needs of your installation.

After the Servant is Started

The following part of the sample master program is run after the servant is started:

Line 14 in **fcfsampm** (Figure 27 on page 192), `MS_char_a(message,lmsg);` Sends the message from the master to the servant. Line 12 in **fcfsamps** is the corresponding instruction. Note that **fcfsampm** and **fcfsamps** make exactly the same call. When the master runs this command, it is a send; when the servant runs it, it is a receive. `MS_flush();` flushes the buffers.

Line 16, `CLSSRV(isrv);` Causes CLIO/S to close the servant on ***mvsnode***.

Line 17, `CLSPAR();` Closes the parallel environment.

Using the DPI Functions

ALERT—Setting an Alert Group

Description

ALERT lets you set an alert group by sending an application-defined code to another process. You can use alerts to provide synchronization between processes, or to provide a non-blocking query function. This call blocks until the specified process has selected the **ALERT** in a **WAITFOR**.

The master/servant concept is not relevant to **ALERT**. The master or any servant may use this call to synchronize with any other process.

Syntax

Language	Programming Syntax
C	**int ALERT (int** *id*, **int** *code* **)**
FORTRAN	**CALL ALERT (** *id*, *code*, *ret* **)**

Where:

id Is an input integer (INTEGER*4) representing the servant identifier. An *id* of zero indicates the master.

code Is an input integer (INTEGER*4) representing an application-defined code sent to the process.

ret Is an output integer (INTEGER*4) representing the return code.

Return Values

Upon successful completion, a 0 is returned. If the function completes unsuccessfully, –1 is returned.

ALERTGRP or ALRTGRP—Adding a Process ID to an Alert Group

Description

ALERTGRP or **ALRTGRP** lets you add a process to an alert group. The alert groups enable **WAITFOR** to wait for more than one process.

Syntax

Language	Programming Syntax
C	**int ALERTGRP (int** *group*, **int** *id* **)**
FORTRAN	**CALL ALRTGRP (** *group*, *id*, *ret* **)**

Where:

group Is an input integer (INTEGER*4) representing the group to which the process is added. The group number must be between 1 and 31, inclusive.

id Is an input integer (INTEGER*4) representing the process identifier.

ret Is an output integer (INTEGER*4) representing the return code.

Return Values

Upon successful completion, a 0 is returned. If the function completes unsuccessfully, –1 is returned.

cdpi_async or CDPIASY—Setting an Asynchronous Flag By Group

Description

cdpi_async or **CDPIASY** lets you set or reset the asynchronous flag for a process group. This function calls **clio_async**. See “clio_async—Setting the Asynchronous Data Flag” on page 173 for more information.

Syntax

Language	Programming Syntax
C	**int cdpi_async (int** *group*, **int** *asyncflag* **)**
FORTRAN	**CALL CDPIASY (** *group*, *asyncflag*, *ret* **)**

Where:

group Is an input integer (INTEGER*4) representing a group of processes as follows:

≤ 0 represents a particular process identifier.
= 1 represents all processes.
> 1 represents all processes belonging to group *group*.

asyncflag Is a logical input integer (LOGICAL*4) representing on (1) or off (0) (true or false).

ret Is an output integer (INTEGER*4) representing the return code.

Return Values

Upon successful completion, a 0 is returned. If the function completes unsuccessfully, –1 is returned.

cdpi_check or CDPICHK—Checking Number of Bytes Transmitted by a Group

Description

cdpi_check or **CDPICHK** lets you check the number of bytes transmitted by a group. This function waits for an asynchronous I/O operation to and from the process, indicated by *group*, to complete. **cdpi_check** calls **clio_check** to find all the indicated processes and returns the total number of bytes transmitted for each process in *group*.

Syntax

Language	Programming Syntax
C	**int cdpi_check (int** *group* **)**
FORTRAN	**CALL CDPICHK (** *group*, *ret* **)**

Where:

group Is an input integer (INTEGER*4) representing a group of processes.

ret Is an output integer (INTEGER*4) representing the total number of bytes transmitted.

Return Values

Upon successful completion, the number of bytes transferred is returned. If the function completes unsuccessfully, –1 is returned.

cdpi_s2c—Returning a CLIO/S Pointer

Description

cdpi_s2c returns a pointer to the CLIO_t structure that is used by many CMI routines when referring to the servant process. A DPI application written in C needs this structure to call CMI routines.

Syntax

Language	Programming Syntax
C	**CLIO_t *cdpi_s2c (int** *servantid* **)**
FORTRAN	There is no FORTRAN equivalent function.

Where:

servantid Is an input integer representing the servant id.

You can use **cdpi_s2c** to obtain the actual protocol used by CLIO/S, which may be different from what was requested. The following code segment illustrates how to do this:

```
isrv = OPNSRV( mach, 1, servername, 0 )
infoptr = cdpi_s2c(isrv);
if (infoptr -> proto == P_ESCON) printf("Protocol is ESCON.");
if (infoptr -> proto == P_PCA  ) printf("Protocol is PCA.  ");
if (infoptr -> proto == P_TCPIP) printf("Protocol is TCPIP.");
```

proto is part of the CLIO_t structure found in **fcfcmi.h**. P_ESCON, P_PCA, and P_TCPIP are also defined in **fcfcmi.h**.

Return Values

Upon successful completion, a pointer to the CLIO_t structure is returned. If the function completes unsuccessfully, a null pointer is returned.

CLFLUSH—Clearing Buffers Written Using SPANY

Description

CLFLUSH clears buffers written using **SPANY** or **SP*xxxx***.

Syntax

Language	Programming Syntax
C	**int CLFLUSH (int** *prc* **)**
FORTRAN	**CALL CLFLUSH (** *prc*, *ret* **)**

Where:

prc Is an input integer (INTEGER*4) representing the process ID used by **SPANY** or **SP*xxxx***.

ret Is an output integer (INTEGER*4) representing the return code.

Return Values

Upon successful completion, a 0 is returned. If the function completes unsuccessfully, a –1 is returned.

clio_l2n or LNxxxx—Converting Data to Network Format

Description

LN*xxxx* or **clio_l2n** converts data from local machine-dependent format to network format, IEEE Big Endian, ASCII. This routine is the same as the CMI routine **clio_l2n**. For more information about converting data, see “clio_l2n or clio_n2l—Converting Data Representations” on page 179.

Syntax

Language	Programming Syntax
C	**int clio_l2n (CLIO_t** ***cp*****, int** *dt*, **void** ***src*****, void** ***dest*****, int** *nel* **)**
FORTRAN	**CALL LN*xxxx* (** *src*, *dest*, *nel* **)**

Where:

cp Is a pointer to the **CLIO_t** structure. Use *cp* to indicate that the conversion should be performed only if the data representation on the node pointed to by *cp* is different from the local node. If *cp* is NULL the conversion is always done.

dt Is an integer (INTEGER*4) representing the data type. The following values are accepted:

0 OPAQ
1 CHAR
2 INTE
3 REAL
4 DOUB
5 LOGI

src Is a pointer to the source array being converted.

dest Is a pointer to the destination array for the converted data.

nel Is an input integer (INTEGER*4) representing the number of bytes being converted.

xxxx can be one of the following:

CHAR represents character data.
INTE represents INTEGER*4.
LOGI represents LOGICAL*4.
REAL represents single-precision floating point.
DOUB represents double-precision floating point.

Return Values

This function returns the number of bytes converted. The array pointed to by *dest* contains the converted data.

clio_n2l or NLxxxx—Converting Data to Local Format

Description

clio_n2l or **NL*xxxx*** converts data from network format, IEEE Big Endian, ASCII to local machine-dependent format. The parameters are the same as for **LN*xxxx***. This routine is the same as the CMI routine **clio_n2l**. For more information about converting data, see “clio_l2n or clio_n2l—Converting Data Representations” on page 179.

Syntax

Language	Programming Syntax
C	**int clio_n2l (CLIO_t ****cp***, int** *dt***, void ****src***, void ****dest***, int** *nel* **)**
FORTRAN	**CALL NL*xxxx* (** *src*, *dest*, *nel* **)**

Where:

cp Is a pointer to the **CLIO_t** structure. Use *cp* to indicate that the conversion should be performed only if the data representation on the node pointed to by *cp* is different from the local node. If *cp* is NULL the conversion is always done.

dt Is an integer (INTEGER*4) representing the data type. The integer representing the data type is one of the following:

0 OPAQ
1 CHAR
2 INTE
3 REAL
4 DOUB
5 LOGI

src Is a pointer to the source array being converted.

dest Is a pointer to the destination array for the converted data.

nel Is an input integer (INTEGER*4) representing the number of bytes being converted.

xxxx can be one of the following:

CHAR represents character data.
INTE represents INTEGER*4.
LOGI represents LOGICAL*4.
REAL represents single-precision floating point.
DOUB represents double-precision floating point.

Return Values

This function returns the number of bytes converted. The array pointed to by *dest* contains the converted data.

CLIOBLK—Setting the Buffer Bypass Limit

Description

CLIOBLK enables the application to bypass buffering for data transfers larger than *newsize* bytes. It has no effect after **OPNPAR** has been called, directly or indirectly, or if the calling program is a servant. Servants inherit this value from their master. The default value is the value returned by **CLIOBUF**.

"Changing the Buffering Mechanism" on page 220 explains how to use the **CLIOBLK** and **CLIOBUF** functions and the CLIOBLK and CLIOBUF environment variables to control buffering.

Syntax

Language	Programming Syntax
C	**int CLIOBLK (int** *newsize* **)**
FORTRAN	**CALL CLIOBLK (** *newsize*, *oldsize* **)**

Where:

newsize Is an input integer (INTEGER*4) representing the new size of the buffer limit, in bytes, where buffering is bypassed.

If *newsize* is –1, **CLIOBLK** returns the current limit size without changing it.

oldsize Is an output integer (INTEGER*4) representing the previous buffer limit.

Return Values

This routine returns the previous value of the buffer limit.

CLIOBUF—Setting the Size of the Internal Buffer

Description

CLIOBUF lets you set the size of the internal buffer to *newsize* bytes. This function returns the previous setting. It has no effect after **OPNPAR** has been called, directly or indirectly, or if the calling program is a servant. Servants inherit this value from their master.

"Changing the Buffering Mechanism" on page 220 explains how to use the **CLIOBLK** and **CLIOBUF** functions and the CLIOBLK and CLIOBUF environment variables to control buffering.

Syntax

Language	Programming Syntax
C	**int CLIOBUF (int** *newsize* **)**
FORTRAN	**CALL CLIOBUF (** *newsize*, *oldsize* **)**

Where:

newsize Is an input integer (INTEGER*4) representing the new size of the internal buffer, in bytes. The minimum size of the buffer is 1024 bytes.

oldsize Is an output integer (INTEGER*4) representing the previous buffer value.

Return Values
The routine returns the previous size of the internal buffer.

CLIODEB—Masking Printed Status Messages

Description
CLIODEB masks the printing of status messages. Servants inherit this value from their master. “Setting CLIO/S Environment Variables” on page 36 lists possible values for this flag.

Note: You can't query the value of the mask without changing it. In the unlikely event that you need to know the value, call CLIODEB with any argument. The return value is the value you want to know. Then call CLIODEB again, passing the previous return value, to restore the original setting.

You can't use **clio_env** to query it. It will return the mask setting in effect when the servant was started, but does not reflect any subsequent changes.

Syntax

Language	Programming Syntax
C	**int CLIODEB (int** *newflag* **)**
FORTRAN	**CALL CLIODEB (** *newflag*, *oldflag* **)**

Where:

newflag Is an input integer (INTEGER*4) representing the new debug flag value.

oldflag Is an output integer (INTEGER*4) representing the previous debug flag value.

Return Values
The previous debug mask is returned.

CLIOENV—Getting Environment Information

Description
CLIOENV provides information about the environment file, similar to the **getenv()** call on UNIX systems. **CLIOENV** is the same as the CMI **clio_env** call. On MVS, it reads the CLIOENV DD card, if it exists, or *user_ID*.**CLIOENV** to find the value. See “clio_exp—Setting the Environment” on page 177 for information about setting the CLIO/S environment.

Syntax

Language	Programming Syntax
C	**char *CLIOENV (char *_name_)**
FORTRAN	**CALL CLIOENV (** _name_**,** _val_ **)**

Where:

name Is (CHARACTER*(*)) representing the name of the environment variable.

val Is (CHARACTER*(*)) representing the value of the environment variable.

Return Values

Upon successful completion, a pointer to the value for the environment variable is returned. If the environment is not found, a null pointer is returned.

CLIOMST—Telling OPNSRV to Start Independent Masters

Description

CLIOMST sets a flag that enables **OPNSRV** to start independent masters that do not know whether they were started by another program. This lets you start a job on MVS from the workstation. By default, **OPNSRV** starts a servant process.

The setting of **CLIOMST** persists until the next call to **CLIOMST**.

Syntax

Language	Programming Syntax
C	**int CLIOMST (int** _newflag_ **)**
FORTRAN	**CALL CLIOMST (** _newflag_**,** _oldflag_ **)**

Where:

newflag Is an input integer (INTEGER*4) specifying how new processes are to be started. Possible values are:

1 True - Start as independent master. Do not connect to child process.

0 False - Start as servant process.

oldflag Is an output integer (INTEGER*4) representing the previous setting.

Return Values

The previous setting of **CLIOMST** (TRUE or FALSE) is returned.

CLIOSOCK—Returning the Socket Descriptor

Description

CLIOSOCK returns the socket descriptor for the control communication link with process *servantid*.

Syntax

Language	Programming Syntax
C	**int CLIOSOCK (int** *servantid* **)**
FORTRAN	There is no FORTRAN equivalent.

Where:

servantid Is an input integer (INTEGER*4) representing the servant identifier.

Return Values

Upon successful completion, the socket descriptor number is returned. If the function completes unsuccessfully, –1 is returned.

CLIOTBS—Changing Send and Receive Buffer Sizes

Description

CLIOTBS changes the SEND and RECEIVE buffer sizes for TCP/IP. It has an effect only if the protocol chosen is TCP/IP. Servants inherit this value from their master.

Syntax

Language	Programming Syntax
C	**int CLIOTBS (int** *newsize* **)**
FORTRAN	**CALL CLIOTBS (** *newsize*, *oldsize* **)**

Where:

newsize Is an input integer (INTEGER*4) representing the new TCP/IP buffer size.

oldsize Is an output integer (INTEGER*4) representing the old TCP/IP buffer size.

Return Values

The routine returns the previous buffer size.

CLIOTMO—Setting the Time-out Period

Description

CLIOTMO sets the length of the time period the master waits for the servant. When a servant process is started, the master waits up to the length of the time-out period you set for a response from the servant. If no response is received, it is assumed that **OPNSRV** failed.

Syntax

Language	Programming Syntax
C	**int CLIOTMO (int** *newtmo* **)**
FORTRAN	**CALL CLIOTMO (** *newtmo*, *oldtmo* **)**

Where:

newtmo Is an input integer (INTEGER*4) representing the number of seconds a master waits for a response from the servant.

oldtmo Is an output integer (INTEGER*4) representing the previous time-out value, in seconds.

Return Values

The old time-out value is returned.

CLSFAM—Stopping All Processes in a Family

Description

CLSFAM stops all processes in a family.

Syntax

Language	Programming Syntax
C	**int CLSFAM (int** *family* **)**
FORTRAN	**CALL CLSFAM (** *family*, *ret* **)**

Where:

family Is an input integer (INTEGER*4) representing the family identifier.

ret Is an output integer (INTEGER*4) representing the return code.

Return Values

Upon successful completion, a 0 is returned. If the function completes unsuccessfully, –1 is returned.

CLSPAR—Closing the DPI Environment

Description

CLSPAR closes the DPI environment and stops all servants.

Syntax

Language	Programming Syntax
C	**void CLSPAR (void)**
FORTRAN	**CALL CLSPAR (** *ret* **)**

Where:

ret Is an output integer (INTEGER*4) representing the return code.

Return Values

Upon successful completion, a 0 is returned. If the function completes unsuccessfully, 0 is returned.

CLSSRV—Stopping the Servant Process

Description

CLSSRV stops a servant process.

Syntax

Language	Programming Syntax
C	**void CLSSRV (int** *srv* **)**
FORTRAN	**CALL CLSSRV (** *srv*, *ret* **)**

Where:

srv Is an input integer (INTEGER*4) representing the identifier of the servant being stopped.

ret Is an output integer (INTEGER*4) representing the return code.

Return Values

Upon successful completion, a 0 is returned. If the function completes unsuccessfully, –1 is returned.

FNDMACH—Locating a Machine by Symbolic Name

Description

FNDMACH lets you locate a machine based on symbolic name. The function returns the entry number of *name* in PARCONF. The output from **FNDMACH** can be used directly as input to **OPNSRV**. This function is the same as the CMI function **clio_n2m**.

Syntax

Language	Programming Syntax
C	**int FNDMACH (char ***_name_ **)**
FORTRAN	**CALL FNDMACH (** *name*, *mach* **)**

Where:

name Is a pointer (CHARACTER*(*)) to the node name. The name can be the nodename or a nickname defined in the PARCONF file.

mach Is an output integer (INTEGER*4) representing an entry number in the PARCONF file.

Return Values

Upon successful completion, the machine number is returned. The machine number is an entry in the PARCONF file. If the function completes unsuccessfully, –1 is returned.

MSANY or MSxxxx—Transferring Data from Master to Servant

Description

MSANY or **MS*xxxx*** transfers *nitem* chunks of data from a master to a servant process. The size of each item is *itemsize* bytes. They are fetched from or stored to *buf*. The distance between the items is *stride* bytes. Each item consists of elements of type *dt*. The datatype can be one of the supported data types: OPAQ, CHAR, INTE, REAL, DOUB, or LOGI. The total number of bytes actually transferred is returned.

If your master process has multiple servants or your program uses the CLIO/S tape interface subroutine **ctpalloc**, specify the target servant by calling the SETSRV subroutine before calling **MSANY** or **MSxxxx**. See "SETSRV—Forcing Data Transfer to the Servant ID" on page 215 for more information.

Data conversions are performed if the sender and receiver reside on different machine types. The sender converts to IEEE Big Endian, and the receiver converts back to the local data representations. For more information about data types see "About CLIO/S Data Conversions" on page 187.

This subroutine is capable of sending disjoint parts of a matrix.

Syntax

Language	Programming Syntax
C	**int MSANY (int** *dt*, **void ****buf*, **int** *nitem*, **int** *itemsize*, **int** *stride* **)**
FORTRAN	**CALL MS*xxxx* (** *buf*, *nitem*, *itemsize*, *stride*, *ret* **)**

Where:

dt Is an input integer (INTEGER*4) representing the datatype being transferred. The integer representing the datatype is one of the following:

0 OPAQ
1 CHAR
2 INTE
3 REAL
4 DOUB
5 LOGI

buf Is a pointer to the array being transferred.

nitem Is an input integer (INTEGER*4) representing the number of items being transferred.

itemsize Is an input integer (INTEGER*4) representing the size of each item being transferred.

stride Is an input integer (INTEGER*4) representing the distance between items.

ret Is an output integer (INTEGER*4) representing the return code.

xxxx can be one of the following:

OPAQ Represents no data conversions done.
CHAR Represents character data.
INTE Represents INTEGER*4.
LOGI Represents LOGICAL*4.
REAL Represents single-precision floating point.
DOUB Represents double-precision floating point.

The following macros, defined in **fcfdpi.h**, simplify starting **MSANY**:

ntrans = MS_char (*buf* **)**
ntrans = MS_long (*buf* **)**
ntrans = MS_logical (*buf* **)**
ntrans = MS_float (*buf* **)**
ntrans = MS_double (*buf* **)**
ntrans = MS_opaque (*buf*, *itemsize* **)**
ntrans = MS_char_a (*buf*, *nitem* **)**
ntrans = MS_long_a (*buf*, *nitem* **)**
ntrans = MS_logical_a (*buf*, *nitem* **)**
ntrans = MS_float_a (*buf*, *nitem* **)**
ntrans = MS_double_a (*buf*, *nitem* **)**

Return Values

Upon successful completion, the number of bytes transferred is returned. If the function completes unsuccessfully, a −1 is returned. If CLIO/S couldn't process all the requested bytes, the function returns −205.

MS_flush or MSFLSH—Confirming Data Was Sent from Master to Servant

Description

MS_flush or **MSFLSH** lets you ensure data is actually sent from the master to a servant process.

Syntax

Language	Programming Syntax
C	**int MS_flush (void)**
FORTRAN	**CALL MSFLSH**

Return Values

Upon successful completion, a 0 is returned. If the function completes unsuccessfully, a −1 is returned.

NMACHS—Counting Valid Entries in PARCONF

Description

NMACHS returns the number of valid entries in PARCONF. The count of valid entries is valid only if this call is done before any other CLIO/S call. This function is the same as the CMI function **clio_init**.

Syntax

Language	Programming Syntax
C	**int NMACHS (void)**
FORTRAN	**CALL NMACHS (** *mach* **)**

Where:

mach Is an output integer (INTEGER*4) containing the number of entries in the PARCONF file.

Return Values

The number of valid entries in PARCONF is returned.

OPNFAM—Opening a Family of Processes

Description

OPNFAM lets you open a family of processes. A CLIO/S process belongs to one or more families. The family concept isolates processes in one family from processes in other families. The **OPNFAM** call is typically used, but not required, when a servant wants to start its own servants.

These restrictions apply to a family of processes:

- A maximum of 31 servants can be active within a family.
- You may have more than 31 servants in a family if some of the servants are stopped. However, servants started after the thirty-first servant may have trouble communicating with other servants; calls that communicate between servants, such as ALERT, SPANY, and RPANY, may return unsuccessfully when sent from or directed to a servant started after the thirty-first servant. Communication with the master is unaffected.

Syntax

Language	Programming Syntax
C	**int OPNFAM (void)**
FORTRAN	**CALL OPNFAM (** *family* **)**

Where:

family Is an output integer (INTEGER*4) representing the new family identifier.

Return Values

Upon successful completion, the family identifier is returned. If the function completes unsuccessfully, a null pointer is returned.

OPNPAR—Initializing the DPI Environment

Description

OPNPAR initializes the DPI environment. If the function is issued by a servant, a connection is made to the master, and *id* has the same value that was returned by the **OPNSRV** call. If the program is running as a master, the return value or *id* is zero.

Syntax

Language	Programming Syntax
C	**int OPNPAR (VoidFunc** *clsexit* **)**
FORTRAN	**CALL OPNPAR (** *id*, *ret* **)**

Where:

id Is an output integer (INTEGER*4) that returns the process ID.

clsexit Is the address of a subroutine of type VoidFunc. **CLSPAR** invokes the *clsexit* subroutine; this lets the program clean up in case of an interrupt. The FORTRAN interface does not support this argument.

ret Is an output integer (INTEGER*4) representing the return code.

Return Values

Upon successful completion of the routine, the CLIO/S process ID is returned (*id* in FORTRAN):

0 The master process
>0 The new servant
−1 The function returned unsuccessfully

In addition to returning *id*, the FORTRAN routine returns *ret*:

0 Successful completion
−1 Failure

OPNSRV—Starting a Servant Process

Description

OPNSRV starts a servant on machine *mach*. The servant accepts functions belonging to group *group*. When the servant is stopped, invoke the routine *clsexit*. **OPNSRV** blocks until the servant has called **OPNPAR** to establish communication. The servants are assigned identifiers sequentially starting with 1; for example, 1, 2, 3, and so on.

These restrictions apply to a family of processes:

- A maximum of 31 servants can be active within a family.

- You may have more than 31 servants in a family if some of the servants are stopped. However, servants started after the thirty-first servant may have trouble communicating with other servants; calls that communicate between servants, such as ALERT, SPANY, and RPANY, may return unsuccessfully when sent from or directed to a servant started after the thirty-first servant. Communication with the master is unaffected.

Note: To start a new process as an independent master, call **CLIOMST(TRUE)** before calling **OPNSRV**. This causes PARSRV to start the new process as an independent master. No communication path is established, and the process is not stopped by **CLSSRV**, **CLSFAM**, or **CLSPAR**.

Syntax

Language	Programming Syntax
C	**int OPNSRV (int** *mach*, **int** *group*, **char ***cmd*, **VoidFunc** *clsexit* **)**
FORTRAN	**CALL OPNSRV (** *cmd*, *mach*, *group*, *srv*, *ret* **)**

Where:

mach Is an input integer (INTEGER*4) representing the machine number.

group Is an input integer (INTEGER*4) representing the group. The value for *group* should be greater than 1.

cmd Is a pointer to a character, (CHARACTER*(*)) containing the command. This is the AIX command line or the name of the MVS JCL that starts the servant.

Note: The names FCFSERV, TAPESRV, CLPLINK, CLFTP, and C*A*N have special meaning to PARSRV. Do not give your servant program these names.

srv Is an output integer representing the servant number.

ret Is an output integer (INTEGER*4) representing the return code.

Return Values

Upon successful completion, the servant identifier (or a 0 if you are running a stand-alone program) is returned. If the function completes unsuccessfully, a –1 is returned.

QMACH—Getting Machine Information

Description

QMACH provides information about a particular machine. The information is read from a configuration file and includes communication protocol to be used and machine type. The function returns a pointer to a structure defined in **fcfcmi.h** or NULL if no entry is available. This function is the same as the CMI function **clio_m2i**.

Note: The machine type information is used only by CLIO/S to determine whether **rexec** can be used to start processes.

Syntax

Language	Programming Syntax
C	**MINFO *QMACH (int** *mach* **)**
FORTRAN	There is no FORTRAN equivalent.

Where:

mach Is an input integer representing the machine number.

Return Values

Upon successful completion, a pointer to the MINFO structure is returned. If the function completes unsuccessfully, a null pointer is returned.

QSLAVE—Getting Servant Machine Information

Description

QSLAVE provides information about the node where *servantid* is running.

Syntax

Language	Programming Syntax
C	**MINFO *QSLAVE (int** *servantid* **)**
FORTRAN	There is no FORTRAN equivalent function.

Where:

servantid Is an input integer representing the servant identifier.

Return Values

Upon successful completion, a pointer to the MINFO structure is returned. If the function completes unsuccessfully, a null pointer is returned.

RPANY or RPxxxx—Receiving Data from a Process

Description

RPANY or **RP*xxxx*** receives *nitem* chunks of data from a process *prc* in the current family. If the connection is not already established, CLIO/S attempts to connect to the process. received from (0 is master, 1 is servant1, and so on.) The parameters are the same as for **MSANY** or **MS*xxxx***.

Syntax

Language	Programming Syntax
C	**int RPANY (int** *prc*, **int** *dt*, **void *** *buf*, **int** *nitem*, **int** *itemsize*, **int** *stride* **)**
FORTRAN	**CALL RP*xxxx* (** *prc*, *buf*, *nitem*, *itemsize*, *stride*, *ret* **)** **CALL RPANY(** *prc*, *dt*, *buf*, *nitem*, *itemsize*, *stride*, *ret* **)**

Where:

prc Is an input integer (INTEGER*4) representing the process identifier.

dt Is an input integer (INTEGER*4) representing the datatype. The following values are accepted:

0 OPAQ
1 CHAR
2 INTE
3 REAL
4 DOUB
5 LOGI

buf Is a pointer to the buffer

nitem Is an input integer (INTEGER*4) representing the number of items being transferred.

itemsize Is an input integer (INTEGER*4) representing bytes per item.

stride Is an input integer (INTEGER*4) representing the distance between items being transferred.

ret Is an output integer (INTEGER*4) representing the return code.

xxxx can be one of the following:

CHAR Represents character data.
INTE Represents INTEGER*4.
LOGI Represents LOGICAL*4.
REAL Represents single-precision floating point.
DOUB Represents double-precision floating point.

The following macros, defined in **fcfdpi.h**, simplify starting **RPANY**:

ntrans = RP_char (*buf* **)**
ntrans = RP_long (*buf* **)**
ntrans = RP_logical (*buf* **)**
ntrans = RP_float (*buf* **)**
ntrans = RP_double (*buf* **)**
ntrans = RP_opaque (*buf*, *itemsize* **)**
ntrans = RP_char_a (*buf*, *nitem* **)**
ntrans = RP_long_a (*buf*, *nitem* **)**
ntrans = RP_logical_a (*buf*, *nitem* **)**
ntrans = RP_float_a (*buf*, *nitem* **)**
ntrans = RP_double_a (*buf*, *nitem* **)**

Return Values

Upon successful completion, the number of bytes received is returned. If the function completes unsuccessfully, a –1 is returned. If CLIO/S couldn't process all the requested bytes, the function returns –205.

rw_dpi—Reading and Writing Data

Description

rw_dpi sends multiple read and write requests to the DPI. It is similar to the UNIX **readv** or **writev** routines but allows you to mix read and write operations. When using one of the CTC protocols (BMCA/PCA or ESCON), this call can improve performance by chaining together read and write operations. For non-CTC protocols, this call generates a sequence of normal read and write calls and does not provide a performance benefit. This is the only CLIO/S call that lets you mix read and write operations.

"Chained Channel Programs for BMCA and ESCON" on page 222 contains a sample program and more information about using **rw_dpi** calls.

Syntax

Language	Programming Syntax
C	**int rw_dpi (int** *procid*, **RW_DPI** **rw_spec*, **int** *numspecs* **)**
FORTRAN	There is no FORTRAN equivalent function.

Where:

procid Is an input integer representing the process identifier. *procid* is either zero (master) or a servant ID returned by **OPNSRV**.

rw_spec Is a pointer to an array of RW_DPI structures. These are defined in **fcfcmi.h**, the CLIO/S CMI header file.

numspecs Is an input integer representing the number of elements in the array.

Return Values

Upon successful completion, a 0 is returned. If the function completes unsuccessfully, a −1 is returned.

SCHEDUL—Scheduling a Task to Start

Description

SCHEDUL allows a servant belonging to group *group* to start a task. You specify two subroutines. One is called when the task is physically dispatched; the other, when the results are to be retrieved. Both subroutines receive the same list of arguments, typically consisting of a sequence of data transfer calls. **SCHEDUL** is intended for asynchronous requests.

Note: The servant must call **WAITREQ**, perform the desired task, and then call **WAITREQ** again to inform the master the task is complete. To do this, the servant must be in an infinite **WAITREQ** loop. The recommended method of allowing the servant to exit the loop is to send a special request code, 0.

Syntax

Language	Programming Syntax
C	**int SCHEDUL (int** *group*, **int** *req*, **VoidFunc** *xarg*, **VoidFunc** *xres*, **int** *narg*, **void** ***args* **)**
FORTRAN	**CALL SCHEDUL (** *group*, *req*, *xarg*, *xres*, *task*, *narg*, *a1*, *a2*, ... **)**

Where:

group Is an input integer (INTEGER*4) representing the group. If *group* is negative, its absolute value is used as a servant identifier.

req Is an input integer (INTEGER*4) representing the request code being sent to the servant. *req* tells the servant what to do (see **WAITREQ** on page 220).

xarg Is a function or subroutine (SUBROUTINE) used to transfer arguments. The *xarg* routine is started when the task is physically dispatched.

xres Is a function or subroutine (SUBROUTINE) used to transfer results. *xres* is started when results are to be retrieved.

narg Is an input integer (INTEGER*4) representing the number of arguments.

args Is an input pointer to an array of pointers to arguments.

task Is an output integer (INTEGER*4) representing the task number.

a1 Is an input of any type (ANYTYPE) that is the first argument to send or get.

a2 Is an input of any type (ANYTYPE) that is the second argument to send or get.

Return Values

Upon successful completion, the task ID is returned. If the function completes unsuccessfully, –1 is returned.

SETFAM—Setting the Family ID

Description

SETFAM sets the current family.

Syntax

Language	Programming Syntax
C	**int SETFAM (int** *family* **)**
FORTRAN	**CALL SETFAM (** *family*, *ret* **)**

Where:

family Is an output integer (INTEGER*4) representing the family identifier.

ret Is an output integer (INTEGER*4) representing the previous family identifier.

Return Values

Upon successful completion, the previous family ID is returned. If the function completes unsuccessfully, a –1 is returned.

SETSRV—Forcing Data Transfer to the Servant ID

Description

SETSRV tells the **MS*xxxx*** and **SM*xxxx*** who the servant is. Use it to cause subsequent data transfers to go to the servant specified by *srv*. Zero indicates master. **SETSRV** is needed only if the application chooses not to use **SCHEDUL** or **SYNCHRO**.

Syntax

Language	Programming Syntax
C	**int SETSRV (int** *srv* **)**
FORTRAN	**CALL SETSRV (** *srv, ret* **)**

Where:

srv Is an input integer (INTEGER*4) representing the servant identifier.

ret Is an output integer (INTEGER*4) representing the return code.

Return Values

Upon successful completion, the previous process ID is returned. If the function completes unsuccessfully, a –1 is returned.

SMANY or SMxxxx—Transferring Data from Servant to Master

Description

SMANY or **SM*xxxx*** transfers *nitem* chunks of data from a servant process to a master. The parameters are the same as for **MS*xxxx***.

If your master process has multiple servants or your program uses the CLIO/S tape interface subroutine **ctpalloc**, specify the target servant by calling the SETSRV subroutine before calling **MSANY** or **MSxxxx**. See “SETSRV—Forcing Data Transfer to the Servant ID” for more information.

Syntax

Language	Programming Syntax
C	**int SMANY (int** *dt***, void ****buf***, int** *nitem***, int** *itemsize***, int** *stride* **)**
FORTRAN	**CALL SM*xxxx* (** *buf***,** *nitem***,** *itemsize***,** *stride***,** *ret* **)**

Where:

dt Is an input integer (INTEGER*4) representing the datatype being transferred. The following values are accepted:

0 OPAQ
1 CHAR
2 INTE
3 REAL
4 DOUB
5 LOGI

buf Is a pointer to the array being transferred.

nitem Is an input integer (INTEGER*4) representing the number of items being transferred.

itemsize Is an input integer (INTEGER*4) representing the size of each item being transferred.

stride Is an input integer (INTEGER*4) representing the distance between items.

ret Is an output integer (INTEGER*4) representing the return code.

xxxx can be one of the following:

OPAQ Represents no data conversions done.
CHAR Represents character data.
INTE Represents INTEGER*4.
LOGI Represents LOGICAL*4.
REAL Represents single-precision floating point.
DOUB Represents double-precision floating point.

These macros, defined in **fcfdpi.h**, simplify starting **SMANY**:

ntrans = SM_char (*buf* **)**
ntrans = SM_long (*buf* **)**
ntrans = SM_logical (*buf* **)**
ntrans = SM_float (*buf* **)**
ntrans = SM_double (*buf* **)**
ntrans = SM_opaque (*buf*, *itemsize* **)**
ntrans = SM_char_a (*buf*, *nitem* **)**
ntrans = SM_long_a (*buf*, *nitem* **)**
ntrans = SM_logical_a (*buf*, *nitem* **)**
ntrans = SM_float_a (*buf*, *nitem* **)**
ntrans = SM_double_a (*buf*, *nitem* **)**

Return Values

Upon successful completion, the number of bytes transferred is returned. If the function completes unsuccessfully, a –1 is returned. If CLIO/S couldn't process all the requested bytes, the function returns –205.

SM_flush or SMFLSH—Confirming Data Was Sent from Servant to Master

Description

SM_flush or **SMFLSH** allows you to ensure that data is actually sent from the servant process to a master.

Syntax

Language	Programming Syntax
C	**int SM_flush (void)**
FORTRAN	**CALL SMFLSH**

Return Values

Upon successful completion, a 0 is returned. If the function completes unsuccessfully, a −1 is returned.

SPANY or SPxxxx—Sending Data to a Process

Description

SPANY or **SP*xxxx*** sends data to a process in the current family. If the connection is not already established, CLIO/S attempts to connect to the process. **SPANY** or **SP*xxxx*** sends *nitem* chunks of data to process *prc*. *prc* specifies which process data is sent to (0 is master, 1 is servant1, and so on.) The parameters are the same as for **MSANY** or **MS*xxxx***. See "MSANY or MSxxxx—Transferring Data from Master to Servant" on page 206 for more information.

Syntax

Language	Programming Syntax
C	**int SPANY (int** *prc*, **int** *dt*, **void** ***buf*, **int** *nitem*, **int** *itemsize*, **int** *stride* **)**
FORTRAN	**CALL SP*xxxx* (** *prc*, *buf*, *nitem*, *itemsize*, *stride*, *ret* **)** **CALL SPANY (** *prc*, *dt*, *buf*, *nitem*, *itemsize*, *stride*, *ret* **)**

Where:

prc Is an input integer (INTEGER*4) representing the process identifier.

dt Is an input integer (INTEGER*4) representing the datatype. The following values are accepted:

0 OPAQ
1 CHAR
2 INTE
3 REAL
4 DOUB
5 LOGI

buf Is a pointer to the buffer

nitem Is an input integer (INTEGER*4) representing the number of items being transferred.

itemsize Is an input integer (INTEGER*4) representing the size of each item being transferred.

itemsize Is an input integer (INTEGER*4) representing bytes per item.

stride Is an input integer (INTEGER*4) representing the distance between items being transferred.

ret Is an output integer (INTEGER*4) representing the return code.

xxxx can be one of the following:

CHAR Represents character data.
INTE Represents INTEGER*4.
LOGI Represents LOGICAL*4.
REAL Represents single-precision floating point.

DOUB Represents double-precision floating point.

The following macros in **fcfdpi.h** simplify calling **SPANY**:

ntrans = SP_char (*buf* **)**
ntrans = SP_long (*buf* **)**
ntrans = SP_logical (*buf* **)**
ntrans = SP_float (*buf* **)**
ntrans = SP_double (*buf* **)**
ntrans = SP_opaque (*buf*, *itemsize* **)**
ntrans = SP_char_a (*buf*, *nitem* **)**
ntrans = SP_long_a (*buf*, *nitem* **)**
ntrans = SP_logical_a (*buf*, *nitem* **)**
ntrans = SP_float_a (*buf*, *nitem* **)**
ntrans = SP_double_a (*buf*, *nitem* **)**

Return Values

Upon successful completion, the number of bytes sent is returned. If the function completes unsuccessfully, –1 is returned. If CLIO/S couldn't process all the requested bytes, the function returns –205.

SYNCHRO—Waiting Until All or Selected Tasks are Completed

Description

SYNCHRO waits until all or a particular task has completed. You can also use **SYNCHRO** to examine the task queue. **SYNCHRO** is intended for asynchronous requests.

Syntax

Language	Programming Syntax
C	**void SYNCHRO (int** *task* **)**
FORTRAN	**CALL SYNCHRO (** *task*, *ret* **)**

Where:

task Is an input integer (INTEGER*4) representing:

>0 The task number.

=0 Wait until all tasks are complete.

<0 Examine the task queue.

ret Is an output integer (INTEGER*4) that is unused but reserved.

Return Values

This routine has no return value.

TSKEXIT—Informing a Master the Servant Process is Finished

Description

TSKEXIT allows an application-provided routine, pointed to by *taskexit*, to be called when a task is completed. This attribute is assigned to the task by **SCHEDUL**, and therefore, is only effective for tasks scheduled after the call to **TSKEXIT**.

Syntax

Language	Programming Syntax
C	**void TSKEXIT (Voidfunc** *taskexit* **)**
FORTRAN	There is no FORTRAN equivalent.

Where:

taskexit Is a pointer to the task exit. The function pointed to by *taskexit* has the following prototype: **void tskexit (int** *taskid*, **int** *status* **)**

where:

taskid Identifies the task.

status Represents whether CLIO/S completed the task. A zero indicates the task was completed, all other values indicate the task was not dispatched.

Return Values

This routine has no return value.

WAITFOR—Waiting Until an Alert is Available

Description

WAITFOR polls the servants specified by *group* for ALERTs and waits until an alert is available. **ALERT** and **WAITFOR** allow the application to perform its own work management. **WAITFOR_t** is defined in **fcfdpi.h**.

Syntax

Language	Programming Syntax
C	**WAITFOR_t *WAITFOR (int** *group***, int ***block* **)**
FORTRAN	**CALL WAITFOR (** *group***,** *block***,** *id***,** *code***,** *family* **)**

Where:

group Is an input integer (INTEGER*4) representing the alert group identifier defined by **ALERTGRP** or one of the special values 0 and 1. 0 means master, and 1 means master or any servant. A negative value is interpreted as a servant identifier.

block Is an input integer (INTEGER*4) representing whether to wait for an ALERT. If *block* is an integer other than zero then wait until an ALERT is available.

id Is an output integer (INTEGER*4) representing the sender's identifier.

code Is an output integer (INTEGER*4) representing code from the corresponding ALERT.

family Is an output integer (INTEGER*4) representing the family ID.

Return Values

Upon successful completion, the C routine returns a pointer to a **WAITFOR_t** structure. If the function completes unsuccessfully, the C routine returns a NULL pointer.

The FORTRAN routine returns the components of the **WAITFOR_t** structure in the output variables *id*, *code*, and *family*. If unsuccessful, it sets *id* and *code* to –1.

WAITREQ—Indicating a Servant is Waiting for Work

Description

WAITREQ allows you to indicate that a servant is waiting for work. A **SCHEDUL** by the master should correspond to a **WAITREQ** by the servant. The convention is when req is positive, the servant fetches the arguments, performs the indicated action, and then again calls **WAITREQ**. When the master is ready to receive the results, it automatically dispatches a task with the same request code, but negative. The servant then sends the results back.

Syntax

Language	Programming Syntax
C	**int WAITREQ (void)**
FORTRAN	**CALL WAITREQ (** *req* **)**

Where:

req Is an output integer (INTEGER*4) representing the request code. This is the *req* provided as input to the **SCHEDUL** routine.

Return Values

Upon successful completion, the request code is returned. A request code of 0 indicates that there is no master or currently active family; the servant can exit the **WAITREQ** loop and continue processing. If the function completes unsuccessfully, a –1 is returned.

Changing the Buffering Mechanism

Data transfer calls usually move data into a buffer. The buffer is flushed when it is full or when a call to **MS_flush** or **SM_flush** is made. The default buffer size is 32768 (32K) bytes and can be changed by calling **CLIOBUF** or by setting the environment variable **CLIOBUF** to the desired value. The minimum size of the buffer is 1024 (1K) bytes. When sending large blocks of data, it is usually desirable to bypass the buffering mechanism. CLIO/S uses the following rule: If the number of bytes of data is greater than or equal to **CLIOBLK**, flush the buffer and send the data. If this case, no copying of data is done. By setting **CLIOBLK** to zero, the CLIO/S buffering mechanism is bypassed entirely.

Buffered and unbuffered data transfers can be mixed. The master and the servant must have the same buffer sizes. The values are sent to the servants when **OPNPAR** is called. Any previous settings are erased. After **OPNPAR** has been called, **CLIOBUF** and **CLIOBLK** return the current values, but do not modify them.

It is advantageous to use the CLIO/S DPI buffering mechanism when sending multiple small data items. However, in some cases, it may degrade performance, because CLIO/S has to first send the size of the buffer and then send the actual data. To reduce this problem, the first 40 bytes of the buffer is sent with the header, resulting in only one send operation for very small buffers.

Using CLIO/S ESTAE Support

CLIO/S DPI provides ESTAE support on MVS to stop servants started by an MVS master program. By default, **CLSPAR** is called to stop all servants. To have a cancelled MVS servant stop its master, install a user ESTAE handler by calling **cliostae**. This handler can stop its master by obtaining its process ID and internet address (*hid*) from the CLIO_t structure and calling **clio_kill**. A pointer to the structure is obtained by calling **cdpi_s2c(0)**.

ESTAE support is provided through an assembler function **cliostae**. This function accepts one argument, *myhandler*, which is a pointer to a function. The first time **cliostae** is called, an ESTAE environment is established. Subsequent calls can be used to change the handler, if desired. The call returns the address of the previous handler. **cliostae** detects whether it is called from C or FORTRAN. When an ABEND occurs, the handler is called with the appropriate language environment. The handler needs the following prototype:

Language	Programming Syntax
C	**void myhand (int** *abend***, int** *psw1***, int** *psw2***, int *** *r13* **)**
FORTRAN	**SUBROUTINE MYHAND (** *iabend***,** *ipswd1***,** *ipsw2***,** *ir13* **)**

The contents of *abend* is 0x'*ffaaarrr*', where *ff* is the flag byte from SDWACMPF, *aaa* is the ABEND code, and *rrr* is the reason code. *psw1* and *psw2* represent the two parts of the Program Status Word (PSW) at the time of the ABEND. *r13* represents register 13 at the time of ABEND.

The handler established by CLIO/S DPI during startup prints a traceback and then attempts to close the CLIO/S environment, stopping all servants started with **OPNSRV**. The traceback is provided through **cliotrce()**, which can also be called from the application. It is available only as C subroutines, and takes one argument, *r13*, which is a pointer to a SAVEAREA. If the argument is NULL, the current register 13 is used.

When using the FORTRAN interface the ESTAE environment established by CLIO/S is cancelled when control is returned to FORTRAN. A possible bypass for this is to call **cliostae** before calling any other CLIO/S subroutine. For example:

```
EXTERNAL MYHAND
CALL CLIOSTAE ( 0 )          Establish ESTAE, no handler
CALL OPNPAR ( IOS )          Establish CLIO/S default handle
CALL CLIOSTAE ( MYHAND )     Optional, override the CLIO/S handle
```

Calling **cliostae** with a parameter of 0 causes the ESTAE exit to percolate with no further action. Similarly, a user ESTAE handler may check the abend code to determine if any action should be performed. It should always return to the caller. **CLSPAR** can be called to stop servants and clean up the environment.

The output produced by the default CLIO/S DPI ESTAE handler is similar to Figure 29, where the program was cancelled while waiting for a CTC read. The Anchor and NAB fields are present for C routines and are blank for other routines. If the name cannot be determined, `Unknown` is printed.

```
EPACMVS:cdpistax_0: CLiO ESTAE active. ABEND=40222000 PSW=078D0E00 8051F6BA
-----------------------------------------------------------------
 Savearea Return   Entry    Anchor   NAB      Name
-----------------------------------------------------------------
 0353d2d8 8353d1da 0051f644                   RS6KWTR
 00010ea0 83535176 0353d1b6                   RS6KWT
 00010df0 8352a2c8 03534290 00010600 00010ea0 rctc
 00010d48 83514b4e 0352a130 00010600 00010df0 ctc_rd
 00010c80 83514960 03514a60 00010600 00010d48 do_clio_rd
 00010bc8 8350d782 03514780 00010600 00010c80 clio_rd
 00010b20 8350d4dc 0350d690 00010600 00010bc8 buf_get
 00010a60 8350ce56 0350d3e0 00010600 00010b20 buf_read
 000109a8 8350993c 0350ca78 00010600 00010a60 do_read
 00010888 835011aa 035096a0 00010600 000109a8 cdpixsma
 000104d0 80008f82 03500e50 00010600 00010888 main
 00005fa0 80fd41b8 0350dddc                   Unknown
-----------------------------------------------------------------
```

Figure 29. Sample Output Produced by Default CLIO/S DPI ESTAE Handler

Notes:

1. The CLIO/S DPI ESTAE environment is cancelled after its first call. To restart it, call **cliostae** again.
2. The library does not establish an ESTAE environment when the environment variable is set to *no* (CLIOSTAE=NO).
3. To prevent the CLIO/S DPI ESTAE environment from being established at all times, call **cliostae(−1)** before calling any other CLIO/S routines.

Chained Channel Programs for BMCA and ESCON

You can improve performance in an application running on MVS by chaining together read and write operations using the **rw_dpi** call to the DPI. The **rw_dpi** call also supports TCP/IP connections and runs on the RS/6000, but does not provide any performance benefit in those environments.

Returning the Protocol Type

Use this call if the protocol is P_PCA or P_ESCON and the application needs the **rw_dpi** call. To obtain the protocol type, use the **cdpi_s2c** routine, as shown in the following code segment:

```
isrv = OPNSRV( mach, 1, servername, 0 )
infoptr = cdpi_s2c(isrv);
if (infoptr -> proto == P_ESCON) printf("Protocol is ESCON.");
if (infoptr -> proto == P_PCA  ) printf("Protocol is PCA.  ");
if (infoptr -> proto == P_TCPIP) printf("Protocol is TCPIP.");
```

The **ISCTC** macro defined in **fcfcmi.h** can be used for the test. Additional information about **cdpi_s2c** is in “cdpi_s2c—Returning a CLIO/S Pointer” on page 196.

Starting a Series of Read or Write Operations

The **rw_dpi** call sends multiple read and write requests to the DPI. “rw_dpi—Reading and Writing Data” on page 212 describes how to call **rw_dpi**. Make a call similar to the following in your program:

```
idum = rw_dpi( procid, rw_dpi_a, numspecs );
```

`rw_dpi_a` is a pointer to an array of read and/or write operations of type **RW_DPI**, as defined in **fcfcmi.h**. Each RW_DPI structure contains this information:

```
cmd    = command (RDCTC for read, WRCTC for write)
types  = 0       /* reserved field; must be 0 */
length = buffer length
bufptr = pointer to buffer
```

The following figure shows a sample array for writing one 4K block of data, followed by reading two more 4K blocks of data:

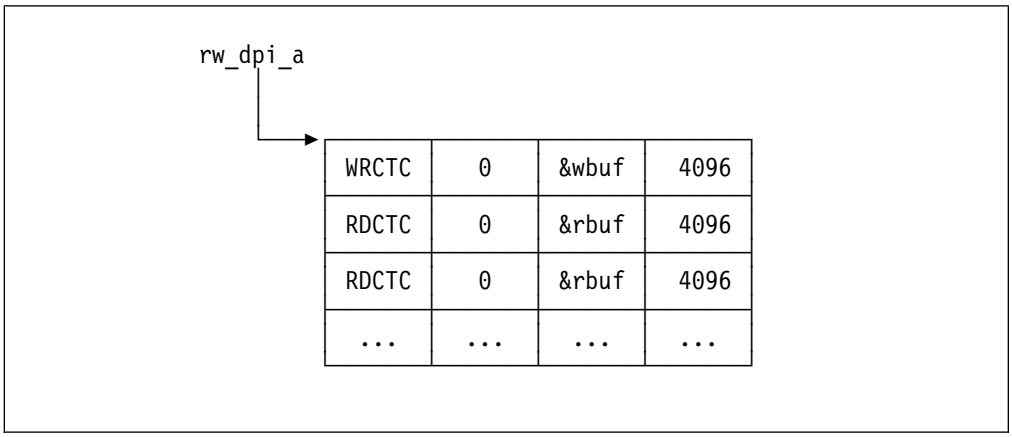


Figure 30. Example of Array of RW_DPI Structures for Chained Reads and Writes (WRCTC=write RDCTC=read)

Using the rw_dpi Call

Figure 31 on page 224 shows sample code using **rw_dpi** to do three chained I/O operations. The RWDPI macro builds the read/write structure and the **rw_dpi** call issues the I/O. This code runs as both master and servant.

```
#define RD ( (myid) ? WRCTC : RDCTC )  /* Read  if master */
#define WR ( (myid) ? RDCTC : WRCTC )  /* Write if master */
#define RWDPI(p,c,b,l) (p)->cmd=c, (p)->types=0, \
   (p)->bufptr=(char*)b, (p)->length=l

myid = OPNPAR(0);
if ( myid > 0 ) {   /* Start slave if master */
   /*
    * try to start a copy of self on 'node'
    */
   if ( (isrv=OPNSRV(FNDMACH(node),2,argv[0],0)) > 0 ) {
      printf("OPNSRV failed. node=%s rc=%i\n", node, isrv );
      return COPNSRV_ERR;
   }
}
/*
 * fill in the read/write structure
 */
for ( i=0, s=buf; i<nbuf; i++, s+=nbyte ) {
   RWDPI( &rw[i], dir ? WR : RD, 0, s, nbyte );
}
/*
 * start the data transfer
 */
iret = rw_dpi( (myid) ? 0 : isrv, rw, i );
if ( iret ) ERRPRT("rwtst",fprintf(stderr,"iret=%d\n",iret) );
```

Figure 31. Sample Code for Using the rw_dpi Call.

Chapter 12. Using the CLIO/S Tape Programming Interface and Support

Tape Device Note

Different tape devices have different requirements for reading, writing, and tape manipulation. For example, /**dev/rmt0** and /**dev/rmt0.1** behave differently, even though they are the same physical device. CLIO/S has no knowledge of specific tape device requirements. Application developers must be aware of the particular requirements of the devices they are using.

The CLIO/S tape programming interface provides access to any tape device attached to a RS/6000 or a MVS/ESA system in a CLIO/S network. All the standard tape operations are available through the interface. The tape interface supports routing, where tape drives may be reached by going through multiple machines. The CLIO/S package includes a sample program **fcfttest.c**, which demonstrates the interface by providing a command line interface to the tape API. See “Using the FCFTTEST Sample Program” on page 242 for more information.

Chapter 4, “Configuring and Managing the CLIO/S Tape Pseudo Device Driver” on page 45 describes the AIX command line interface to the CLIO/S tape server.

The CLIO/S Tape Header File fcftpint.h

The header file **fcftpint.h** contains definitions required by the CLIO/S tape C functions. You should **#include** it in your tape application. You can find the header file in the following locations:

MVS **SYS1.SFCFHDRS(FCFTPINT)**

AIX **/usr/lpp/fcf/inc/fcftpint.h**

Sample CLIO/S Tape Program

The CLIO/S tape interface is one instance of a CLIO/S DPI application. Before reading this section, it is helpful to understand how the DPI works. See “The CLIO/S DPI Sample Program” on page 192 for more information.

The following code is used as an elementary example. Line numbers have been added for reference. This example uses the same environment as Chapter 11, “Using the Distributed Processing Interface (DPI)” on page 187. The following code will run on workstation ***riscnode***, and it will write a message to a tape drive on the MVS system ***mvsnode***.

© Copyright IBM Corp. 1993, 1995, 1996

```
 1 #include <fcftpint.h>           /* Tape interface declarations */
 2 main()
 3 {
 4   int tlu;
 5   char *tapedata = "Data for Tape";
 6   ctpalloc(&tlu,"mvsnode",512,\
 7      "DD=TAPE93 DSN=CLIOTP VOL=CLIO UNIT=CTAPE NEW NL","dummy");
 8   ctpopen(tlu,"volser", 2);
 9   ctprew(tlu);
10   ctpwrite(tlu, tapedata, strlen(tapedata));
11   ctpclose(tlu,0);
12   ctpunall(tlu);
13   exit(0);
14 }
```

Figure 32. Example Tape Interface C Code

Line 1, `#include <fcftpint.h>`, includes the definitions used to call the CLIO/S tape functions. This header must be included by any application that uses the tape interface.

Line 6, `ctpalloc(tlu,"mvsnode",512,"DD=TAPE93 DSN=CLIOTP VOL=CLIO UNIT=CTAPE NEW NL","dummy");` , initializes the tape access environment. **ctpalloc** looks at the node value of ***mvsnode*** and realizes that it is not the local node. **ctpalloc** then issues a CLIO/S DPI **OPNSRV** function, directing CLIO/S to start the `TAPESRV` process on MVS. `TAPESRV` in turn calls another copy of **ctpalloc** on the remote MVS node, and passes it the same allocation parameters that were originally used. **ctpalloc** sets the value of *tlu* before returning.

Line 8, `ctpopen(tlu,"volser", 2);`, opens the desired unit for writing. Again, the tape operations passed to the local copy of **ctpopen** are passed by the DPI to the remote copy of **ctpopen** running on MVS.

Note that **ctpalloc** references the address of *tlu* (`tlu`) while the other functions reference its value (`&tlu`).

The tape interface uses the value of *tlu* to determine which tape device is being accessed. Applications can allocate and open multiple devices, keeping a separate *tlu* value for each one.

Subsequent tape operations are treated similarly. Along with the tape operations, any data to be read or written will be transmitted between systems by the high-speed CLIO/S transfer functions.

Using the CLIO/S Tape Interface Functions

The CLIO/S tape calls and functions are described in this section. The calls and functions are available in C and FORTRAN. The following table lists the CLIO/S tape subroutines and describes the function provided by each.

Subroutine			
C	**FORTRAN**	**Description**	**Page**
ctpalloc	tpalloc	Starting the Remote Tape Server	227

Subroutine			
C	**FORTRAN**	**Description**	**Page**
ctpclose	tpclose	Closing the Tape	230
ctpopen	tpopen	Opening the Tape Device for Use	231
ctpread	tpread	Reading the Tape	232
ctprew	tprew	Rewinding the Tape	233
ctpskpf	tpskpf	Skipping Tape Files	233
ctpskpr	tpskpr	Skipping Tape Records	234
ctpunall	tpunall	Stop the Remote Tape Server	235
ctpvols	tpvols	Getting a List of Tape Volumes	235
ctpweof	tpweof	Writing Tape Marks	236
ctpwrite	tpwrite	Writing to tape	237
ctpwrto	tpwrto	Writing to Operator	237

ctpalloc or tpalloc—Starting the Remote Tape Server

Description

The tape allocation call is required by CLIO/S to set up internal control blocks and start the remote tape server.

Syntax

Language	Programming Syntax
C	**int ctpalloc (int ****tlu***, char ****node***, int** *maxdata***, char ****allocparm***, char ****utility** **)**
FORTRAN	**tpalloc (** *tlu*, *node*, *maxdata*, *allocparm*, *utility*, *ret* **)**

Where:

tlu Is a pointer to an integer (INTEGER*4) representing the tape logical unit number. CLIO/S fills in this value during the allocation routine. It is used by subsequent tape calls to refer to the device. Note that the other calls use the value of *tlu* rather than its address (&*tlu*). An application or group of applications can allocate up to 10 tape units with successive calls to this routine.

node Is a pointer (CHARACTER*(*)) to a network node where the tape station resides. The node names are TCP/IP node names or CLIO/S nicknames, as defined in the PARCONF file.

You can also specify a route of machines, such as A:B:C, where the tape resides on machine C, which is reached by going through machines A and B. If A and B (or B and C) are RS/6000 nodes, they must be running the same shell.

"A:B:C" corresponds to a nickname in the PARCONF file on your local system, as described in "The PARCONF File" on page 26. In addition, the PARCONF file on each intermediate host must contain an entry defining the nickname for the downstream host. For example, the PARCONF file on the local host contains the routed

nickname “A:B:C” and the nicknames “A”, “B”, and “C”. The PARCONF file on host A contains an entry defining the nickname “B”; the PARCONF file on host B contains an entry defining the nickname “C”.

maxdata Is an input integer (INTEGER*4) representing the maximum record length on the tape.

allocparm Is a pointer (CHARACTER*(*)) to the allocation parameters. It is limited to 256 bytes in length. For AIX, the allocation parameter is simply the special file name of the desired device, for example:

```
/dev/rmt2
```

For MVS, the allocation parameter specifies the unit name and other allocation data for the dynamic allocation of the tape. MVS allocation parameters are listed in Appendix E, “Dynamic Device Allocation on MVS” on page 303.

```
DD=TAPE01 VOL=LADUKS UNIT=3480 NEW NL
```

CLIO/S has tested the following MVS tape allocation parameters:

CA — Indicates that you want to use MVS cataloged tape data sets.

COMP, NOCOMP — Turns IDRC compaction on (COMP) or off (NOCOMP).

If neither COMP nor NOCOMP is specified, the default will be the installation default.

DCBDS=*model.dataset.name* — Specifies the name of the model data set. This allocation parameter is supported only for GDG data sets.

DD=*tape-DD-name* — Specifies the tape DD name

DSN=*dataset-name* — Specifies the tape data set name

DSN=*qual1.qual2* — Specifies the stem of a GDG data set name. This allocation parameter is supported only for GDG data sets.

D800,D1600,D6250 — Specifies the tape density as 800, 1600, or 6250 BPI

EX=*yyddd* — Specified the expiration date

MEMBER=+1 — Specifies the suffix increment for the next GDG data set name. This allocation parameter is supported only for GDG data sets.

NEW,OLD — Specifies whether to create new data set or reuse an old one

NL,SL,AL,BLP — Specifies tape label attributes

Note: For AL (ANSI standard label) tapes, CLIO/S supports ANSI version 3 standard labels.

RET=*num* — Specifies the retention period

SUP_EOV — Ensures that CLIO/S gives control back to the application when hitting End of Tape on input or output. If SUP_EOV is not specified, CLIO/S will automatically do end of volume processing when the user specifies a dataset name but no volser (see "Effects of MVS Tape Management on CLIO/S" on page 241 for more information).

UNIT=*unit-name* — Unit name

VOL=*volser-name* — Specifies the tape volser. Do not specify this option if you are using MVS tape management.

IBM has tested the following combinations of the DD, VOL, UNIT, NEW | OLD, and NL | AL | BLP | SL allocation parameters:

- DD=*tape-DD-name* VOL=*volser-name* UNIT=*unit-name* NEW NL
- DD=*tape-DD-name* VOL=*volser-name* UNIT=*unit-name* OLD NL
- DD=*tape-DD-name* VOL=*volser-name* UNIT=*unit-name* OLD AL
- DD=*tape-DD-name* VOL=*volser-name* UNIT=*unit-name* OLD BLP
- DD=*tape-DD-name* VOL=*volser-name* UNIT=*unit-name* OLD SL DSN=*dataset-name*
- DD=*tape-DD-name* VOL=*volser-name* UNIT=*unit-name* NEW SL DSN=*dataset-name* EX=*yyddd*
- DD=*tape-DD-name* VOL=*volser-name* UNIT=*unit-name* NEW SL DSN=*dataset-name* RET=*num*

utility — Represents a pointer (CHARACTER*(*)) to information for user exit FCFTPX00 on MVS and **ctpexit** on AIX. The maximum length of this parameter is 256 bytes. This exit is called before allocation and open and after close and unallocation. See "Tape Exit On MVS and AIX" on page 240 for more information.

ret — Is an integer (INTEGER*4) representing the return value.

Return Values

This function returns one of the following return values:

0	OK
–6	COMM_ERR
–10	ALLINUSE
–11	UNKNOWN_NODE
–12	COPNPAR_ERR
–13	COPNSRV_ERR
8	The installation validation routine denied this request.
10	An error occurred during allocation.
16	An internal CLIO/S error occurred. Follow your local procedures for reporting problems.

20, 24, 28 These return values indicate an error found while processing the allocation parameters. These return values require special testing. The actual return value is located in the low-order 3 bytes of the return value. The first byte contains the index of the allocation parameter which was in error. To get the index of the allocation parameter which is in error, shift the return value to the right by 24 bits. To test for the return value, you can clear out the first byte of the return value by AND-ing the value with `x'00FFFFFF'`.

20 (x'14') An invalid keyword was specified in the allocation parameters.
24 (x'18') An internal CLIO/S error has occurred. Follow your local procedures for reporting problems.
28 (x'1C') An invalid subparameter was specified after the equal sign (=) in an allocation parameter specification.

other Other values are the ERROR REASON codes returned by SVC 99, which are described in the *MVS/ESA SP V4 Authorized Assembler Programming Guide*, GC28-1645. Some common SVC 99 ERROR REASON codes include:

528 The requested data set is unavailable because it is allocated to another job and its usage attribute conflicts with this request.
532 The requested unit is not available.
540 The requested unit name is undefined.
544 The requested volume is not available.
552 The requested volume or unit is in use by the system.
860 An invalid parameter was specified in a text unit.
888 A duplicate KEY was specified in a text unit.
892 An invalid length was specified in a text unit.
1156 The request was denied by the operator.

ctpclose or tpclose—Closing the Tape

Description

This call writes an end-of-volume mark and rewinds the tape. Under MVS, it unmounts the tape volume. On a RS/6000, it frees the device; the next volume can be mounted manually or automatically using the *option* parameter.

Syntax

Language	Programming Syntax
C	**int ctpclose (int** *tlu*, **int** *option* **)**
FORTRAN	**tpclose (** *tlu*, *option*, *ret* **)**

Where:

tlu Is an integer (INTEGER*4) representing the tape logical unit number provided by the **ctpalloc** call.

option Is an integer (INTEGER*4) representing the stacker option, which is required when using the STK4280 on the RS/6000. The following are valid values for the *option* parameter:

0 Ignore this option.
–2 Load the next cartridge into the stacker using automatic mode.
–3 Leave the tape drive in the current position during and after the close (avoid rewinding).

ret Is an integer (INTEGER*4) representing the return value.

Return Values

This function returns one of the following return values:

0	OK
–6	COMM_ERR
other	The errno value returned by the failing **close** function.

See “CLIO/S Tape Interface Return Values” on page 239 for more information about the return values.

ctpopen or tpopen—Opening the Tape Device for Use

Description

This call prepares for reading, writing, or updating (reading and writing). See “Effects of MVS Tape Management on CLIO/S” on page 241 about special considerations arising from MVS tape management.

Syntax

Language	Programming Syntax
C	**int ctpopen (int** *tlu*, **char ***_volser_, **int** *mode* **)**
FORTRAN	**tpopen (** *tlu*, *volser*, *mode*, *ret* **)**

Where:

tlu Is an integer (INTEGER*4) representing the tape logical unit number provided by the **ctpalloc** call.

volser Is an pointer (CHARACTER*(*)) to the volume serial number being opened.

AIX This is the name of the tape device, for example, /**dev**/**rmt0**.

MVS If you are not using MVS tape management, this is the volser of the tape. When using MVS tape management, specify the volser as a string of blanks.

mode Is an integer (INTEGER*4) specifying the tape mode:

1 Input: open for input and allow rewinding
2 Output: open for output and allow rewinding
3 Update: open for update (read/write) and allow rewinding
4 Input with no rewind: open for input and avoid rewinding
5 Output with no rewind: open for output and avoid rewinding
6 Update with no rewind: open for update (read/write) and avoid rewinding
7 Output extend: open for update (read/write) and prepare to extend the tape (the same as DISP=MOD on a DD statement)

ret Is an integer (INTEGER*4) representing the return value.

Return Values

This function returns one of the following return values:

0	OK
–6	COMM_ERR
–17	NOTPSRV
other	The errno value returned by the failing **open** function.

See “CLIO/S Tape Interface Return Values” on page 239 for more information about the return values.

ctpread or tpread—Reading the Tape

Description

This call reads data from an open tape device.

Syntax

Language	Programming Syntax
C	**int ctpread (int** *tlu*, **char ***buffer*, **int** *length* **)**
FORTRAN	**tpread (** *tlu*, *buffer*, *length*, *ret* **)**

Where:

tlu Is an integer (INTEGER*4) representing the tape logical unit number provided by the **ctpalloc** call.

buffer Is a pointer (CHARACTER*(*)) to the size of the buffer.

length Is an integer (INTEGER*4) representing the number of bytes being read from the tape.

ret Is an integer (INTEGER*4) representing the return value. If the return value is greater than or equal to zero, it is the number of bytes read.

Return Values

If successful, this function returns the number of bytes read. If unsuccessful, it returns one of these return values:

12	Out of memory. **Note:** Avoid requests to read 12 bytes of data. You won't be able to determine whether the routine was successful.
–1	TEOF
–2	LOADPOINT
–3	EOV
–5	IOERR
–6	COMM_ERR
–7	INVALPARM - The record length was greater than the maximum record length specified on **ctpalloc** or **tpalloc**, or an invalid record length was specified.
–17	NOTPSRV

See “CLIO/S Tape Interface Return Values” on page 239 for more information about the return values.

ctprew or tprew—Rewinding the Tape

Description

This call rewinds the tape.

Syntax

Language	Programming Syntax
C	**int ctprew (int** *tlu* **)**
FORTRAN	**tprew (** *tlu*, *ret* **)**

Where:

tlu Is an integer (INTEGER*4) representing the tape logical unit number provided by the **ctpalloc** call.

ret Is an integer (INTEGER*4) representing the return value.

Return Values

This function returns one of the following return values:

0	OK
–6	COMM_ERR
–17	NOTPSRV
other	The errno value returned by the failing **ioctl** function.

See “CLIO/S Tape Interface Return Values” on page 239 for more information about the return values.

ctpskpf or tpskpf—Skipping Tape Files

Description

This function skips forward or backward the specified number of tape marks on the tape.

Syntax

Language	Programming Syntax
C	**int ctpskpf (int** *tlu*, **int** *count* **)**
FORTRAN	**tpskpf (** *tlu*, *count*, *ret***)**

Where:

tlu Is an integer (INTEGER*4) representing the tape logical unit number which was provided by the **ctpalloc** call.

count Is an integer (INTEGER*4) representing the number of tape marks to be skipped. This can be a positive or negative number.

ret Is an integer (INTEGER*4) representing the return value.

Return Values

This function returns one of the following return values:

0	OK
–1	TEOF
–2	LOADPOINT
–3	EOV
–4	EOT
–5	IOERR
–6	COMM_ERR
–17	NOTPSRV
other	The errno value returned by the failing **ioctl** function.

See “CLIO/S Tape Interface Return Values” on page 239 for more information about the return values.

ctpskpr or tpskpr—Skipping Tape Records

Description

This function skips forward or backward the specified number of tape records on the tape.

Syntax

Language	Programming Syntax
C	**int ctpskpr (int** *tlu*, **int** *count* **)**
FORTRAN	**tpskpr (** *tlu*, *count*, *ret* **)**

Where:

tlu Is an integer (INTEGER*4) representing the tape logical unit number which was provided by the **ctpalloc** call.

count Is an integer (INTEGER*4) representing the number of records to skip. The value can be positive or negative.

ret Is an integer (INTEGER*4) representing the return value.

Return Values

This function returns one of the following return values:

0	OK
–1	TEOF
–2	LOADPOINT
–3	EOV
–4	EOT
–5	IOERR
–6	COMM_ERR
–17	NOTPSRV
other	The errno value returned by the failing **ioctl** function.

See “CLIO/S Tape Interface Return Values” on page 239 for more information about the return values.

ctpunall or tpunall—Stopping the Remote Tape Server

Description

This call uses the tape unallocation call to stop the remote tape server and free internal control blocks. This is required by CLIO/S. Under MVS, dynamic tape unallocation is called.

Syntax

Language	Programming Syntax
C	**int ctpunall (int** *tlu* **)**
FORTRAN	**tpunall (** *tlu*, *ret* **)**

Where:

tlu Is an integer (INTEGER*4) representing the tape logical unit number provided by the **ctpalloc** call.

ret Is an integer (INTEGER*4) representing the return value.

Return Values

This function returns one of the following return values:

0 OK
–6 COMM_ERR

See “CLIO/S Tape Interface Return Values” on page 239 for more information about the return values.

ctpvols or tpvols—Listing the Tape Volumes

Description

The **ctpvols** call returns a list of the tape volumes being used. This call can be used in the following situations:

- When MVS tape management is being used and a multivolume tape is written.

  ctpvols can be called after **ctpclose** to get a list of the volumes that have been written. MVS tape management automatically mounts and unmounts multiple volumes while the multivolume tape is being written.

- When MVS tape management is being used and a multivolume tape is being read.

  ctpvols can be called after **ctpopen** to get the list of volumes that will be read. MVS tape management automatically mounts and unmounts multiple volumes while the multivolume tape is being read.

- When MVS tape management is not being used and multiple volumes have been written.

  ctpvols can be called after **ctpclose** and before **ctpunall** to get the list of volumes. In this case, you must specify the volumes in **ctpopen** calls, so the list of volumes may already be known without calling **ctpvols**.

See “Effects of MVS Tape Management on CLIO/S” on page 241 for more information.

Syntax

Language	Programming Syntax
C	**int ctpvols (int** *tlu*, **int ***nvols*, **char ***bf* **)**
FORTRAN	**tpvols (** *tlu*, *nvols*, *bf* , *ret* **)**

Where:

tlu Is an integer (INTEGER*4) representing the tape logical unit number, provided by the **ctpalloc** call.

nvols Is an integer (INTEGER*4) representing the number of tape volumes, returned by **ctpvols**.

bf Is a pointer (CHARACTER*(*)) to a buffer returned by **ctpvols**. The buffer is divided into 8-byte segments, with each 8-byte segment containing a volser.

ret Is an integer (INTEGER*4) representing the return value.

Return Values

This function returns one of the following return values:

0	OK
–1	TEOF
–6	COMM_ERR
–17	NOTPSRV

See “CLIO/S Tape Interface Return Values” on page 239 for more information about the return values.

ctpweof or tpweof—Writing Tape Marks

Description

This call writes tape marks as specified.

Syntax

Language	Programming Syntax
C	**int ctpweof (int** *tlu*, **int** *count* **)**
FORTRAN	**tpweof (** *tlu*, *count*, *ret* **)**

Where:

tlu Is an integer (INTEGER*4) representing the tape logical unit number provided by the **ctpalloc** call.

count Is an integer (INTEGER*4) representing the number of tape marks to be written to the tape.

ret Is an integer (INTEGER*4) representing the return value.

Return Values

This function returns one of the following return values:

0	OK
–6	COMM_ERR
–17	NOTPSRV
Other	The errno value returned by the failing **ioctl** function.

See “CLIO/S Tape Interface Return Values” on page 239 for more information about the return values.

ctpwrite or tpwrite—Writing to Tape

Description

This call writes data to an open tape device.

Syntax

Language	Programming Syntax
C	**int ctpwrite (int** *tlu*, **char** **buffer*, **int** *length* **)**
FORTRAN	**tpwrite (** *tlu*, *buffer*, *length*, *ret* **)**

Where:

tlu Is an integer (INTEGER*4) representing the tape logical unit number provided by the **ctpalloc** call.

buffer Is a pointer (CHARACTER*(*)) to the buffer.

length Is an integer (INTEGER*4) representing the number of bytes being written to the tape.

ret Is an integer (INTEGER*4) representing the return value.

Return Values

This function returns one of the following return values:

nnn	The number of bytes written.
–1	Write failed because of device error; see associated message for details about the error.
–2	LOADPOINT
–5	IOERR
–6	COMM_ERR
–7	INVALPARM
–17	NOTPSRV

See “CLIO/S Tape Interface Return Values” on page 239 for more information about the return values.

ctpwrto or tpwrto—Writing to the Operator

Description

This call writes messages to the operator, with or without a reply. See “Using TPWRTO Support On AIX” on page 246 for AIX-specific information about this function.

Syntax

Language	Programming Syntax
C	**int ctpwrto (int** *tlu*, **char *****node*, **char *****msg*, **char *****replybuf*, **int** *replylen*, **int** *replycode* **)**
FORTRAN	**tpwrto (** *tlu*, *node*, *msg*, *replybuf*, *replylen*, *replycode*, *ret* **)**

Where:

tlu Is an integer (INTEGER*4) representing the tape logical unit number. The value of *tlu* is interpreted as follows:

≥ 0 *tlu* represents the tape server through which message are sent. This value was returned by **ctpalloc** when the tape was allocated. If the tape is on a remote node, this call causes the existing TAPESRV on the node specified by *tlu* to display the message.

Because *tlu* specifies the tape server, the *node* parameter is not required for addressing purposes. On MVS, the *node* parameter is ignored in this situation. On AIX, the *node* parameter specifies the name of a device or file where you want the message to be displayed. For example, you could direct messages to a specific display by specifying /**dev**/**ptyp0** and to a file by specifying /**tmp**/**clio.tape.messages**.

< 0 A negative value of *tlu* indicates that a tape server does not already exist. (Use this if you need to send a message before allocating a tape drive.)

In this case, the *node* parameter identifies the node on which the message is to be displayed. If the tape is on a remote node, the call starts a new tape message server, WTOSRV.

node Is a pointer (CHARACTER*(*)) (CHARACTER*(*)) to a network node where the messages are sent. The node names are TCP/IP node names or CLIO/S nicknames, as defined in the PARCONF file. On AIX, this parameter is ignored if *tlu* contains a non-negative number.

msg Is a pointer (CHARACTER*(*)) to the message being sent to the operator.

replmsg Is a pointer (CHARACTER*(*)) to the reply message being returned to the user.

replylen Is an integer (INTEGER*4) specifying the maximum length of reply message.

replycode Is an integer (INTEGER*4) specifying whether the message requests a reply. Each message may be no longer than 100 bytes.

0 Sends a message without reply.
1 Sends a message with reply. If a reply is requested, the call blocks until the reply is received.

ret Is an integer (INTEGER*4) representing the return value.

Return Values

This function returns one of the following return values:

0	OK
–4	Unable to open device
–10	ALLINUSE
–11	UNKNOWN_NODE
–12	COPNPAR_ERR
–13	COPNSRV_ERR
–14	MSGTOOLONG
–17	NOTPSRV

See “CLIO/S Tape Interface Return Values” for more information about the return values.

CLIO/S Tape Interface Return Values

The return values from the CLIO/S tape interface functions are:

Name	Number	Message
OK	0	The operation completed successfully.
TEOF	–1	A tape mark was hit on a read. Write failed because of device error; see associated message for details about the error.
LOADPOINT	–2	The loadpoint was hit as it attempted to backspace past the beginning of the tape.
EOV	–3	A tape mark was hit twice on a read.
EOT	–4	End of the tape was hit on a write.
IOERR	–5	A tape I/O error occurred.
COMM_ERR	–6	A communication error in the CLIO/S network occurred.
INVALPARM	–7	An invalid parameter was entered in a CLIO/S call or function.
ALLINUSE	–10	All tape drives are in use. A maximum of 10 tape drives can be accessed at the same time.
UNKNOWN_NODE	–11	The node was not found in the PARCONF file.
COPNPAR_ERR	–12	The CLIO/S **OPNPAR** call failed.
COPNSRV_ERR	–13	The CLIO/S **OPNSRV** call failed.
MSGTOOLONG	–14	**ctpwrto** error. The message length exceeds the maximum 100 bytes.
NOTPSRV	–17	No connection to a tape server is available. **ctpalloc** is not called.

Tape Exit On MVS and AIX

CLIO/S provides a set of tape exit programs that you can tailor to meet the needs of your installation. There is a tape exit for use on MVS and another for use on AIX. The following sections describe these exits.

MVS Tape Exit FCFTPX00

FCFTPX00 is the CLIO/S tape exit for MVS/ESA. The assembler source for FCFTPX00 is shipped in **SYS1.SFCFSAMP(FCFTPX00)**. You modify this exit to tailor the CLIO/S tape package to your installation. The FCFTPX00 exit receives the following input:

FCFTPX00 (int ***tlu*****, int** ***when*****, char** ***volser*****, char** ***allocparm*****, int** ***asize*****,**
int ***alloccode*****, char** ***utility*****, int** ***size*****, int** ***volinfo*** **)**

Where:

tlu Is a pointer to the tape logical number. When the CLIO/S tape server is running *tlu* is zero. The tape logical number defines the local tape being used, for example, 0, 1, 2, 3, and so on.

when Is a pointer to a code specifying when the tape exit is called. The following values are available:

volser Is a pointer to the volume serial number of the tape currently in use.

allocparm Is a pointer to the allocation parameters. See Appendix E, “Dynamic Device Allocation on MVS” on page 303 for a list of supported allocation parameters.

asize Is a pointer to the size of the allocation parameters.

alloccode Is a pointer to a code defining whether new tape allocation parameters are being specified. To return the modified allocation parameters, change *alloccode* from 0 to 1.

utility Is a pointer to the user exit information set in the *utility* parameter in **ctpalloc**.

size Is a pointer to the size of *utility*.

volinfo Is a pointer to a two word field. Word 1 is the number of *volsers*. Word 2 a pointer to the *volser* list. The list is divided into eight-byte segments. Each segment contains one volume. For output tapes this information is available when **FCFTPX00** is called during a close or unallocation. For input tapes this information is not available.

AIX Tape Exit ctpexit

ctpexit is the CLIO/S tape exit for AIX. The source for **ctpexit** is shipped in the file **ctpexit.c** in the **/usr/lpp/fcf/samplet** directory. You can modify this program to tailor the CLIO/S tape package to your environment.

ctpexit (int *tlu***, int** *callcode***, char** ***unit*****, int** *maxdata***, char** ***utility*** **)**

Where:

tlu Is a pointer to the tape logical number. When the CLIO/S tapeserver is running *tlu* is zero. The tape logical number defines the local tape being used, for example, 0, 1, 2, 3, and so on.

callcode Is a code indicating when the tape exit is called. The following values are available:

unit Is a pointer to the name of the tape device (for example, /**dev**/**rmt0**).

maxdata Is an integer containing the maximum data length.

utility Is a pointer to the user exit information passed in the *utility* parameter in **ctpalloc**.

Effects of MVS Tape Management on CLIO/S

CLIO/S tape functions and their use differ somewhat depending on whether or not your allocation parameters request MVS tape management. The following table describes the differences.

	With MVS Tape Management	Without MVS Tape Management
Input	Volume lists are available to the user through **ctpvols** call after **ctpopen**.	The **ctpvols** call is not applicable.
	When CLIO/S reaches a tape mark, the EOV macro is issued to switch to the next volume. Tape marks are not reported back to you.	When CLIO/S reaches a tape mark, the tape mark (EOF or EOV), is reported back to you.
Output	Tape allocation is by data set name. The volume serial number, *volser*, should not be specified.	Tape allocation is by *volser*. The data set name is required for SL tapes or if the data set is cataloged.
	The tape management supplies the volume serial numbers and the used volume serial numbers which are available using the **ctpvols** call between **ctpclose** and **ctpunall.**	*volser* is specified in **ctpopen** and *volser*s are accumulated by CLIO/S on MVS/ESA.
	Data sets can be cataloged using the *allocation* parameter in **ctpalloc** by specifying NEW CA.	The data set can be cataloged through the FCFTPX00 **unalloc** exit by specifying CTLG in the *utility* parameter on **ctpalloc**.
	When CLIO/S reaches the end of the tape a new volume is started. The volume change is transparent to the CLIO/S application and only a **ctpvols** call after a close informs the user on the number of *volsers* used.	The *volser* list is available through **ctpvols** call between **ctpclose** and **ctpunall**. The **ctpvols** call is not required if all *volsers* are specified.
	If you write tape marks followed by data, you can not read data written after the tape mark if you allocate the tape by dataset name. To read the data, allocate by *volser*, without specifying the data set name.	Tape marks can be written anywhere on the tape. You can read beyond the tape marks.
	MVS automatically mounts a new tape when you reach the end of the current tape. A single open and close is required.	Multivolume support is done using multiple opens and close, as shown in Figure 33 on page 242.

Reading and Writing Multiple Tape Volumes

When working with multiple volumes of tape on AIX or on MVS without MVS tape management, use tape calls to open and close each tape as it is required. A high-level view of the sequence of tape calls used to do this follows:

1. Allocate a tape device

   ctpalloc (****tlu*, *node*, *maxdata*, *allocparm*, *utility*)**

2. Process as many tapes as required

   a. Process the first tape

      ctpopen (*tlu*, *volser*, *mode*)
      ... process tape as you like ... **ctpclose (*tlu*, *option*, *ist*)**

   b. Process a second tape

      ctpopen (*tlu*, *volser*, *mode*)

      ... process tape as you like ...

      ctpclose (*tlu*, *option*)

   c. Process a third tape

      ctpopen (*tlu*, *volser*, *mode*)

      ... process tape as you like ...

      ctpclose (*tlu*, *option*)

   d. Repeat for each tape

3. Unallocate the device

   ctpunall (*tlu*)

Figure 33. Sequence of Calls for Reading and Writing Multiple Tape Volumes

Compiling and Linking CLIO/S Tape Interface Programs

On MVS/ESA

The FCFSAMPT JCL, located in **SYS1.SFCFSAMP(FCFSAMPT)**, compiles and links the sample C code which is provided with CLIO/S tape interface. You can modify this JCL to compile and link your own CLIO/S tape interface programs.

On the RS/6000

The makefile **/usr/lpp/fcf/samplet/Makefile** can be used to compile and link the sample CLIO/S tape interface programs that are provided with CLIO/S. You can modify it to compile and link your own CLIO/S tape interface applications.

Using the FCFTTEST Sample Program

FCFTTEST is a CLIO/S sample program that reads C-like commands from standard input and converts them to function calls to the CLIO/S Tape Interface. On AIX, it is located in the file **/usr/lpp/fcf/samplet/fcfttest.c**. On MVS, it is in the data set **SYS1.SFCFSAMP(FCFTTEST)**. Compile it as you would any other CLIO/S Tape Interface program. On MVS use the sample JCL in **SYS1.SFCFSAMP(FCFSAMPT)**; on AIX use **/usr/lpp/fcf/samplet/Makefile**.

To use FCFTTEST, pass subcommands to the program. Pass the name of an output file as the first line of input, in response to the prompt. After the output file name, you can specify blank or null lines, comments, or subcommands. Comment lines, starting with the **#** character, are allowed as any line except the first.

The subcommands for FCFTTEST are read from standard input; you can either type them manually or read them from a file or data set. The output is written to the file specified as the first line of input. Enter the subcommands in the case shown. Do not use the space character for spacing within the subcommand's argument string; it is interpreted literally.

To exit from the FCFTTEST program on AIX, use **<ctrl-d>**.

The following FCFTTEST subcommands provide a command-line interface with function equivalent to the CLIO/S Tape Interface subroutines:

tpalloc(*tlu***,***node***,***maxdata***,***allocparm***,***utility***)**

tlunall(*tlu***)**

tpopen(*tlu***,***volser***,***mode***)**

tpclose(*tlu***,***option***)**

tpread(*tlu***,***filename***,***length***)**

tpwrite(*tlu***,***filename***,***length***)**

tpskf(*tlu***,***count***)**

tpskr(*tlu***,***count***)**

tpweof(*tlu***,***count***)**

tprew(*tlu***)**

tpwrto(*tlu***,***node***,***message***,***replycode***)**

The following FCFTTEST subcommands provide additional examples. Before using these subcommands, use the **tpalloc** subcommand to allocate a tape device.

WRITEFILE(*tlu***,***filename***)**

This subcommand writes the specified file to a tape with a series of **tpwrite** subcommands.

MWRITE(*tlu***,***ignored***,***length***,***count***)**

This subcommand writes the specified number of records to the tape from a buffer coded in the **fcfttest.c** program. The code implementing this subcommand demonstrates using a series of calls to the **ctpwrite** subroutine.

MREAD(*tlu***,***ignored***,***length***,***count***)**

This subcommand reads the specified number of records from the tape into a buffer coded in the **fcfttest.c** program. The code implementing this subcommand demonstrates using a series of calls to the **ctpread** subroutine.

READ&VERIFY(*tlu***,***filename***,***record***)**

This subcommand reads a record from a tape and compares it to the corresponding record in the specified file.

Arguments to the subcommands are:

filename Specifies the name of the file containing the data to read or written. For **tpwrite**, this is the input file whose records are written to tape. For **tpread**, this is the output file created from records on the tape.

You can specify *filename* as the character string **none**, specifying that records are read into or written from an internal buffer in the **fcfttest.c** program.

ignored This argument is ignored.

message Specifies the text of the message to the operator. The message may include imbedded space characters.

record Specifies the number of the record in the file or on the tape. The first record is record number 1.

tlu Is one of the character strings **tplu1**, **tplu2**, or **tplu3**. This lets you work with up to three logical tape units at a time.

The **tpwrto** subcommand also accepts a *tlu* value of –**1**. See "ctpwrto or tpwrto—Writing to the Operator" on page 237 for more information.

Other arguments correspond to identically-named parameters passed to the CLIO/S Tape Interface subroutines.

The following example demonstrates using FCFTTEST on AIX to read and write to a tape device on a remote host. Invoke FCFTTEST with this command:

```
fcfttest < tapecmds
```

The subcommands in the file **tapecmds** on the local host are shown in Figure 34 on page 245. Line numbers have been added for reference in this example.

```
1.  /tmp/ttest.log
2.  # Write log to file /tmp/ttest.log
3.
4.  tpalloc(tplu1,riscnode1,8192,/dev/rmt0)
5.
6.  # Open tape for write
7.  tpopen(tplu1,DUMMY,3)
8.
9.  # Write a file to tape
10. WRITEFILE(tplu1,data1)
11.
12. # Close tape
13. tpclose(tplu1,0)
14.
15. # Open for read
16. tpopen(tplu1,DUMMY,1)
17.
18. # Read two first records of first file
19. READ&VERIFY(tplu1,data1,1)
20. READ&VERIFY(tplu1,data1,2)
21.
22. # Rewind tape
23. tprew(tplu1)
24.
25. # Close & unallocate the tape
26. tpclose(tplu1,0)
27. tpunall(tplu1)
```

Figure 34. Sample FCFTTEST Subcommands

Let's look at the subcommands in detail:

Line 1, `/tmp/ttest.log`
: Directs the results of FCFTTEST into file /**tmp**/**ttest.log**. This file contains a record of the subcommands issued and the result of each subcommand. If the CLIODEB environment variable is set, debug messages are written to standard output.

Line 4, `tpalloc(tplu1,riscnode1,8192,/dev/rmt0)`
: Allocates the tape device /**dev**/**rmt0** on host **riscnode1**, specifying a maximum record length of 8192. This program uses only one tape device, specified as **tplu1**.

  In this example, there is no data to pass to the AIX tape exit. Therefore, the last argument to **tpalloc** is omitted.

Line 7, `tpopen(tplu1,DUMMY,3)`
: Opens the tape volume “DUMMY” for update.

Line 10, `WRITEFILE(tplu1,data1)`
: Writes the contents of the file **data1** to the tape. Each line in the input file, delimited by the newline character, is written as a separate record.

Line 13, `tpclose(tplu1,0)`
: Closes the tape.

Line 16, `tpopen(tplu1,DUMMY,1)`
: Reopens the same volume for reading.

Lines 19, `READ&VERIFY(tplu1,data1,1)`
: Reads record 1 from the tape and compares it to line 1 of **data1**. Line 20 does the same with the next record.

When processing is complete, the program rewinds the tape, closes the tape volume, and unallocates the device.

Using TPWRTO Support On AIX

Using the TP Utility for Logging Messages

tp is an interactive shell program for maintaining the message log. It is provided as a sample of how you can work with messages in the AIX environment. It's located in **/usr/lpp/fcf/samplet/tp**.

tp reads the logfile (**tapemsg.log**) created by the real time logger, and allows the user to list the log and reply to pending messages. When starting the real time logger, **tp** will prompt for a device name to print messages. If you want them to appear in a different window than the one where **tp** is running, open a new window; use the **ps** command in that window to get the name of the pseudo terminal; enter this when starting the logger. For example:

```
tapemsg> s
Device to print messages (this window): /dev/pts/7
```

Note: The logging function must be started and stopped explicitly. It is not done automatically when you start and stop **tp**.

Specifying a Message Directory with the TAPEMSG Variable

A call to **tp** will normally write the messages to disk in default directory /**tmp** or as specified in the environment variable TAPEMSG. The messages can then be obtained by the utility program **tp**. If you are using TAPEMSG, add it to your **.login** or **.kshrc** file.

By specifying a NFS filesystem you can run one **tp** program that log messages for all hosts. For example, you can log messages on the NFS filesystem /**nfs** by putting the following line in your in **.login**:

```
setenv TAPEMSG /nfs/myhomedir/messages
```

or this line in your **.kshrc**:

```
export TAPEMSG=/nfs/myhomedir/messages
```

Determining Which Node Receives the Message

The value of the *tlu* parameter determines whether the *node* parameter is used. If *tlu* value is negative, the node parameter is used. If *tlu* value is valid the node associated with this *tlu* is used. The message will first be sent to that node, and then written to the message directory. For example:

```
ist = ctpwrto(tlu1,NULL,"Operator message",0);
```

The message will be sent to node on which `tlu1` is allocated and then written to the message directory. Node parameter is ignored.

```
ist = ctpwrto(-1,"afrodite","Operator message",0);
```

The message will be sent to node ***afrodite*** and written to the message directory. No previous **ctpalloc** is required.

```
ist = ctpwrto(-1,NULL,"Operator message",0);
```

The message will be written to the message directory by the local node.

Writing Directly to Devices

It is possible to write directly to a device instead of a message directory. The name of device must then be specified in the TAPEMSG variable. The result of writing directly to a device is not guaranteed. Do not use the reply option when writing to terminals running interactive shells. By using the **tp** program instead, you can redirect the messages to any printable device and they will also be logged.

Example 1:

```
setenv TAPEMSG /dev/console

ist = ctpwrto(-1,"afrodite","Operator message",0);
```

This call will result in "Operator message" appearing on the console on the node ***afrodite***.

Example 2:

```
setenv TAPEMSG /dev/pts/3

ist = ctpwrto(-1,"afrodite","Operator message",0);
```

This call will result in "Operator message" appearing on the pseudo terminal (xterm window) connected to pts/3 on node ***afrodite***.

Note: Writing to a remote device is performed by another process set up with the default environment. The value of TAPEMSG must therefore be set in the **.login** or **.kshrc** files.

Special Considerations for ANSI Standard Label (AL) Tapes

Creating an ANSI Standard Label tape (AL tape) on MVS has these steps:

1. Preformatting the tape
2. Writing the data set name in the header file
3. Writing data to the tape

You can't create AL tapes on AIX.

The following sections briefly explain how you create an AL tape.

Preformatting the AL Tape

You must preformat an AL tape with the MVS program IEHINITT. This sample JCL shows how to start IEHINITT:

```
//ALINITT JOB (USER INFO),'USER NAME',TIME=1,MSGCLASS=4,
//    NOTIFY=&SYSUID
//P1       EXEC PGM=IEHINITT
//SYSPRINT DD  SYSOUT=*
//TAPE     DD  UNIT=(3480,1,DEFER)
//SYSIN    DD  *
TAPE       INITT  SER=CLIOA,LABTYPE=AL
```

Writing the Data Set Name in the Tape Label Header File

After the tape is preformatted, you run the IEBGENER program to write the data set name in the second of the three header files in the label. This JCL shows how to submit that job:

```
//ALGENER  JOB (USER INFO),'USERNAME',MSGCLASS=4,
//    NOTIFY=&SYSUID
//STEP1    EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DUMMY,DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//SYSUT2   DD DSN=DUMMY,UNIT=(3480,,DEFER),LABEL=(,AL),
//            DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//SYSIN    DD DUMMY
```

Performing Tape I/O

After the label is written, you can write data to the tape using either the tape API subroutines or the UNIX tape pseudo device driver interface. An AL (or SL) tape rewinds but does not unload when the UNIX command is complete; the tape is unloaded when you issue **cltunall**.

Part 5. Appendixes

© Copyright IBM Corp. 1993, 1995, 1996

Appendix A. User Specified I/O Exits for CLFTP and CLPLINK

Use the I/O site parameter to specify a user-written Input/Output routine to be given control by CLIO/S as part of the data transfer process. The routine is specified as part of the CLFTP command and the CLPLINK command with the Site subcommand and a *io_name* parameter. The files, **fcffcltp.c, fcffrds1.c**, and **fcffwts1.c**, are provided as samples of user specified IO routines.

io_name is a function that is dynamically loaded to do transfer of data to devices that are not otherwise supported. *info* is information that is passed to *io_name* when it is called.

IO Parameter

Syntax

IO=(*io_name*, *info*)

Parameters

io_name	The name of the routine that performs I/O operations.
info	Information that is passed to *io_name*.

The IO Routine

The syntax of the IO routine must be:

```
io_name(int * rC, char ** iOBuf, int verbose, FCFFAlloc * allocP,
  FCFFVoidFunc msgFunc, char * msgData, int * lreclP, nt[4]
  recPos, int curInfoLen, char * curInfoP, char * recInfoP, int * truncP)
```

io_name must accept and return buffers in LENFIELDS format.

The parameters to be used with **IO** are:

© Copyright IBM Corp. 1993, 1995, 1996

```
  Rc  Input: Function call type
             FCFFNormal - Normal input or output.
                 When inputting from a file:
                   io_name should set *iOBuf to the address of the
                   next record.
                 When outputting to a file:
                   When io_name is called, *iOBuf contains the
                   address of the next record to be written to file.
             FCFFFirstTime - First time io_name is called. *iOBuf
                 contains the string entered in the info field of the
                 IO=(io_name, info) specification of the site command.
             FCFFLastTime - Last time io_name is called. *iOBuf is NULL.
       Output: Return code
             FCFFOk - The IO operations completed successfully.
             FCFFNoRecov - Everything went ok, but this is the first
                 call to io_name, and io_name is telling clplink
                 that it does not support recovery.

(first time, info)
     iOBuf    - If *rC == FCFFFirstTime, the info string associated with
                     the input/output routine in the site command.
                Else if *rC == FCFFLastTime, *iOBuf = NULL
                Else if we are inputting from the file
                  *iOBuf is not set on entry to io_name. io_name sets
                  *iOBuf to point to the next record input from the file
                  on exit.
                Else we are outputting to the file:
                  When io_name is called, *iOBuf points to the next
                  record that should be written to the file.

     lreclP   - If *rC == FCFFFirstTime, io_name should set *lreclP to
                     the logical record length to be used for binary
                     files. When writing to a file, clftp and clplink
                     return records of the length specified by io_name
                     in *lreclP.
                Else
                     lreclP is ignored.
     truncP     When writing to a file, if a record is truncated,
                * trunc should be set to 1.  Otherwise, it should be
                set to 0.
```

IO Parameters for Use with Recovery

If *io_name* does not support recovery, the following parameters can be ignored. Simply return the *FCFFNoRecov* value in **rC* on the first call to *io_name*.

Recovery with Input Files

The following applies only to input files:

recPos *io_name* should set *recPos* to contain information needed in order to recover to the record that it is returning. When recovery is being performed, during the first call to *io_name*, CPLINK passes the same *recPos* value back to *io_name* so that *io_name* can recover to that record.

For example, if record number 500 is returned, and the file access method used by *io_name* allows *io_name* to start reading record

number 500 by doing a seek to an offset of 40000 into the file, a value of 40000 could be returned in *recPos*[0]. Then, during recovery, if record number 500 was the last one handled correctly, CLPLINK would call *io_name* with a value of *recPos*[0] == 40000 on its first call. This signals to *io_name* that it should seek to offset 40000 and start reading the next record from there.

If *recPos* is set to –1, –1, –1, –1, on input no recovery is being done.

If *recPos* is set to –1, –1, –1, –1, on output from *io_name* a record cannot be recovered to.

Recovery with Output Files

The following applies only to output files:

recPos *io_name* should set *recPos* to contain information needed to recover to the record that it is writing. When recovery is being performed, during the first call to *io_name*, CLPLINK passes the same *recPos* value back to *io_name* so that *io_name* can recover to that record.

For example, if record number 500 is written, and the file access method used by *io_name* allows *io_name* to start writing record number 500 by doing a seek to an offset of 40000 in the file, a value of 40000 could be returned in *recPos*[0]. Then, during recovery, if record number 500 was the last one handled correctly, CLPLINK would call *io_name* with a value of *recPos*[0] == 40000 on its first call. This signals to *io_name* that it should seek to offset 40000 and start writing the next record from there.

If *recPos* is set to –1, –1, –1, –1, on input no recovery is being done.

If *recPos* is set to –1, –1, –1, –1, on output from *io_name* a record cannot be recovered to.

When writing, buffered IO sometimes makes it impossible to return a *recPos* value for every record. It is also sometimes impossible to return a value until several calls after the record is passed to *io_name*. The *curInfoLen*, *curInfoP*, and *recInfoP* parameters are provided to handle these situations. Refer to “Discussion of Recovery with Output Files and User I/O Routines” on page 255.

curInfoLen The length of the data area pointed to by *curInfoP* and *recInfoP*. *curInfoLen* is the same on every call to *io_name*.

curInfoP Information associated with the current record. The length of the information is specified by *curInfoLen*.

This information is supplied in order to support buffered output to a file. When output to a file is buffered, it may be possible to recover only to the first record of each block that is written. Thus, the recovery information associated with the first record in a block may not be available until the entire block is written. This means that the recovery information can't be returned to CLPLINK at the time *io_name* is called for a record in a block. It must be returned several records later when those records have filled the block and been written.

The way that CLPLINK handles this situation is through the *curInfoLen*, *recPos*, and *recInfoP* fields. The following rules should be followed for using these values correctly.

1. When a record is safely written, and recovery information is available for that record, the recovery information should be returned in *recPos*. This information can be returned even if the recovery information is for a record that was given to *io_name* several calls before the current one.
2. The recovery information that is returned in *recPos* is associated with some record. The *curInfoP* information that was passed into *io_name* with that record is also associated with it. When the *recPos* information associated with a record is returned, the *curInfoP* information associated with the same record must also be returned in the *recInfoP* field. A **memcpy()** call can be used to copy the information in. If the *recPos* information is for the record that is currently being passed into *io_name*, a call will take care of things.
3. If output buffering is being done, the *curInfoP* information must be saved by *io_name* until it can be returned. Local storage should be allocated and the data copied in (for example **memcpy(***locStorageP, curInfoP, curInfoLen***)**) so that it can be returned later (for example **memcpy(***recInfoP, locStorageP, curInfoLen***)**).

recInfoP Information that was originally passed into *io_name* in the *curInfoP* field and is associated with a record that can be recovered to.

Notes

Verbose 0 - Verbose messages are not being printed.

allocP Pointer to information about the file. The contents of the **allocP** structure are described in **fcffile.h**.

msgFunc A function that prints messages. It is important to use **msgFunc** rather than another function such as **fprintf** because **msgFunc** writes the message to the correct file. The parameters for **msgFunc** are similar to **fprintf**. They are:

```
msgFunc(char * msgData, char * formatString, <parm1 <,parm2 ...>>)
```

msgData The *msgData* parameter passed to *io_name*

formatString The **fprintf** style format string for the message. Imbedded % specifications such as %s and %d can be used.

parm1, parm2, ... Parameters that are used with the % specifications of the formatString.

msgData Data that is passed to **msgFunc** as the first parameter.

All records pointed to by *iOBuf* must have the following format.

4 bytes Length of record	Record contents	4 bytes Length of record

When *io_name* is called, the length fields that are passed may not be on 4 byte boundaries. *io_name* can not assume that the record lengths can be read using a pointer to an integer, since pointers to integers will work only if the length is stored on a 4 byte boundary.

Discussion of Recovery with Output Files and User I/O Routines

Returning recovery information from an **io_name** routine that does blocking of records presents a major challenge. To understand the difficulties, consider the following file made up of 17 records contained in three blocks.

Record 1	Record 2	Record 3	Record 4	Record 5	Record 6	

Record 7	Record 8	Record 9	Record 10	Record 11	Record 12	

Record 13	Record 14	Record 15	Record 16	Record 17	

The records of this file are passed to **io_name** for writing to disk. When Record 1 is passed to **io_name**, it places the record in a block in memory, but does not write the record to disk.

Block in Memory

Record 1	

It is impossible for the **io_name** to return recovery information for Record 1 now because the record has not yet been written to disk.

When Records 2 through 5 are passed, they also are stored in the block in memory and can not be stored out to disk. No recovery information can be returned for any of these records.

Block in Memory

Record 1	Record 2	Record 3	Record 4	Record 5	

Only when Record 6 is passed to **io_name** can it store the records to disk and get recovery information.

Record 1	Record 2	Record 3	Record 4	Record 5	Record 6	

The recovery information that is available now is the location of the beginning of the block on disk. This is the location on disk of the beginning of Record 1. An example of a file access method that provides this kind of information is the BSAM macros on MVS. The NOTE macro returns the position of the most recently written block. Once *io_name* has the position of the record it passes the record back to CLPLINK.

Our next block of records, Records 7 though 12, are written to disk when Record 12 is passed to **io_name**.

Record 7	Record 8	Record 9	Record 10	Record 11	Record 12	

The file access method returns the position on disk of Record 7, which **io_name** then passes back to CLPLINK.

When Records 13 through 15 are passed to **io_name** they are blocked in memory.

Block in Memory

Record 13	Record 14	Record 15				

At this point we stop the transfer so that it can be recovered later with the CLPLINK recovery option.

At recovery time, CLPLINK passes to **io_name** the recovery information for the last record that **io_name** said could be recovered to. In this example, that is Record 7. The recovery information is the position of Record 7 on disk. **io_name** uses this information to reposition the file to start writing at Record 7. After this, CLPLINK passes records to **io_name** starting with Record 7. **io_name** blocks Records 7 through 12 and writes them to the file at the same place they were written the previous time.

Record 7	Record 8	Record 9	Record 10	Record 11	Record 12	

If we were to try to start recovering at any of Records 8 through 12, we would have to start writing in the middle of a block. We might be able to do this if we could open the file for input and output and read in the first part of the block before changing the last part of it, but this is unnecessarily complicated and may cause problems if opening a file in both read and write mode at the same time is not supported. By recovering to the first record of the block we can reopen the file in write mode as it was opened when we started the transfer.

After writing the block for Records 7 through 12, Records 13 though 17 are passed to **io_name** and written to disk.

Record 13	Record 14	Record 15	Record 16	Record 17		

The file can be closed and the transfer ended.

Now we will look at how the *recPos, curInfoLen, curInfoP,* and *recInfoP* variables would be used in this example.

When Record 1 is passed to **io_name**, *curInfoLen* and *curInfoP* contain information that is associated with Record 1. This information must be saved by **io_name** until we know the recovery information that allows us to recover to Record number 1. A memcpy (*saveAreaP, curInfoP, curInfoLen*) can be used to save the information. **io_name** will save Record 1 in a block in memory and then set *recPos*[0] through *recPos*[3] to -1,-1,-1,-1 to indicate that no recovery information is available. Since no recovery information is available, CLPLINK ignores the value pointed to by *recInfoP*.

Records 2 through 5 are treated like Record 1 except that there is no need to save the *curInfoLen* and *curInfoP* information for these records since **io_name** can not recover to them. When Record 6 is handled and the block is written to disk, **io_name** gets the recovery information from the file access method.

Record 1	Record 2	Record 3	Record 4	Record 5	Record 6	

io_name returns the recovery information in *recPos*[0] through *recPos*[3].

Since Record 1 is the record that can be recovered to, the *curInfoLen* and *curInfoP* information that was saved by **io_name** when Record 1 was handled, is returned in the area pointed to by *recInfoP*. This tells CLPLINK that if it recovers to the point specified in *recPos*, it should start writing at Record 1.

Record 7 is treated like Record 1. Its *curInfoLen* and *curInfoP* information must be saved, and *recPos* is be set to -1,-1,-1,-1 before returning. Records 8 through 11 are treated like Records 2 through 5. When Record 12 is handled, the block is written to disk. **io_name** gets the recovery information from the file access method.

Record 7	Record 8	Record 9	Record 10	Record 11	Record 12	

fcffio_ returns the recovery information in *recPos*[0] through *recPos*[3].

Since Record 7 is the record that can be recovered to, the *curInfoLen* and *curInfoP* information that was saved by **io_name** when Record 7 was handled is returned in the area pointed to by *recInfoP*. This tells CLPLINK that if it recovers to the point specified in *recPos*, it should start writing at Record 7.

Records 13 through 17 are treated like Records 7 through 12. Then the file is closed.

Linking the IO Routine on AIX

On AIX, the IO routine must be linked with the **-e** flag to identify the *io_name* entry point. The Makefile provided in the CLIO/S sample directory contains examples of how this is done. The section of the Makefile for IO routines is shown below.

```
#       Compile, link, and copy the fcffrds1 program.
        $(CC) -c $(COMPILE_FLGS) -o fcffrds1.o fcffrds1.c
        $(CC) -o fcffrds1 fcffrds1.o $(LNK_FLGS) -e fcffrds1
        mv fcffrds1 $(BIN)

#       Compile, link, and copy the fcffwts1 program.
        $(CC) -c $(COMPILE_FLGS) -o fcffwts1.o fcffwts1.c
        $(CC) -o fcffwts1 fcffwts1.o $(LNK_FLGS) -e fcffwts1
        mv fcffwts1 $(BIN)
```

Loading the IO Routine on AIX

The IO routine is loaded when the IO SITE parameter is entered for CLFTP or CLPLINK. The IO routine can be specified in the same way that any file is specified to CLFTP or CLPLINK, that is, it can be specified with its full path name or with a name relative to the current directory. An example of specifying with a full path name is:

```
clftp host1
clftp> site io=(/usr/lpp/fcf/bin/fcffrds1)
```

An example of specifying with a name relative to the current directory is:

```
clftp host1
clftp> lcd /usr/lpp/fcf/bin
clftp> locsite io=(fcffrds1)
```

In addition the LIBPATH environment variable can be set to point to a directory where the IO routine can be found. When this is done, the file name can be used.

```
setenv LIBPATH /usr/lpp/fcf/bin:${LIBPATH}
clftp host1
clftp> locsite io=(fcffrds1)
```

If LIBPATH must be set on the server machine so that IO can be used with the SITE command, the FCFSHELL shell script should be updated to set the LIBPATH variable before executing the FCFFSERV program (the server side of the CLFTP program) or the CLPLINK program (the server side of CLPLINK.

Linking the IO Routine on MVS

On MVS, the IO routine must be linked with an ENTRY statement to identify the *io_name* entry point. The FCFSAMP JCL in the CLIO/S SFCFSAMP partitioned dataset contains examples of how this is done. The section of the JCL for IO routines is shown below.

```
//FCFFRDS1 EXEC FCFSMPCL,
//           MBR=FCFFRDS1
//CLKED.SYSIN DD *
   ENTRY FCFFRDS1
   NAME FCFFRDS1(R)
//FCFFWTS1 EXEC FCFSMPCL,
//           MBR=FCFFWTS1
//CLKED.SYSIN DD *
   ENTRY FCFFWTS1
   NAME FCFFWTS1(R)
```

Loading the IO Routine on MVS

On MVS, the IO routine must be a member of a partitioned dataset in the JOBLIB or STEPLIB.

Appendix B. User Specified Data Translation for CLFTP and CLPLINK

Use the CONV site parameter for the CLFTP and CLPLINK commands to specify a user-written data conversion routine to be given control by CLIO/S as part of the data transfer process. The user-written routine can change the content and/or length of records. It can also add or delete records. The routine is specified as part of the Site subcommand and a *conv_name* parameter. The file, **fcffcnv1.c**, is provided as a sample of a user specified data translation routine.

conv_name is a function that is be dynamically loaded to do translation of data. *data1* and *data2* are strings that are passed to *conv_name* the first time that it is called.

CONV Parameter

Syntax

CONV=(*conv_name*,*data1*,*data2*)

Parameters

conv_name The name of the routine that will translate records.

data1 **and** *data2*
Strings that are passed to *conv_name*. The first time *conv_name* is called, these strings are used to pass information from the CLFTP or CLPLINK user to the *conv_name* routine.

The *conv_name* Routine

The syntax of the conversion routine must be:

```
conv_name(int * code, char **in_buf, char **out_buf,
int verbose, FCFFAlloc * allocP, FCFFVoidFunc msgFunc, char * msgData)
```

The parameters to *conv_name* are:

```
Input:
  FCFFNormal - Normal conversion. *inBuf contains the address
      of the buffer to be translated.
  FCFFFirstTime - First time conv_name is called. *inBuf
      contains the first data string associated with the
      conversion routine by the site command (data1).
      *outBuf contains the second data string associated
      with the conversion routine by the site command (data2).
  FCFFLastTime - Last time conv_name is called. *inBuf
      and *outBuf are NULL.
  FCFFInsert - conv_name is being called so it can return
      a buffer that is to be inserted. *inBuf is NULL.
```

© Copyright IBM Corp. 1993, 1995, 1996

```
                Output: Return code
                  FCFFOk - Record translated. No error.
                  FCFFErrConv - An error occurred in the conversion.
                  FCFFInsertReq - The record was translated. After the
                      record is handled by the calling program, the
                      conv_name routine should be called again with input
                      at Code=FCFFInsert so that it can return another record.
                  FCFFErrProto - A protocol error has occurred.
                  FCFFErrProg - An internal program error has occurred.
                  FCFFNoRecov - When *rC == FCFFFirstTime on entry,
                      FCFFNoRecov can be returned to indicate that this
                      conversion routine keeps state information between
                      records and thus can not be used at a server when
                      recovery is being done. The routine can be used at
                      the client side of the transfer.

in_buf   - If *rC == FCFFFirstTime, the data1 string from the CONV
                parameter of the site subcommand.
           Else if *rC == FCFFLastTime, *inBuf == NULL
           Else if *rC == FCFFInsert, *inBuf == NULL
           Else the buffer to translate.

outBuf   - Input:
             If *rC == FCFFFirstTime, the data2 string from the CONV
                 parameter of the site subcommand.
             Else *outBuf == NULL.
           Output:
             If *rC == FCFFFirstTime on input, *outBuf is ignored on
                 output.
             Else if *rC == FCFFOk or FCFFInsertReq on ouput,
                 *outBuf should be set to point to the converted record.
                 If *outBuf == *inBuf, the conversion must not
                     change the length of the record.
                 If *outBuf == NULL, the record has been deleted
                     rather than translated.
                 If *rC == FCFFLastTime on input, conv_name can return
                     an output record. The calling routine continues to
                     call conv_name_ with *rC == FCFFLastTime as long as
                     conv_name returns with *outBuf != NULL and
                     *rC == FCFFInsertReq.
             else *outBuf == NULL.
```

Notes

outBuf should always be allocated to be at least 1 byte longer than the largest record it contains. CLFTP and CLPLINK sometimes need to use the extra byte.

verbose 0 - Verbose messages are not being printed.
1 - Verbose messages are being printed.

allocP Pointer to information about the file. The contents of the *allocP* structure are described in **fcffile.h**.

msgFunc A function that can be called to print messages. It is important to use **msgFunc** rather than another function such as **fprintf** because **msgFunc** writes the message to the correct file. The parameters for **msgFunc** are similar to **fprintf**. The syntax of **msgFunc** is:

```
msgFunc(char * msgData, char * formatString, <parm1 <,parm2 ...>>)
```

Where:

msgData The *msgData* parameter passed to **conv_name**.

formatString The **fprintf** style format string for the message. Imbedded % specifications such as %s and %d can be used.

parm1, parm2, ... Parameters that are used with the % specifications of the formatString.

msgData Data that is passed to **msgFunc** as the first parameter.

Record Boundaries for the CONV Routine

Record boundaries for *in_buf* are determined as follows:

- On MVS:
  - By the physical records on the disk.
- On AIX:
  - ASCII files - By new line characters.
  - Binary files - By the MAXBLKSIZE site parameter.
  - RDW files - By the length fields stored in the records.
  - LENFIELDS files - By the length fields stored in the records.

When *conv_name* is called, the length fields that are passed may not be on 4 byte boundaries. *conv_name* can not assume that the record lengths can be read using a pointer to an integer, since pointers to integers will work only if the length is stored on a 4 byte boundary.

Linking the CONV Routine on AIX

On AIX, the CONV routine must be linked with the –**e** flag to identify the *conv_name* entry point. The Makefile provided in the CLIO/S sample directory contains an example of how this is done. The section of the Makefile for the CONV routine is shown below.

```
#       Compile, link, and copy the fcffcnv1 program.
        $(CC) -c $(COMPILE_FLGS) -o fcffae.o fcffae.c
        $(CC) -c $(COMPILE_FLGS) -o fcffcnv1.o fcffcnv1.c
        $(CC) -o fcffcnv1 fcffcnv1.o fcffae.o $(LNK_FLGS) -e FCFF
```

Loading the CONV Routine on AIX

The CONV routine is loaded when the CONV SITE parameter is entered for CLFTP or CLPLINK. The CONV routine can be specified in the same way that any file is specified to CLFTP or CLPLINK, that is, it can be specified with its full path name or with a name relative to the current directory. An example of specifying with a full path name is:

```
clftp host1
clftp> site conv=(/usr/lpp/fcf/bin/fcffcnv1)
```

An example of specifying with a name relative to the current directory is:

```
clftp host1
clftp> lcd /usr/lpp/fcf/bin
clftp> locsite conv=(fcffcnv1)
```

In addition, the LIBPATH environment variable can be set to point to a directory where the CONV routine can be found. When this is done, the file name can be used.

```
setenv LIBPATH /usr/lpp/fcf/bin:${LIBPATH}
clftp host1
clftp> locsite conv=(fcffcnv1)
```

If LIBPATH must be set on the server machine so that CONV can be used with the SITE command, the FCFSHELL shell script should be updated to set the LIBPATH variable before executing the FCFFSERV program (the server side of the CLFTP program) or the CLPLINK program (the server side of CLPLINK.

Linking the CONV Routine on MVS

On MVS, the CONV routine must be linked with an ENTRY statement to identify the *conv_name* entry point. The FCFSAMP JCL in the CLIO/S SFCFSAMP partitioned dataset contains an example of how this is done. The section of the JCL for the CONV routines is shown below.

```
//FCFFCNV1 EXEC FCFSMPCL,
//           MBR=FCFFCNV1
//CLKED.SYSIN DD *
   INCLUDE SYSLIB(FCFFAE)
   ENTRY FCFFCNV1
   NAME FCFFCVN1(R)
```

Loading the CONV Routine on MVS

On MVS, the CONV routine must be a member of a partitioned dataset in the JOBLIB or STEPLIB.

Appendix C. Support Programs and Sample JCL

This appendix describes some of the support programs and sample JCL that are provided with CLIO/S.

The sample JCL described in this appendix should be modified by the installer to provide specific information as required by the installation. On MVS, this information may include, but is not limited to, job card information, device types, and data set names.

Note: If you use a security product other than RACF, you need to make the appropriate changes to the default JCL generation programs.

On AIX, Makefiles are provided to compile and link the samples in **/usr/lpp/fcf/sample** and **/usr/lpp/fcf/samplet**. Modify these Makefiles if your installation has special requirements, for example, different compiler names and library locations.

On AIX, these sample programs are located in the directory **/usr/lpp/fcf/bin**. On MVS, they are in the partitioned data set **SYS1.SFCFSAMP**.

If your installation has started more than one CLIO/S server (PARSRV) on the host to which the tape drive is attached, set the CLIOPSPT environment variable to the appropriate port number. On AIX, set the CLIOPSPT and CLIOPSID environment variables before invoking the program. On MVS, use *userid***.CLIOENV** or the CLIOENV DD statement in the JCL that starts the job.

If PARSRV is registered in **/etc/services** on AIX or in **TCPIP.profile.TCPIP** on MVS, you must set the CLIOPSID environment variable to the name corresponding to the server/port combination you want to use.

CLIO/S Support Programs

The support programs described here assist you with various CLIO/S tasks. On MVS, sample JCL is provided for each program. The JCL should be modified to match your installation requirements and to pass your parameters to these programs. On AIX, you can use the programs without modification. The following table lists the name of the program on both MVS and AIX, and the tasks each program helps you perform.

Table 10 (Page 1 of 2). CLIO/S Support Programs

This program on ...			
MVS	**AIX**	**Helps you with this task...**	**Page**
FCFCAN	<ctl-C>	Cancelling CLIO/S jobs	264
FCFIDUM	fcfidum	Verifying that CLIO/S can communicate between master and servant programs	265
FCFFCLPR, FCFFCLPW	clplink	Starting the CLPLINK program	266
FCFFPIPR, FCFFPIPW	fcffpipr, fcffpipw	Testing the CLIO/S Cross System Pipe Link	272
FCFPARDE	pardeb	Switching server debug flags	268

© Copyright IBM Corp. 1993, 1995, 1996

Table 10 (Page 2 of 2). CLIO/S Support Programs

This program on ...			
MVS	**AIX**	**Helps you with this task...**	**Page**
FCFPARIN	parinfo	Displaying server information	268
FCFPARPI	parping	Tests whether PARSRV is running	269
FCFPARSR	parsrv	Starting a DPI server	269
FCFPARST	parstop	Stopping a DPI server	271
FCFVARY	n/a	Varying on/off subchannels on MVS, starting subchannels on the workstation FCFVARY is the master program that you submit on MVS; it starts the servant program, **varys**, on AIX	273
n/a	online	Starting subchannels on the workstation	275

Cancelling a CLIO/S Job—FCFCAN, FCFCANC

If a CLIO/S master program on AIX is killed with <**ctrl-c**>, the associated MVS servants are automatically cancelled using the program specified in **FCFDEFS**. The two programs for cancelling CLIO/S servant jobs are:

- The FCFCAN authorized program
- The FCFCANC REXX EXEC

The person who installs CLIO/S selects one method as the installation default and modifies the C*A*N section of **FCFDEFS** accordingly. The CLIO/S Program Directory explains how to install these programs and modify **FCFDEFS** to select the installation default. You can override the installation default by modifying the **FCFDEFS** used when you start PARSRV.

The FCFCAN Authorized Program

FCFCAN is an authorized program that cancels CLIO/S jobs. FCFCAN cancels CLIO/S jobs with any job name.

FCFCAN takes a single parameter, *tnnnnn*. This is the process ID used in the CMI and DPI interfaces, and is composed of:

t Is a code indicating the type of job. Valid job types are:

2 Batch job
3 Started task
4 TSO user

nnnnn Specifies the job number.

The following JCL submits the cancel job. In this sample, the parameter is 204563, which cancels batch job (2) number 04563. This sample JCL is provided with CLIO/S in **SYS1.SFCFSAMP(FCFCAN)**.

```
//FCFCAN JOB 1,PP.NUMBER.5648.129,MSGLEVEL=1,REGION=4096K
//*****************************************************
//* FUNCTION:    CLIO/S CANCEL MVS JOB
//*****************************************************
//*
//CLIOCAN  EXEC PGM=FCFCAN,PARM='204563'
//STEPLIB  DD   DISP=SHR,DSN=SYS1.APFLOAD
//SYSPRINT DD   SYSOUT=*
```

Figure 35. JCL for Cancelling a CLIO/S Job (FCFCAN)

FCFCANC REXX EXEC

FCFCANC is a REXX EXEC that cancels CLIO/S jobs. A sample exec is provided in **SYS1.SFCFSAMP(FCFCANC)**. **FCFCANC** takes two parameters, *tnnnnn* and *userid*, where:

tnnnnn Identifies the job to be cancelled

t Is a code indicating the type of job. Valid job types are:

2 Batch job
3 Started task
4 TSO user

nnnnn Specifies the job number.

userid Specifies the user that originated the job.

Note: **FCFCANC** uses the TSO CANCEL command to cancel the job. It only works on jobs with TSO job names consisting of the user ID plus one character. If you need to cancel jobs with more generic names, you must use the **FCFCAN** authorized program.

Verifying that CLIO/S Can Communicate Between Master and Servant—FCFIDUM

The FCFIDUM program helps you verify that your CLIO/S environment is correct. If you can run the FCFIDUM program from both ends of a CLIO/S connection, then the CLIO/S environment is correct. The Installation Verification Procedure (IVP) uses the FCFIDUM program to test the CLIO/S connection MVS-to-MVS and AIX-to-AIX. To verify that the mainframe-to-workstation CLIO/S connection is correct, debug your environment by running FCFIDUM:

- On MVS, specifying an AIX host
- On AIX, specifying an MVS host

The program establishes a DPI environment on a remote host, starts a servant process, transfers data, and displays informational messages about the processing. The code dynamically determines whether it is running as a master or servant process.

FCFIDUM takes four arguments: *node*, *size*, *count*, and *report*, where:

node Specifies the remote node on which to start the servant. This argument is required.

size Specifies the size of the array to be transmitted. The default value is 1000.

count Specifies the number of times you want to transmit the array. The default value is 1.

report Specifies when to generate an interim report. The report is generated every *report* iterations of sending the array. If *count* is less than *report*, no report is generated. The default value is 100.

On AIX, this program is located in **/usr/lpp/fcf/bin/fcfidum**.

The following JCL can be used to submit the FCFIDUM master process on MVS. It is located in **SYS1.SFCFSAMP(FCFIDUM)**. Modify the sample JCL to work at your installation, and to specify the desired arguments to the FCFIDUM program.

```
//jobname  JOB (ACCT),'NAME',.....
//*
//* Start IDUM with node=rs2, size=10000,
//*                 count=10, and report=default.
//IDUM     EXEC PGM=IDUM,PARM='rs2 10000 10'
//* Modify SYS1.SFCFLOAD: name of file containing
//*                       FCFIDUM module.
//STEPLIB  DD   DISP=SHR,DSN=SYS1.SFCFLOAD
//         DD   DISP=SHR,DSN=SYS1.SEDCLINK
//*
//* These values may fetched by CLIOENV()
//*
//CLIOENV  DD   *       If omitted, userid.CLIOENV will be used
CLIODEB=8
/*
//* Define the network
//*
//PARCONF  DD   *       If omitted, userid.PARCONF will be used
rs2    1   0   kgnrs2  # RS6000, node=kgnrs2, protocol=dynamic
rs2t   1   1   kgnrs2  # RS6000, node=kgnrs2, protocol=TCPIP
rs2p   1   3   kgnrs2  # RS6000, node=kgnrs2, protocol=PCA
mvs    2   0   kgnmvs  # MVS/ESA, node=kgnmvs, protocol=dynamic
/*
//* Provide userid and password
//*
//NETRC    DD   *       If omitted, userid.NETRC   will be used
machine kgnrs2 login rs2user password secret
machine kgnmvs login mvsuser password secret
/*
```

Figure 36. JCL for Sample Master Program (FCFIDUM)

Starting the CLPLINK program—FCFFCLPR, FCFFCLPW, clplink

The CLPLINK command connects your system to a remote system using the CLIO/S Cross System Pipe Link. “clplink–Connecting To a CLIO/S Named Pipe” on page 148 describes how you use this command. To start the program on AIX, use the **clplink** command, located in **/usr/lpp/fcf/bin**. On MVS, start the program from the TSO command line or by submitting JCL.

Sample JCL for starting CLPLINK is provided in **SYS1.SFCFSAMP**. The sample JCL shows how to connect to BatchPipes/MVS because that is the more complex

environment. You may modify this JCL to read or write directly to MVS data sets or AIX programs. The following samples are provided:

FCFFCLPR Starts a CLPLINK that reads from an MVS BatchPipe.

FCFFCLPW Starts a CLPLINK that writes to an MVS BatchPipe.

You can use these programs in concert with FCFFPIPR and FCFFPIPW to test the CLIO/S Cross System Pipe Link. These programs are described in “Testing the CLIO/S Cross System Pipe Link—FCFFPIPR, FCFFPIPW, fcffpipr, fcffpipw” on page 272

```
//FCFFCLPR JOB 1,PP.NUMBER.5648.129,MSGLEVEL=1,REGION=17M
//*************************************************************
//* Call CLPLINK to read from pipe &SYSUID..TEST and to send the
//* data to another CLPLINK instance that will connect to this
//* one on port 50000.
//*************************************************************
//CLPLINK  EXEC PGM=CLPLINK,PARM='-r dd:pipe -p 50000'
//STEPLIB  DD DSN=SYS1.SFCFLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=4
//*************************************************************
//* Update the SUBSYS= parameter below to the value needed for
//* batch pipes on your system.
//*************************************************************
//PIPE     DD DSN=&SYSUID..TEST,DISP=SHR,SUBSYS=BP01,
//            DCB=(LRECL=260,RECFM=VB)
```

Figure 37. JCL for Starting CLPLINK to Read From an MVS BatchPipe (FCFFCLPR)

```
//FCFFCLPW JOB 1,PP.NUMBER.5648.129,MSGLEVEL=1,REGION=17M
//*************************************************************
//* Call CLPLINK to read from another CLPLINK instance that
//* connects to this one on port 50000. This instance of
//* CLPLINK will then write the data to batch pipe &SYSUID..TEST.
//*************************************************************
//CLPLINK  EXEC PGM=CLPLINK,PARM='-w dd:pipe -p 50000'
//STEPLIB  DD DSN=SYS1.SFCFLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=4
//*************************************************************
//* Update the SUBSYS= parameter below to the value needed for
//* batch pipes on your system.
//*************************************************************
//PIPE     DD DSN=&SYSUID..TEST,DISP=(OLD,DELETE,DELETE),
//            SPACE=(100,(100,100)),SUBSYS=BP01,
//            DCB=(LRECL=260,RECFM=VB)
```

Figure 38. JCL for Starting CLPLINK to Write to an MVS BatchPipe (FCFFCLPW)

Switching Server Debug Flags—FCFPARDE, pardeb

This program switches a debug flag in PARSRV to enable or disable the echoing to stderr of JCL produced during JCL generation. It can be run on the mainframe or on a workstation, but the argument must be the name of an MVS system. The arguments are zero or more node names. If no arguments are supplied, PARSRV on the local node is used.

On AIX, this program is located in **/usr/lpp/fcf/bin/pardeb**. For MVS, sample JCL for submitting FCFPARDE is in **SYS1.SFCFSAMP(FCFPARDE)**. This is the JCL that submit the FCFPARDE job:

```
//FCFPARDE JOB 1,PP.NUMBER.5648.129,MSGLEVEL=1,REGION=4096K
//***-------------------------------------------------
//* CLIO/S - PARALLEL SERVER DEBUG FLAG
//*-------------------------------------------------
//FCFPARDE  EXEC PGM=FCFPARDE,PARM='MVSNODE'
//STEPLIB   DD DISP=SHR,DSN=SYS1.SFCFLOAD
//          DD DISP=SHR,DSN=VSPASCAL.VSPV1R2.SAMPRUN2
//SYSPRINT  DD SYSOUT=*,DCB=BLKSIZE=3429
```

Figure 39. JCL for Switching PARSRV's Debug Flag (FCFPARDE)

Displaying Server Information—FCFPARIN, parinfo

FCFPARIN prints information about the status of PARSRV. The arguments to FCFPARIN are zero or mode node names. Here is a sample output of FCFPARIN:

```
Nload=0 Ntran=3 DEBUG=1. Started Wed 11/05/94 21:47:10. Version=210.
```

This output says that:

`Nload` Shows the number of programs started or submitted. In this case, no jobs have been started or submitted since PARSRV was started.

`Ntran` Shows the number of transactions PARSRV has handled. In this example, PARSRV has handled 3 transactions.

`DEBUG` Displays the current value of debug flag. In this case, the DEBUG flag has the value 1, or ON.

`Started Wed 11/05/94 21:47:10`
Indicates the day of the week, date, and time that PARSRV was started.

`Version` Displays the version of CLIO/S that is running. In this example, CLIO/S Version 2 Release 1.0 is running.

To run FCFPARIN on AIX, use the **parinfo** command, located in **/usr/lpp/fcf/bin**:

```
parinfo nodename
```

where *nodename* is the name of the node about which you want information.

The sample JCL, provided in **SYS1.SFCFSAMP(FCFPARIN)**, is:

```
//FCFPARIN JOB 1,PP.NUMBER.5648.129,MSGLEVEL=1,REGION=4096K
//***-------------------------------------------------
//* CLIO/S - DISPLAY PARALLEL SERVER INFORMATION
//*-------------------------------------------------
//FCFPARIN  EXEC PGM=FCFPARIN,PARM='MVSNODE'
//STEPLIB   DD DISP=SHR,DSN=SYS1.SFCFLOAD
//          DD DISP=SHR,DSN=VSPASCAL.VSPV1R2.SAMPRUN2
//SYSPRINT  DD SYSOUT=*,DCB=BLKSIZE=3429
```

Figure 40. JCL for Displaying Information about PARSRV (FCFPARIN)

Testing Whether PARSRV is Running—FCFPARPI, parping

FCFPARPI tests whether PARSRV is running on a specified host.

FCFPARPI accepts zero or mode node names as parameters. If you omit the node name, FCFPARPI tests the PARSRV server on your local host.

To run FCFPARPI on AIX, use the command **/usr/lpp/fcf/bin/parping**.

If the PARSRV server is running on the specified host, the response displays elapsed time (similar to FTP's PING command). For example, if the PARSRV server on host mvsnode is running, you receive a response similar to this:

```
$ parping mvsnode
mvsnode(0x812807e2):PING took 10.414004 ms
```

If the PARSRV server is not running on host ***riscnode1***, the response is similar to this:

```
$ parping riscnode1
riscnode1(0x81286b08):CLIO/S Parallel Server not responding
```

For MVS, the sample JCL for starting FCFPARPI, provided in **SYS1.SFCFSAMP(FCFPARPI)**, is the following:

```
//FCFPARPI JOB 1,PP.NUMBER.5648.129
//*-------------------------------------------
//*  FCFPARPI: Test whether PARSRV is running
//*-------------------------------------------
//FCFPARPI EXEC PGM=FCFPARPI,PARM='MVSNODE RISCNODE1'
//STEPLIB  DD   DISP=SHR,DSN=SYS1.SFCFLOAD
//SYSPRINT DD   SYSOUT=4,DCB=BLKSIZE=3429
//SYSTSPRT DD   SYSOUT=4,DCB=BLKSIZE=3429
```

Figure 41. JCL for Testing If PARSRV is Running (FCFPARPI)

Starting the CLIO/S Server–FCFPARSR, parsrv

PARSRV is the CLIO/S server; it starts servant jobs. For more information about using the PARSRV program, see "The CLIO/S Server—PARSRV" on page 34. "Running Multiple PARSRVs" on page 41 describes considerations for running multiple PARSRV servers on one host.

Starting PARSRV on AIX

On AIX, start PARSRV using the **parsrv** command. **parsrv** is installed in the directory /**usr**/**lpp**/**fcf**/**bin**.

The AIX **parsrv** can only submit jobs in the directories specified by the path $CLIOBIN:$CLIOSYS:$CLIOPATH. These environment variables are described on pages 37 through 39.

On AIX, normal processing messages are written to stdout; error messages are written to stderr. You can redirect stdout and stderr to a file to log the activities of the PARSRV server. For example:

```
parsrv 1> parsrv.log 2>&1
```

Starting PARSRV on MVS

You start PARSRV on MVS by submitting JCL. The JCL that starts PARSRV specifies the data sets containing the jobs that PARSRV can start.

On MVS, logging information for PARSRV is written to the STDOUT and STDERR files.

You can start PARSRV in either an MVS or a TSO address space. Normally, PARSRV runs in an MVS address space. Starting it in a TSO address space lets your REXX JCL generation program access TSO features. This can help you debug problems with the REXX program.

To start PARSRV as a started task in an MVS address space, use the sample JCL provided in **SYS1.SFCFSAMP(FCFPARSR)**. This is the way PARSRV is started by default. You may modify the JCL to start PARSRV as a batch job. The sample JCL is:

```
//*FCFPARSR PROC 1,PP.NUMBER.5648.129
//************************************************
//*         CLIO/S - START THE PARSRV SERVER
//************************************************
//FCFPARSR PROC
//*
//FCFPARSR EXEC PGM=FCFPARSR
//STEPLIB  DD   DISP=SHR,DSN=SYS1.SFCFLOAD
//*        DD   DISP=SHR,DSN=SYS1.SEDCLINK
//*        DD   DISP=SHR,DSN=SYS1.SEQALINK
//*        DD   DISP=SHR,DSN=VSPASCAL.VSPV1R2.SAMPRUN2
//SYSPRINT DD   SYSOUT=4,DCB=BLKSIZE=3429
//INSPLOG  DD   SYSOUT=4,DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//DDTERM   DD   DISP=SHR,DSN=SYS1.SFCFSAMP(FCFDDT)
//DEFAULTS DD   DISP=SHR,DSN=SYS1.SFCFSAMP(FCFDEFS)
```

Figure 42. JCL for Starting the CLIO/S server PARSRV as a started task in an MVS Address Space (FCFPARSR)

To use the debugging features of TSO for debugging PARSRV or the JCL it creates for submitting jobs, you must start PARSRV in a TSO address space, using the IKJEFT01 service, as shown in the sample JCL in **SYS1.SFCFSAMP(FCFPRTSO)**, shown here:

```
//*FCFPRTSO PROC 1,PP.NUMBER.5648.129
//*--------------------------------------------
//* Start FCFPARSR in a TSO address space.
//*--------------------------------------------
//FCFPRTSO PROC
//*
//PARSRV   EXEC PGM=IKJEFT01
//STEPLIB  DD   DISP=SHR,DSN=SYS1.SFCFLOAD
//*        DD   DISP=SHR,DSN=SYS1.SEDCLINK
//*        DD   DISP=SHR,DSN=SYS1.SEQALINK
//*        DD   DISP=SHR,DSN=VSPASCAL.VSPV1R2.SAMPRUN2
//SYSPRINT DD   SYSOUT=4,DCB=BLKSIZE=3429
//SYSTSPRT DD   SYSOUT=4,DCB=BLKSIZE=3429
//INSPLOG  DD   SYSOUT=4,DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//DDTERM   DD   DISP=SHR,DSN=SYS1.SFCFSAMP(FCFDDT)
//DEFAULTS DD   DISP=SHR,DSN=SYS1.SFCFSAMP(FCFDEFS)
//REXXJCL  DD   DISP=SHR,DSN=SYS1.SFCFSAMP
//    PEND
//P1  EXEC PARSRV
//SYSTSIN  DD   *
   CALL 'SYS1.SFCFLOAD(FCFPARSR)'
/*
```

Figure 43. JCL for Starting the CLIO/S server PARSRV in a TSO Address Space (FCFPRTSO)

Stopping a CLIO/S Server—FCFPARST, parstop

To stop a CLIO/S server, use FCFPARST or **/usr/lpp/fcf/bin/parstop**. These commands take zero or mode node names as input parameters. These are the names of systems on which you want to stop the PARSRV servers. If you don't specify a node name, **parstop** stops the PARSRV on your local system.

To stop a PARSRV server from an AIX system, use the **parstop** command, in **/usr/lpp/fcf/bin**. Specify the name of the node on which the server is running as a parameter to the command, as shown here:

```
parstop nodename
```

where *nodename* is the name of the node you are stopping.

To stop PARSRV from an MVS system, submit JCL similar to that found in **SYS1.SFCFSAMP(FCFPARST)**, shown here:

```
//FCFPARST JOB 1,PP.NUMBER.5648.129,MSGLEVEL=1,REGION=4096K
//***-------------------------------------------------
//* CLIO/S - STOP PARALLEL SERVER
//*-------------------------------------------------
//FCFPARST  EXEC PGM=FCFPARST,PARM='AIXNODE'
//*FCFPARST  EXEC PGM=FCFPARST,PARM='MVSNODE'
//STEPLIB   DD DISP=SHR,DSN=SYS1.SFCFLOAD
//          DD DISP=SHR,DSN=VSPASCAL.VSPV1R2.SAMPRUN2
//SYSPRINT  DD SYSOUT=4,DCB=BLKSIZE=3429
```

Figure 44. JCL for Stopping a DPI Server (FCFPARST)

When the server is stopped, stdout contains the following message:

```
nodeid(node hexaddress): CLIO/S parallel server stopping
```

Where *nodeid*, *node*, and *hexaddress* are variables that display your installation specific information.

Note: When CLIO/S is shutdown or closed, TCP/IP can take a few minutes to release the connection.

Testing the CLIO/S Cross System Pipe Link—FCFFPIPR, FCFFPIPW, fcffpipr, fcffpipw

The FCFFPIPR and FCFFPIPW programs let you test the CLIO/S Cross System Pipe Link. To start the program on AIX, use the **fcffpipr** and **fcffpipw** commands, located in /**usr**/**lpp**/**fcf**/**bin**. On MVS, start the programs from the TSO command line or by submitting JCL.

Sample JCL for starting FCFFPIPR and FCFFPIPW is provided in **SYS1.SFCFSAMP**. The sample JCL shows how to connect to BatchPipes/MVS because that is the more complex environment. You may modify this JCL to read or write directly to MVS data sets or AIX programs. The following samples are provided:

FCFFPIPR Starts sample program FCFFPIPR, reading from an MVS BatchPipe.

FCFFPIPW Starts sample program FCFFPIPW, writing to an MVS BatchPipe.

```
//FCFFPIPR JOB 1,PP.NUMBER.5648.129,MSGLEVEL=1,REGION=4096K
//*************************************************************
//* Call the FCFFPIPR program, which will read from batch pipe
//* &SYSUID..TEST.
//*************************************************************
//FCFFPIPR EXEC PGM=FCFFPIPR,PARM='dd:pipe'
//STEPLIB  DD DSN=SYS1.SFCFLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=4
//*************************************************************
//* Update the SUBSYS= parameter below to the value needed for
//* batch pipes on your system.
//*************************************************************
//PIPE     DD DSN=&SYSUID..TEST,DISP=SHR,SUBSYS=BP01,
//            DCB=(LRECL=260,RECFM=VB)
```

Figure 45. Testing the CLIO/S Cross System Pipe Link by reading from an MVS BatchPipe (FCFFPIPR)

```
//FCFFPIPW JOB 1,PP.NUMBER.5648.129,MSGLEVEL=1,REGION=4096K
//*************************************************************
//* Call the FCFFPIPW program, which will cause 100000 bytes to
//* be written to batch pipe &SYSUID..TEST.
//*************************************************************
//FCFFPIPW EXEC PGM=FCFFPIPW,PARM='dd:pipe 1000000'
//STEPLIB  DD DSN=SYS1.SFCFLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=4
//*************************************************************
//* Update the SUBSYS= parameter below to the value needed for
//* batch pipes on your system.
//*************************************************************
//PIPE     DD DSN=&SYSUID..TEST,DISP=(NEW,DELETE,DELETE),
//            SPACE=(100,(100,100)),SUBSYS=BP01,
//            DCB=(LRECL=260,RECFM=VB)
```

Figure 46. Testing the CLIO/S Cross System Pipe Link by writing to an MVS BatchPipe (FCFFPIPW)

Varying On and Off Subchannels—FCFVARY

FCFVARY is an authorized program on MVS that lets you vary on the subchannels on the RS/6000 without logging onto the workstation and without having to log onto the MVS operator's console.

ESCON subchannels can be varied on entirely from the MVS side, using one of these methods:

- From the MVS console, issue MVS operator commands to vary on the subchannels
- From an authorized user ID, run the FCFVARY program to vary on the subchannels

To vary on BMCA/PCA subchannels on MVS, you must ensure that they are held open and started on the AIX side while MVS varies them on. CLIO/S provides two programs for doing this:

FCFVARY — This program, which runs on MVS, takes care of all of the steps necessary. You don't need to log onto AIX.

online — This program (described in “Starting BMCA/PCA Subchannels on the Workstation With the online Program” on page 275) runs on AIX. You use this program in conjunction with operator commands on the MVS side.

FCFVARY is the master program that runs on MVS. For ESCON subchannels, it issues the MVS commands to vary them on or off. For BMCA/PCA subchannels, it starts **varys**, the vary server, on the RS/6000. The vary server keeps the BMCA/PCA subchannels open and started on the workstation while FCFVARY issues the MVS commands to vary them on or off from the MVS side. See the Program Directory for instructions about installing the master and servant programs. You must customize the FCFVARY JCL before using it.

Customizing FCFVARY

You specify the subchannels you want to vary on or off by modifying the JCL for FCFVARY, which is provided in **SYS1.SFCFSAMP(FCFVARY)**. Include an entry following the DD VARY statement for each range of subchannels you want to vary on or off. Each entry has one of these formats:

```
risc1st risclast mvs1st mvslast chpid ONLINE  riscnode PCA adapternum parctchost
risc1st risclast mvs1st mvslast chpid OFFLINE riscnode PCA adapternum parctchost
risc1st risclast mvs1st mvslast chpid ONLINE  riscnode ESC adapternum parctchost
risc1st risclast mvs1st mvslast chpid OFFLINE riscnode ESC adapternum parctchost
```

Where:

risc1st Is the device number of the first RS/6000 subchannel, in hexadecimal

risclast Is the device number of the last RS/6000 subchannel, in hexadecimal

mvs1st Is the device number of the first MVS subchannel, in hexadecimal. The *mvs1st* and *mvslast* values must match the *unitrange* field of the PARCTC entry for this *parctchost*

mvslast Is the device number of the last MVS subchannel, in hexadecimal. The *mvs1st* and *mvslast* values must match the *unitrange* field of the PARCTC entry for this *parctchost*

chpid Is the MVS CHPID, in hexadecimal, associated with this adapter

ONLINE or **OFFLINE**
Specifies whether to vary the subchannel on or off

riscnode Is the name of the RS/6000 system. This name must *exactly* match the *nodeaddress* field of the PARCONF file on MVS. For example, if the PARCONF entry specifies `hpssl1`, this entry should contain `hpssl1`; if PARCONF specifies `hpssl1.kgn.ibm.com`, this entry should contain `hpssl1.kgn.ibm.com`

PCA or **ESC** Indicates whether this is a PCA or ESCON adapter

adapternum Is the adapter number. If the adapter is /**dev**/**cat0** or /**dev**/**esca0**, specify **0** for the *adapternum*; for /**dev**/**cat1** or /**dev**/**esca1**, use *adapternum* of **1**, and so on

parctchost Is the host node of the MVS system. This corresponds to a *hostname* specified in the PARCTC file

For example, if this statement is in your DD VARY statement:

```
40 4F 640 64F 1E online hpssl1 pca 0 kn1mvst
```

FCFVARY varies online the subchannel range 40 through 4F for the RS/6000 node named ***riscnode1***, using PCA adapter /**dev**/**cat0**. The MVS CHPID is 1E. This sample FCFVARY entry matches the IOCDS statements shown in Figure 3 on page 20, the sample PARCONF file shown in "The PARCONF File" on page 26 and the sample PARCTC file shown in "The PARCTC File" on page 32.

Using FCFVARY

To start or stop subchannels on your RS/6000 systems, run the FCFVARY program on MVS. FCFVARY differs from **online**, which performs a related function, in that you can manage the workstation subchannels from MVS, without logging onto the RS/6000 machines.

Before you run FCFVARY, make sure PARSRV is running on the workstations whose subchannels you want to modify.

On MVS, the sample JCL is:

```
//FCFVARY JOB 1,PP.NUMBER.5648.129,MSGLEVEL=1,REGION=4096K
//*-------------------------------------------------------
//* Vary ON/OFF MVS/AIX CTC Connections
//*-------------------------------------------------------
//FCFVARY  EXEC  PGM=FCFVARY
//STEPLIB  DD DSN=SYS1.APFLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*,DCB=BLKSIZE=3429
//SYSUDUMP DD SYSOUT=*,DCB=BLKSIZE=3429
//VARY     DD *
40 4f 640 64F 1E online hpssl1 pca 0 kn1mvst /* PCA to riscnode1
c0 cf 6c0 6cf 26 online hpssl2 ESC 0 kn1mvst /* ESCON to riscnode2
/*
```

Figure 47. JCL to Vary On and Off Workstation Subchannels (FCFVARY)

Starting BMCA/PCA Subchannels on the Workstation With the online Program

Description

Use **online** on your workstation to start BMCA/PCA subchannels during installation and setup. You can also use it to reconfigure your environment and restart the subchannels. **online** is one of two methods you can use to start BMCA/PCA subchannels on the workstation; the other is FCFVARY, described in “Varying On and Off Subchannels—FCFVARY” on page 273. The **online** program is available only on the RS/6000. It is located in **/usr/lpp/fcf/bin/online**.

ESCON subchannels do not need to be explicitly varied on from the workstation. Varying them on from MVS is sufficient.

Syntax

online *startsubch lastsubch* [*adapter_switch*]

Parameters

startsubch The address of the first subchannel to be varied online, in decimal.

lastsubch The address of the last subchannel to be varied online, in decimal.

adapter_switch If you omit *adapter_switch*, **online** starts the subchannels on adapter /**dev**/**cat0**. If you specify any value for *adapter_switch*, **online** processes adapter /**dev**/**cat1**.

Usage Notes

This program requires you to coordinate tasks between the workstation and MVS, following this sequence of events:

1. On the workstation, run **online**.
2. When **online** prompts you, go to MVS and vary on the corresponding MVS addresses.
3. After you have completed varying on the subchannels on MVS, tell the **online** program to exit.

Examples

The following figure shows how to use **online** to start a subchannel for the adapter /**dev**/**cat0**, attached to the MVS addresses 660 through 663.

Step	On AIX	On MVS
1.	Determine the input to the **online** command by taking the last two digits of the MVS addresses (660 and 663), and converting them from hexadecimal (60 and 63) to decimal (96 and 99).	

2.	Invoke the **online** program. You don't need to specify the adapter number when varying on subchannels associated with adapter /**dev**/**cat0**. Issue this command:
	`online 96 99`
	The program displays output similar to this:
	`argc : 3`
	`ONLINE : opening 370-PCA adapter /dev/cat0`
	`ONLINE - first_ctc : 96`
	`ONLINE - last_ctc : 99`
	`ONLINE - CIO_START subchannels`
	`ONLINE - CIO_START subch : 96`
	`ONLINE - CIO_START subch : 97`
	`ONLINE - CIO_START subch : 98`
	`ONLINE - CIO_START subch : 99`
	`ONLINE - option :`
	`ONLINE - option : 20`
	`Enter option 0 - Exit`	

3.		While the **online** program waits, vary on the subchannels:
		`v path(660-663,17),online`
		`v 660-663,online`

4.	Once the host addresses are varied on, the connection is established. Wrap up processing and exit from **online** by entering `'0'` at the prompt. For example: `Enter --> 0`	

To start workstation subchannels 670 through 67F for the second block multiplexer adapter (/**dev**/**cat1**), issue:

```
online 112 127 1
```

The last parameter is now 1, indicating the subchannels on the second block multiplexer or adapter (/**dev**/**cat1**). The addresses are calculated as before: take the last two digits of the MVS addresses (670 and 67F), and convert them from

hexadecimal (70 and 7F) to decimal (112 and 127). The workstation stdout is similar same except for the following lines:

```
argc : 4
ONLINE : opening 370-PCA adapter /dev/cat1
```

Appendix D. Controlling How CLIO/S Servers Are Started

This chapter describes how to use FCFREXX to generate JCL to control new server jobs that are submitted on MVS. It also describes how to specify which language each PARSRV server should use and describes considerations for migrating from earlier versions of the CLIO/S product.

This chapter also describes how to use FCFSHELL to control how new AIX server processes are started.

CLIO/S creates servant processes on MVS by submitting JCL. The JCL is dynamically created by FCFPARSR when it receives an OPNSRV request from a CLIO/S application. FCFPARSR calls a REXX program, FCFREXX. You modify FCFREXX to control the JCL that starts the servant job. "Using REXX/MVS to Generate PARSRV JCL" on page 280 describes how to use REXX to customize the JCL created by PARSRV.

Generating JCL for a program is a fairly intricate process. If you install CLIO/S with the default data set names, the sample FCFREXX program generates JCL that is sufficient for most purposes.

Note: If you use a security product other than RACF, you need to make the appropriate changes to the default JCL generation program.

Specifying the Location of the FCFREXX Routine

You specify the location of the FCFREXX JCL generation routine with the REXXJCL DD statement in the JCL that starts FCFPARSR.

Note: The REXXJCL DD statement must point to a partitioned data set containing a member named FCFREXX.

If you have multiple PARSRVs running on an MVS system, each can point to its own version of FCFREXX, if desired, or multiple PARSRVs can share the same FCFREXX.

Migrating From CLIO/S Version 1

This section describes considerations for customers migrating from CLIO/S Version 1 or Version 2.1. If you did not use CLIO/S Version 1 or Version 2.1, you can skip this section.

CLIO/S Version 1 provided only the pseudo-JCL language for JCL generation. CLIO/S Version 2.1 used REXX programs, but supported the pseudo-JCL language to help you migrate from the earlier version.

Note

The CLIO/S pseudo-JCL language is no longer supported. It is not possible to use new CLIO/S functions with the pseudo-JCL language.

© Copyright IBM Corp. 1993, 1995, 1996

Any JCL you created to start PARSRV servers will continue to run without change. However, IBM recommends that you modify any existing JCL that starts PARSRV servers at your installation to specify the REXX language (include the REXXJCL DD statement).

Running multiple PARSRVs can help you migrate. Run one PARSRV that uses your old pseudo-JCL language program; this lets your users submit jobs while you convert the customized JCL program from pseudo-JCL to REXX. If this PARSRV uses the default port number, your users need not take special action when starting their CLIO/S applications or when using CLIO/S's commands. While you convert your pseudo-JCL language program to REXX, use another PARSRV, running on a different port, for your testing. See “Running Multiple PARSRVs” on page 41 for more information.

Using REXX/MVS to Generate PARSRV JCL

The JCL that starts FCFPARSR, the PARSRV CLIO/S server, specifies the REXX program that generates JCL for servant processes started by that server. The REXXJCL DD statement lists the name of a partitioned data set with a member named FCFREXX. Figure 42 on page 270 shows sample JCL that uses REXX.

Services Available to FCFREXX

FCFREXX can call other REXX programs or programs in other languages. You can use any valid REXX syntax.

FCFREXX can use TSO calls and services if FCFPARSR was started in a TSO address space. This can help you debug problems in your FCFREXX program. To start FCFPARSR in a TSO address space, use the IKJEFT01 service as shown in the sample JCL in **SYS1.SFCFSAMP(FCFPRTSO)**. To see sample JCL, refer to Figure 43 on page 271.

The default JCL supplied with CLIO/S starts FCFPARSR in an MVS address space. TSO services are not available in this case. The FCFREXX sample code does not require a TSO address space.

See the chapter 'Using REXX in TSO/E and other MVS Address Spaces' in the *TSO Extensions Version 2 REXX/MVS User's Guide, SC28-1882*, for a discussion of which services are available in which environments.

Input to FCFREXX

FCFREXX is passed a variable number of arguments when it is called by FCFPARSR. These arguments are in the form of character strings which can be executed as REXX statements via the REXX **interpret** call.

For example, one of the arguments is MstPort, the socket associated with the master process which called FCFPARSR. If the socket is 4047, then the string passed to FCFREXX is `MstPort=4047`. If your REXX program reads the string into the variable `myvar`, use the REXX statement `interpret myvar` to execute `MstPort=4047`, assigning the value 4047 to the REXX variable MstPort.

Two types of arguments are passed:

- job control information

- CLIO/S environment variables

Job control information, information needed for creating MVS jobs, is passed to FCFREXX first. The variables, which are not in a specific order, are:

MVSUSER The MVS user ID.

MVSPW The MVS password. The value is unmasked.

SRVPROG The name of the program to be started as a servant, specified in the parameters to **clio_create** or **OPNSRV**. For example, if the master application passed **clio_create** or **OPNSRV** the command string `myprog new x=2` then PARSRV passes `SRVPROG=myserv` and `SRVPARMS='new x=2'`.

The default FCFREXX code uses this value to create special JCL for starting servants for the CLIO/S tape server (SRVPROG=TAPESRV), the CLIO/S File Transfer Program server (SRVPROG=FCFSERV), the CLIO/S FTP client (SRVPROG=CLFTP), the CLIO/S cancel routine, and the CLIO/S Cross System Pipe Link server (SRVPROG=CLPLINK). If you modify FCFREXX, make the appropriate changes to those sections.

SRVPARMS The argument string passed to the servant program, specified in the parameters to **clio_create** or **OPNSRV**. For example, if the master application passed **clio_create** or **OPNSRV** the command string `myprog new x=2` then PARSRV passes `SRVPROG=myserv` and `SRVPARMS='new x=2'`.

SRVNUM The number of servants started by this PARSRV server, including the current request. The default FCFREXX uses this value to construct job names for the servants it starts, to provide compatibility with the JCL generation in CLIO/S Version 1. Feel free to modify this section to create different job numbers if needed at your installation.

CLIO/S environment variables follow the job control information. (See “Setting CLIO/S Environment Variables” on page 36 for more information.) The first environment variable is MstPort. The rest may be in any order. The following list describes environment variables commonly passed by PARSRV to servant programs. If other variables are defined by the master program, they will be included.

MstPort The socket number of the master process.

MstHid The internet address of the master process.

MstPid The process id of the master process.

CLIODEB The CLIODEB setting of the master.

CLIOADEB The CLIOADEB setting of the master.

CLIOFLAG This flag is reserved for use by CLIO/S internals. Do not change it.

Any additional CLIO/S environment variables set for the master process are passed similarly, following these known variables. There is no practical limit to the number environment variables that can be passed to FCFREXX. These may include other environment variables defined by CLIO/S (described in “Setting CLIO/S Environment Variables” on page 36), as well as variables created by the master application with the **clio_exp** routine (see page 177).

Passing Environment Variables to the Servant

You must ensure that the JCL created by your REXX program sets the necessary environment variables following the CLIOENV DD statement, so that the servant application can establish communication with the master. This includes the value of the MstPort argument and those that follow it in the preceding list. Be cautious when changing the value of any of these variables (for example, to have the servant run with a different CLIODEB setting than the master); if master and servant values don't match for some variables, the results are unpredictable.

FCFPARSR maintains its own copies of all these variables. Changing the values of the variables passed to FCFREXX has no effect on FCFPARSR behavior.

Setting the Return Code In FCFREXX

FCFREXX should return an integer greater than zero when it terminates. A return code less than or equal to zero causes FCFPARSR to return –1 to the master CLIO/S application, indicating that the servant application could not be started.

Using the SAY and TRACE Commands

If FCFPARSR runs in batch mode, the output of the `SAY` and `TRACE` commands is printed to SYSTSPRT. If the JCL for FCFPARSR includes a SYSTSPRT DD statement, you can retrieve the output. This lets you collect messages and debugging information (for example, using `TRACE R` to display the commands lines with variables resolved). In batch mode, you cannot use `TRACE` for interactive debugging.

FCFPARSR can be started from the TSO command line; this can help you develop and debug both JCL generation programs and CLIO/S distributed applications. In this mode, `SAY` and `TRACE` output is printed to the terminal, and `TRACE` can be used interactively. To get control of your console back in this mode, stop FCFPARSR from a workstation using the CLIO/S command `parstop` *`mvsnode`*, where *`mvsnode`* is the name workstation CLIO/S applications use to refer to the MVS machine.

Under the Covers of FCFREXX

FCFPARSR allocates a JES internal reader with the DD name FCFJCL. FCFREXX is then started via the MVS service IRXEXEC. FCFREXX writes JCL statements into FCFJCL, creating a valid JCL deck. When FCFREXX returns, FCFPARSR closes FCFJCL, submitting the job to MVS.

See the *TSO Extensions Version 2 REXX/MVS User's Guide*, SC28-1882 and the *TSO Extensions Version 2 REXX/MVS Reference*, SC28-1883 for details information about the syntax and usage of REXX in the MVS environment.

Sample FCFREXX Program

Sample FCFREXX code is supplied with CLIO/S in the dataset **SYS1.SFCFSAMP(FCFREXX)**. Figure 48 on page 284 shows the sample code. If your installation installed CLIO/S using the default dataset names, FCFREXX works without any changes. Otherwise, you must change at least the STEPLIB dataset names.

Your installation may customize FCFREXX. The default code may be modified in any way, as long as valid REXX syntax is used to create valid JCL.

Any changes made to FCFREXX take effect immediately. There is no need to refresh FCFPARSR, as you would with the pseudo-JCL method. For this reason, you should work on a *copy* of the FCFREXX member rather than the FCFREXX member currently being used by PARSRV.

```
/* REXX - tell TSO that this is a REXX exec                          */
/*********************************************************************
*
*     This module is "RESTRICTED MATERIALS OF IBM"
*     Licensed Materials - Property of IBM
*
*     5648-129
*     (C) COPYRIGHT IBM CORP.  1995, 1996
*     All rights reserved.
*     US Government Users Restricted Rights - Use,
*     duplication or disclosure restricted by GSA ADP
*     Schedule Contract with IBM Corp.
*
*********************************************************************/
/* End copyright. */
/*********************************************************************
*
* This REXX exec is provided as sample code with the CLIO/S product.
* When called by FCFPARSR, it produces JCL statements, writes them
* into a JES internal reader, and submits the JCL by closing the reader.
* The submitted JCL starts a servant process requested by a CLIO/S
* master process.
*
* The JCL created by this exec will work 'as is' if your installation
* placed CLIO/S components in the default datasets. If not, the STEPLIB
* variables must be modified accordingly.
*
* This exec must return a value > 0, otherwise FCFPARSR will report
* back to the master process that the servant creation process failed.
*
* See the CLIO/S User's Guide for more information on this exec.
*
*********************************************************************/

numargs = arg()                        /* How many arguments?        */
do i = 1 to numargs                    /* Save the arguments before  */
   p.i = arg(i)                        /*    initialization.         */
   end
call initialize                        /* Do some preparation.       */

/*********************************************************************
* This exec is written so as to make it easy to modify the items that
* most users will have to modify. Those items are gathered together
* and some complexities are hidden.
*********************************************************************/
/*********************************************************************/
/*********************************************************************/
/*********************************************************************/
/*                    Begin user modification area                   */
/*                    Begin user modification area                   */
/*                    Begin user modification area                   */
/*                    Begin user modification area                   */
/*                    Begin user modification area                   */
/*                    Begin user modification area                   */
/*                    Begin user modification area                   */
/*********************************************************************/
/*********************************************************************/
```

Figure 48 (Part 1 of 16). Sample JCL Generation Program using REXX (FCFREXX)

```
/*********************************************************************
* Now set some variables that will be used during JCL generation.
* Change these as desired.
*********************************************************************/
batchpipes   = 'BP01'   /* If you have MVS batch pipes, batchpipes
                           should be set to the subsystem name for
                           MVS batch pipes. */
jobname = MVSUSER||substr(jobstring,(SRVNUM-1)//length(jobstring)+1,1)
              /* Choose jobletter based on SRVNUM, append to userid  */
              /* to create jobname. So jobnames are useridB, useridC,*/
              /* useridD, etc. Change as desired, CLIO/S has no      */
              /* jobname requirements.                               */
/* if CLIOARB <> ''        */  /* Has CLIOARB been passed from AIX?  */
/* then jobname = CLIOARB  */
              /* This alternate method of setting the jobname allows */
              /* the AIX process to choose the MVS jobname. See the  */
              /* User's Guide for more information on CLIOARB.       */

/*********************************************************************
* 'MVSUSER' and 'MVSPW' are passed from to FCFREXX by FCFPARSR. They
* are originally sent from the client process, which found them in the
* .netrc file or queried them at the command line.
*
* The above variables are now used to create job cards. Change, add or
* delete entries as required.
*********************************************************************/
call jc("//"jobname  "JOB (572260,6MM,414-1),'"MVSUSER"',")
call jc("//   CLASS=A,REGION=17M,MSGCLASS=4,MSGLEVEL=(1,1),")
call jc("//   NOTIFY="MVSUSER",TIME=1000,")
call jc("//   USER="MVSUSER",PASSWORD="MVSPW)

/*********************************************************************
*
*  The following STEPLIB datasets should reflect your own installation.
*  Change, add or delete entries as required. Other likely candidates
*  are the Pascal (SAMPRUN2) libraries needed by TCP/IP version 2, and
*  the Fortran libraries (VSF2LOAD) needed if you code applications
*  using the Fortran interface API of CLIO/S.
*
*********************************************************************/
clioload = SYS1.SFCFLOAD
cliosamp = SYS1.SFCFSAMP
call sl("//STEPLIB DD DISP=SHR,DSN="clioload)
call sl("//        DD DISP=SHR,DSN="cliosamp)
call sl("//        DD DISP=SHR,DSN=SYS1.SEDCLINK")
```

Figure 48 (Part 2 of 16). Sample JCL Generation Program using REXX (FCFREXX)

```
/*********************************************************************
*
*  Other miscellaneous DD cards. Change, add or delete entries as
*  required. The SYSTCPD job card points to the tcpip.data dataset. If
*  tcpip.data is not in a default location, then complete the DSN
*  portion of the JCL statement and uncomment it.
*
*********************************************************************/
call dd("//SYSPRINT DD   SYSOUT=*")
call dd("//SYSERR   DD   SYSOUT=*")
call dd("//SYSTERM  DD   SYSOUT=*")
call dd("//STDERR02 DD   SYSOUT=*")
call dd("//STDOUT02 DD   SYSOUT=*")
call dd("//SYSUDUMP DD   SYSOUT=*")
call dd("//SYSDUMP  DD   SYSOUT=*")
call dd('//*SYSTCPD  DD DISP=SHR,DSN=(location of tcpip.data)')

/*********************************************************************
*
*  Special server environment variables. Usually all the required
*  environment variables are passed by the client process, but
*  sometimes it is advantageous to override them or add new ones.
*  These are commented out, change and uncomment as required.
*
*********************************************************************/
/* call env('CLIODEB=-1')         */  /* Override inherited debug    */
                                       /*    flag setting             */
/* call env('CLIONODE=127.0.0.1') */  /* Explicitly set local tcp/ip */
                                       /*    address                  */
/*********************************************************************/
/*********************************************************************/
/*********************************************************************/
/*                     End user modification area                    */
/*                     End user modification area                    */
/*                     End user modification area                    */
/*                     End user modification area                    */
/*                     End user modification area                    */
/*                     End user modification area                    */
/*                     End user modification area                    */
/*********************************************************************/
/*********************************************************************/
/*********************************************************************
*
*  You can modify any of the other code in this exec, but it will be
*  more work to analyse the code and get the results you are after.
*
*********************************************************************/
/*********************************************************************
```

Figure 48 (Part 3 of 16). Sample JCL Generation Program using REXX (FCFREXX)

```
*
* Call the appropriate JCL generation routine, based on the name of
* the application to be started.
*
* Valid JCL statements must be produced. All statements must be 80
* characters or less in length.
*
* JCL statements are placed into the stem variable 'jcl'. The entire
* stem will then be written into the JES internal reader in one
* operation. (This is not a requirement. It would also be possible to
* write statements one at a time into the reader as they are created.)
*
* For started tasks, see the routine 'stask' below. It is not used by
* any of the default code, and will have to adapted for your own
* environment.
*
* Change these as desired.
*********************************************************************/
if SRVPROG == 'FCFFEXE' then /* FCFFEXE program for appl linker.     */
   do                        /*    fcffexeInit will change SRVPROG,  */
   call fcffexeInit          /*    so this call must be made outside */
   end                       /*    of the following select.          */

select;
  when SRVPROG == 'TAPESRV'  /* CLIO/S Tape server.                  */
     then call tapesrv

  when SRVPROG == 'FCFFSERV' /* CLIO/S FTP server.                   */
     then call fcffserv

  when SRVPROG == 'CLPLINK'  /* CLIO/S Pipe link.                    */
     then call clplink

  when SRVPROG == 'CLFTP'    /* CLIO/S FTP client.                   */
     then call clftp

  when SRVPROG == 'FCFFPIPR' /* Test program for pipe reading.       */
     then call fcffpip

  when SRVPROG == 'FCFFPIPW' /* Test program for pipe reading.       */
     then call fcffpip

  when SRVPROG == 'FCFFPMNX' /* Test program for app linker hopping  */
     then call fcffpmnx

  when SRVPROG == 'C*A*N'    /* CLIO/S Cancel routine.               */
     then call fcfcanc

  otherwise call default     /* Default JCL.                         */
  end;                       /* end select                           */

call j(rexxtag)         /* insert comment at end of JCL for info only */
call print_jcl          /* Copy JCL to FCFPARSR SYSTSPRT, if desired. */
```

Figure 48 (Part 4 of 16). Sample JCL Generation Program using REXX (FCFREXX)

```
/*********************************************************************
*
*  By this point the required JCL has been written into the 'jcl.'
*  array. Now we write the JCL statements into the JES internal reader
*  or return them to fcfparsr depending on whether NoMstHid or
*  FCFRETJOBNUM is set.
*
*********************************************************************/
if (NoMstHid = 1) | (FCFRETJOBNUM = 1) then
  do
    /* Return JCL lines to FCFPARSR. This is done when the JCL is
       started for an application linker. */
    if jcl_lines > 0 then
      do
        /* Some JCL was created. Return it. */
        jclStr = ''
        do i = 1 to jcl_lines
          jclStr = jclStr||left(jcl.i,80)
        end
        return jclStr /* Return jcl */
      end
    else
      do
        /* No jcl was created. Return 0 to indicate error. */
        return 0 /* Return 0 to indicate error */
      end
  end
else
  do
    /* Write JCL to internal reader. This is the default. */
    'EXECIO * DISKW FCFJCL ( STEM JCL.'  /* Write JCL to FCFJCL.     */
    if rc <> 0 then do
       say 'Write error:' rc
       exit 0    /* Will cause FCFPARSR to report failure to master. */
       end

    return jcl_lines /* WARNING: return value must be > 0, otherwise */
                     /*    FCFPARSR will report failure back to master  */
                     /*    process.                                     */
  end
exit
```

Figure 48 (Part 5 of 16). Sample JCL Generation Program using REXX (FCFREXX)

```
/*********************************************************************/
generic:
/*********************************************************************
* Create a generic set of JCL cards
*********************************************************************/
call add_jobcards                    /* insert predefined job cards */
if usingFcffexe = 1 then
  call fcffexeFirst
call j(exec_string)
if no_more_parms <> 'true' then call formSrvparms
if (CLPLINKX <> "CLPLINKX") & (SRVPROG = "CLPLINK") then
  do
    /* CLPLINK provides a mechanism to pass information to JCL
       generation through the CLPLINKX variable. The value for CLPLINKX
       is set to whatever was passed in though the -X option on the
       CLPLINK command. You will only enter this section of the code
       if the -X option was used. You can use the values passed in
       CLPLINKX to generate additional JCL. */
  end
call add_stepcards               /* insert predefined steplib cards */
call add_ddcards                     /* insert predefined dd cards   */

call add_env                         /* Add environment variables    */

if usingFcffexe = 1 then
  call fcffexeLast


return

/*********************************************************************/
default:
/*********************************************************************
*
*********************************************************************/
exec_string = "//"SRVPROG"   EXEC PGM="SRVPROG","
call generic

return
```

Figure 48 (Part 6 of 16). Sample JCL Generation Program using REXX (FCFREXX)

```
/*********************************************************************/
fcffserv:
/*********************************************************************
*
*********************************************************************/

SRVPARMS2 = "-c "MstPort","MstHid SRVPARMS /* Use SRVPARMS2 instead of
                              SRVPARMS if CLIOENV will not be passed
                              through the JCL (ie if add_env is not
                              called below). */
/* Make sure we can run under TSO. If we can't, run as a normal job.
   When TSO isn't used the ls subcommand will not work, but at least the
   user many other commands. The only time that we should not
   be able to run under TSO is when the application linker starts clftp.
   In this case, the callStr string below may be too long to fit on
   one record of JCL input. */
callStr = "CALL '"clioload"("SRVPROG")' '"SRVPARMS"'"
if (length(callStr) <= 72) then
  do
    call add_jobcards                /* insert predefined job cards */
    call j("//"SRVPROG"  PROC")
    call j("//"SRVPROG"  EXEC PGM=IKJEFT01")
    call add_stepcards            /* insert predefined steplib cards */
    call dd("//SYSIN    DD   DUMMY")
    call dd("//INSPLOG  DD   SYSOUT=4,")
    call dd("//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)")
    call dd("//REXXLST  DD   DSN="cliosamp",DISP=SHR")
    call dd("//SYSTSPRT DD   SYSOUT=4,DCB=BLKSIZE=3429")
    call add_ddcards                /* insert predefined dd cards   */
    call j("//    PEND")
    call j("//P1  EXEC "SRVPROG)
    call j("//SYSTSIN  DD   *")
    call j(callStr)
  end
else
  do
    call add_jobcards                /* insert predefined job cards */
    call j("//"SRVPROG"  EXEC PGM="SRVPROG",")
    call formSrvparms
    call add_stepcards            /* insert predefined steplib cards */
    call add_ddcards                /* insert predefined dd cards   */
  end

if CLFTPX <> "CLFTPX" then
  do
    /* CLFTP provides a mechanism to pass information to JCL
       generation through the CLFTPX variable. The value for CLFTPX
       is set to whatever was passed in though the -X option on the
       CLFTP command. You will only enter this section of the code
       if the -X option was used. You can use the values passed in
       CLFTPX to generate additional JCL. */
  end

call add_env                    /* Add environment variables */

return
```

Figure 48 (Part 7 of 16). Sample JCL Generation Program using REXX (FCFREXX)

```
/*********************************************************************/
clplink:
/*********************************************************************
*
*********************************************************************/
SRVPARMS2 = "-c "MstPort","MstHid SRVPARMS /* Use SRVPARMS2 instead of
                              SRVPARMS if CLIOENV will not be passed
                              through the JCL (ie if add_env is not
                              called below). */

exec_string = "//"SRVPROG"  EXEC PGM="SRVPROG","
call generic

return

/*********************************************************************/
clftp:
/*********************************************************************
*
*********************************************************************/
SRVPARMS2 = "-c "MstPort","MstHid SRVPARMS /* Use SRVPARMS2 instead of
                              SRVPARMS if CLIOENV will not be passed
                              through the JCL (ie if add_env is not
                              called below). */

exec_string = "//"SRVPROG"  EXEC PGM="SRVPROG","
call generic

return

/*********************************************************************/
tapesrv:
/*********************************************************************
*
*********************************************************************/
exec_string = "//"SRVPROG"   EXEC PGM="SRVPROG",PARM='NOSTAE,NOSPIE'"
no_more_parms = 'true'
call generic

return
```

Figure 48 (Part 8 of 16). Sample JCL Generation Program using REXX (FCFREXX)

```
/*********************************************************************/
fcffpip:
/*********************************************************************
*
* Submit JCL for the fcffpipr and fcffpipw commands.
*
*********************************************************************/
parse value SRVPARMS with pipeName otherParms  /* Get the name of
                    the file we will call the pipe. */
SRVPARMS = "DD:PIPE" otherParms /* Use dd name PIPE as the pipe name */
pipeName = translate(pipeName) /* Translate to upper case */

call add_jobcards                    /* insert predefined job cards */
if usingFcffexe = 1 then
  call fcffexeFirst
call j("//"SRVPROG"  EXEC PGM="SRVPROG",")
call formSrvparms
call add_stepcards                /* insert predefined steplib cards */
if (SRVPROG == 'FCFFPIPR') then
  do
    /* Define batch pipe for fcffpipr. */
    call dd("//PIPE     DD DSN="pipeName",DISP=SHR,SUBSYS="batchpipes",")
    call dd("//         DCB=(LRECL=260,RECFM=VB)")
  end
else
  do
    /* Define batch pipe for fcffpipw. */
    call dd("//PIPE     DD DSN="pipeName",DISP=(NEW,DELETE,DELETE),")
    call dd("//         SPACE=(100,(100,100)),SUBSYS="batchpipes",")
    call dd("//         DCB=(LRECL=260,RECFM=VB)")
  end
call add_ddcards                     /* insert predefined dd cards   */
call add_env                         /* Add environment variables    */
if usingFcffexe = 1 then
  call fcffexeLast

return

fcffpmnx:
/*********************************************************************
*
* Application linker node hopping program.
*
*********************************************************************/
exec_string = "//"SRVPROG"   EXEC PGM="SRVPROG","
call generic

return
```

Figure 48 (Part 9 of 16). Sample JCL Generation Program using REXX (FCFREXX)

```
/*********************************************************************/
fcfcanc:
/*********************************************************************
*
* Notes about FCFCANC:
*    -  The JCL generated here calls FCFCANC, which is another
*       REXX exec. You could just as well insert FCFCANC code here.
*       This was not done in order to match the previous method.
*********************************************************************/
call add_jobcards                    /* insert predefined job cards */
call j("//FCFCANC   EXEC PGM=IKJEFT01")
call j("//SYSTSPRT DD   SYSOUT=*")
call j("//SYSTSIN  DD   *")
call j("EXEC '"cliosamp"(FCFCANC)' '"SRVPARMS"' EXEC")
call j("/*")

return

/*********************************************************************/
fcffexeInit:
/*********************************************************************
*
* Initialize to use fcffexe as the first and last step of every
* job. To do this, we must parse the SRVPARMS into various parts.
*
*********************************************************************/

parse value SRVPARMS with flags '@@@@' SRVPROG SRVPARMS
SRVPROG = translate(SRVPROG) /* Translate to upper case */
usingFcffexe = 1
fcffexeFirstParms = flags '-F'
fcffexeLastParms = flags '-L'

return
/*********************************************************************/
fcffexeFirst:
/*********************************************************************
*
* Create JCL that can be used to invoke fcffexe as the first step
* of a job.
*
*********************************************************************/

call j("//FCFFEXEF EXEC PGM=FCFFEXE,")
call j("// PARM='"fcffexeFirstParms"'")
call add_stepcards                /* insert predefined steplib cards */
call add_ddcards                      /* insert predefined dd cards   */
call j("//* End of first FCFFEXE")

return
```

Figure 48 (Part 10 of 16). Sample JCL Generation Program using REXX (FCFREXX)

```
/*********************************************************************/
fcffexeLast:
/*********************************************************************
*
* Create JCL that can be used to invoke fcffexe as the first step
* of a job.
*
*********************************************************************/

call j("//FCFFEXEL EXEC PGM=FCFFEXE,")
call j("// PARM='"fcffexeLastParms"'")
call add_stepcards                /* insert predefined steplib cards */
call add_ddcards                      /* insert predefined dd cards   */
call j("//* End of last FCFFEXE")

return
/*********************************************************************/
stask:
/*********************************************************************
*
*  Run the servant process as a started task. None of the defaults use
*  this method, but it is provided here as a template. This template
*  must be modified before use. See the User's Guide for discussion of
*  servants as started tasks.
*
*********************************************************************/
call add_jobcards                     /* insert predefined job cards */
call j("/*$VS,'S "SRVPROG"."MVSUSER",PARM=("MstPort","MstHid")'")

/* Started tasks don't get the CLIOENV DD card. See User's Guide.    */

return

/*********************************************************************/
j: /* Add a string to 'jcl.' array, update count of JCL lines created.*/

parse arg argstring
jcl_lines = jcl_lines + 1
jcl.jcl_lines = argstring

return
```

Figure 48 (Part 11 of 16). Sample JCL Generation Program using REXX (FCFREXX)

```
/*********************************************************************/
env:   /* Create environment variable definition                     */
parse arg argstring
parse var argstring lhs '=' rhs
lhs = strip(lhs)
rhs = strip(rhs)
previously_defined = 'false'
do i = start_env to numargs  /* search defined varaibles           */
   parm = p.i
   if pos('=',parm) = 0 then iterate
   parse var parm leftside '=' rightside
   leftside = strip(leftside)
   rightside = strip(rightside)
   if leftside == lhs  /* already defined, overwrite */
      then do
      p.i = lhs'='rhs /* overwrite parm */
      previously_defined = 'true'
      leave
      end
   end i
if previously_defined = 'false' /* new, add to list */
   then do
   numargs = numargs + 1
   p.numargs = lhs'='rhs
   end
return

/*********************************************************************/
add_env:   /* Add environment variables to jcl.                      */
if datatype(start_env, 'W') = 1        /* Any environment variables?  */
   then do
   call j("//CLIOENV  DD   *")         /* Create CLIOENV DD card,     */
   /* If you must set the CLIONODE variable, set it by uncommenting
      and modifying the following lines. */
   /* jcl_lines = jcl_lines + 1
      jcl.jcl_lines = "CLIONODE=hostname"  */
   do i = start_env to numargs         /*   write variables into it.  */
      call j(p.i)
      end i
   call j("/*")                        /* Terminate the DD card.      */
   end

return
```

Figure 48 (Part 12 of 16). Sample JCL Generation Program using REXX (FCFREXX)

```
/*********************************************************************
*
*  Create predefined JCL cards. Multiple entries are used to allow a
*  single subroutine to handle multiple types without bothering the
*  user with extra complexity.
*
*********************************************************************/
jc:                                  /* Create predefined job cards  */
if type == '' then type = 'jc'
sl:                                /* Create predefined steplib cards  */
if type == '' then type = 'sl'
dd:                                   /* Create predefined dd cards  */
if type == '' then type = 'dd'

parse arg argstring
num_cards.type = num_cards.type + 1
cardindex = num_cards.type
card.type.cardindex = argstring
type = ''

return
/*********************************************************************/
no: /* no-op string handler */
return
/*********************************************************************/
add_jobcards:                       /* Add predefined job cards to jcl */
if type == '' then type = 'jc'
add_stepcards:                  /* Add predefined steplib cards to jcl */
if type == '' then type = 'sl'
add_ddcards:                         /* Add predefined dd cards to jcl */
if type == '' then type = 'dd'

do cardcount = 1 to num_cards.type      /* insert predefined jobcards */
   call j(card.type.cardcount)
   end

type = ''
return
```

Figure 48 (Part 13 of 16). Sample JCL Generation Program using REXX (FCFREXX)

```
/*********************************************************************/
formSrvparms:
/*********************************************************************
*
* This subroutine formats parameters so that they can spread over
* several lines.
*
*********************************************************************/

if (length(SRVPARMS) < 62) & (SRVPROG <> "CLPLINK") then
  do
    /* The parms all fit on one line. */
    call j("// PARM='"SRVPARMS"'")
  end
else
  do
    if (length(SRVPARMS) > 100) | (SRVPROG = "CLPLINK") then
      do
        /* The parameters are too long. This is a problem unless we
           are using clftp or clplink, which have a special mechanism
           for handling this. */
        if (SRVPROG = "CLPLINK") | (SRVPROG = "CLFTP") | ,
           (SRVPROG = "FCFFSERV") then
          do
            /* Do special handling for clftp and clplink. We can pass
               the parameters in through dd name PARMDD. */
            parse value SRVPARMS with dbgTest dbgVal theRest
            if (dbgTest = "-D") then
              do
                /* Pass debugging through so that we get info on the
                   reading in of the arguments. */
                parse value dbgVal with num "," theRest
                call j("// PARM='-z -D "num"'")
              end
            else
              do
                /* Don't worry about debugging. */
                call j("// PARM='-z'")
              end
            call j("//PARMDD   DD   *")
            call j(SRVPROG" @"); /* See note below for explanation
                          of why @ is used. */
            parms = SRVPARMS
            do while (length(parms) > 0)
              if (length(parms) <= 60) then
                do
                  /* Write the last line. Put "@" at the end of the
                     line so that when the program reads the line it
                     does not truncate blanks. The program will check
                     for @ and ignore it if it is there. */
                  call j(parms"@")
                  parms = ""
                end
              else
                do
```

Figure 48 (Part 14 of 16). Sample JCL Generation Program using REXX (FCFREXX)

```
                  /* Write the next line. Put "@" at the end of the
                     line so that when the program reads the line it
                     does not truncate blanks. The program will check
                     for @ and ignore it if it is there. */
                  call j(substr(parms,1,60)"@")
                  parms = substr(parms,61)
                end
            end
            call j("/*")
          end
        else
          do
            /* The parameters are too long. Report the error. */
            say "ERROR: The parameter string for" SRVPROG "is too long:"
            say "      " SRVPARMS
            call j("// PARM='"SRVPARMS"'")
          end
      end
    else
      do
        /* The parms are not too long, but they don't fit on one line.
           Break them up into two lines. */
        call j("// PARM='"substr(SRVPARMS,1,62)"X")
        call j("//             "substr(SRVPARMS,63)"'")
      end
  end

return
/*********************************************************************/
print_jcl:       /* Copy JCL to FCFPARSR SYSTSPRT, but hide password. */

say copies('-', 80) /* eyecatching separator */

do i = 1 to jcl_lines
   if (pos('PASSWORD='||MVSPW, jcl.i)) > 0
      then do until pos('PASSWORD='||MVSPW, temp) == 0
      if length(MVSPW) = 0 then leave
      temp = jcl.i
      parse var temp first 'PASSWORD=' rest
      lpw = length(MVSPW)
      rest = copies('*', lpw) || substr(rest, lpw+1)
      temp = first || 'PASSWORD=' || rest
      say temp
      end
   else say jcl.i               /* Copy JCL to SYSTSPRT, if desired. */
   end

return
```

Figure 48 (Part 15 of 16). Sample JCL Generation Program using REXX (FCFREXX)

```
/*********************************************************************/
initialize:

parse source . . . . . environment . /* Just for information,      */
                                     /*    could be 'MVS' or 'TSO'.*/
jobstring = 'BCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'
jcl_lines = 0
num_cards. = 0
type = ''
rexxtag = "//* JCL generated by FCFREXX on" date('U') "at" time('C')

ugly_args = 'SRVPARMS CLPLINKX CLFTPX CLFTPSRV CLIODBGFILE'

do i = 1 to numargs                  /* Process the arguments.      */
   parm = p.i
   if pos('=',parm) = 0 then iterate   /* Not 'interpret'able         */
   parse var parm leftside '=' rightside
   leftside = strip(leftside)
   rightside = strip(rightside)
   if leftside == 'MstPort' &,       /* Start of Environment data,  */
      datatype(start_env, 'W') <> 1
      then start_env = i             /*    capture for later use.   */
   if leftside == 'NoMstHid' &,      /* Start of Environment data,  */
      datatype(start_env, 'W') <> 1
      then start_env = i             /*    capture for later use.   */

   if wordpos(leftside, ugly_args) <> 0
      then do

      rightside = c2x(rightside)     /* special handling so         */
                                     /*    'interpret' is not       */
                                     /*    confused by quotes or    */
                                     /*    other special chars.     */

      interpret leftside'=x2c(rightside)' /* Execute the parameter,  */
                                     /*    if desired.              */
      end
   else interpret leftside'='rightside
   end
if (substr(SRVPARMS,1,1) = "'") & (length(SRVPARMS) > 1) then do
  SRVPARMS = substr(SRVPARMS,2,length(SRVPARMS)-2)
  end

return
```

Figure 48 (Part 16 of 16). Sample JCL Generation Program using REXX (FCFREXX)

FCFSHELL

CLIO/S can start new processes on remote AIX systems in either of two ways. The method used by earlier releases of CLIO/S is simply to exec the desired process (or rexec it if the local system is also AIX). The other method, new with CLIO/S Version 2 Release 2, is to exec (or rexec) a shell script called **FCFSHELL**, which in turn executes the desired process. This method provides increased flexibility and control. It is the default.

Setting the CLIOSHELL Environment Variable

The CLIOSHELL environment variable setting on the local machine determines whether FCFSHELL is used to start the remote process.

If CLIOSHELL is not set, **/bin/FCFSHELL** is called on the remote system. A working sample **/bin/FCFSHELL** is created by the installation process.

If CLIOSHELL is set to a particular path and file, then that file will be executed instead of the default FCFSHELL. For example, if you set CLIOSHELL to **$HOME/myfcfshell**, then myfcfshell in your home directory will be run. myfcfshell would then have the responsibility of starting the new CLIO/S process.

If CLIOSHELL is set to NO (`export CLIOSHELL=NO`), then FCFSHELL will not be called, and and the new process is started directly via exec or rexec.

The same exec/rexec parameters are used regardless of the value of CLIOSHELL.

Modifying FCFSHELL

The sample **FCFSHELL** provided with the product should work without modification. It receives the name of the desired program as a parameter, issues a message, and then executes the new program. The new process inherits the **FCFSHELL** environment, which includes the information necessary to establish contact with the originating process.

You can modify **FCFSHELL** to suit your own purposes. You may want to start other processes before or after the CLIO/S process, change the environment, make log entries, etc..

Be careful about modifying the environment that is passed to FCFSHELL. For example, MstPort and MstHid must be available to the new server process or it will not be able to contact the client process.

In many cases the parameters passed to FCFSHELL contain quotes and special characters. Be sure that your modified FCFSHELL is able to handle the parameters you will send it.

Regardless of other changes you may make, FCFSHELL must eventually execute the desired program.

CLIOARB

A special CLIO/S environment variable, CLIOARB, is provided for use with **FCFSHELL**. CLIOARB is an arbitrary string set by the user at the master process. It is passed to **FCFSHELL** by CLIO/S when the remote servant is started. **FCFSHELL**, with appropriate user modifications, can use this information to trigger special behavior. CLIOARB is not referenced by CLIO/S processes other than **FCFSHELL**.

When a server is being started on MVS, CLIOARB is also available to FCFREXX.

Be careful about the value of CLIOARB

As CLIOARB is passed between shell scripts and REXX execs, strange things can happen if there are quotes, imbedded blanks, or lower case letters. If you wish to use complex strings as values for CLIOARB, do some testing to make sure that the string that is sent is the same one that is received.

FCFSHELL Sample Code

```
#!/bin/ksh
#
# Start copyright.
#
#     This module is "RESTRICTED MATERIALS OF IBM"
#     Licensed Materials - Property of IBM
#
#     "5665-637 CLNETTAPE (C) Copyright IBM Corp. 1996
#     (C) COPYRIGHT IBM CORP. 1996
#     All rights reserved.
#     US Government Users Restricted Rights - Use,
#     duplication or disclosure restricted by GSA ADP
#     Schedule Contract with IBM Corp.
#
# End copyright.


if [ $# -gt 0 ]
 then
   print "$0: on host \"`/bin/hostname -s`\" execute command $@"
   eval $*
 else
   print "$0: on host \"`/bin/hostname -s`\" Error! No command specified."
fi
```

Figure 49. Sample FCFSHELL shell script

Modifying FCFSHELL

Here are some examples of simple modifications you could make to FCFSHELL.

- Add the AIX **date** command to FCFSHELL. This results in a timestamp being issued when the CLIO/S process is started. This can be helpful in process management and debugging.
- Export the LIBPATH environment variable.

  `export LIBPATH=`**`/lib:/project/clio/v2.2/testbin`**

  This is necessary when using the user I/O exits described in Appendix A, “User Specified I/O Exits for CLFTP and CLPLINK” on page 251.
- Turn on the CLIO/S debug messages for the remote process without turning them on for the local process. Here are two variations of this:
  - Put the commands directly into CLIOARB, and have FCFSHELL execute CLIOARB as a command.

    On the local system, set CLIOARB and CLIOSHELL as follows:

```
export CLIOSHELL=/$HOME/FCFSHELL
export CLIOARB='"export CLIODEB=-1; export CLIODFILE=$HOME/cliodebug.out"'
```

Note that both double and single quotes are used to ensure that FCFSHELL receives the correct string. Then in your copy of FCFSHELL in your home directory on the remote system, add the following:

```
eval $CLIOARB
```

After these changes are made, CLIO/S processes started from the local system will not generate debug messages, but the servants on the remote system will write debug messages into **$HOME/cliodebug.out**.

– Another approach would be to simply use CLIOARB as a flag, and put the actual commands into FCFSHELL. On the local system, set CLIOARB and CLIOSHELL as follows:

```
export CLIOARB=debug
export CLIOSHELL=/$HOME/FCFSHELL
```

Then in your copy of FCFSHELL in your home directory on the remote system, add the following:

```
if [ $CLIOARB = "debug" ]
  then
    export CLIODEB=-1
    export CLIODFILE=$HOME/cliodebug.out
fi
```

Appendix E. Dynamic Device Allocation on MVS

This appendix describes the parameters you can pass to MVS dynamic allocation. It assumes you are familiar with MVS dynamic allocation parameters and that you only need to determine the equivalent CLIO/S parameters to use. For more information about MVS device allocation, see *MVS/ESA JCL User's Guide*.

Table 11 shows each MVS device allocation parameter, its corresponding CLIO/S parameter, and any applicable parameter values.

You can enter the CLIO/S allocation parameters in upper, lower, or mixed case. The minimum abbreviation for each parameter is shown in upper case in the table. For example, COPies= indicates that you can specify any of COPIES=, COPIE=, COPI=, or COP=.

The FLAGS1 and FLAGS2 Fields

FLAGS1 and **FLAGS2** are fields of the SVC 99 dynamic allocation call. With CLIO/S Version 2 Release 2, you can set these fields. To set **FLAGS1**, use **FLAGS1**=xxxx, where xxxx is 4 hex digits. To set **FLAGS2**, use **FLAGS2**=xxxxxxxx, where xxxx is 8 hex digits. You must code the correct number of digits, that is, you cannot drop off trailing zeros.

Here is an example for the CLPLINK command:

```
clplink -h mvsnode -x mvsfile -w aixfile -O logfile \
        -S "alloc='unit=ctape sl ca new flags2=10000000'"
```

This setting has the effect of requesting that MVS wait until a tape device is available instead of cancelling the job if no device is available. Note that for this setting the MVS process must run in authorized state.

See MVS dynamic allocation documentation for more information on the possible **FLAGS1** and **FLAGS2** settings.

Other Allocation Parameters

Other allocation parameters are eventually translated into SVC 99 text units.

Table 11 (Page 1 of 5). MVS Device Allocation Parameters and CLIO/S Equivalents

DD Statement Parameter	CLIO/S Parameter	Parameter Values
COPIES=*num*	COPies=*num*	
DATACLAS=*class*	DATACLAS=*class*	
DCB=(*.*ddname*)	DCBDD=*ddname*	
DCB=(*dsname*)	DCBDS=*dsname*	
DCB=(BLKSIZE=*num*)	BLKSize=*num*	
DCB=(BUFALN=*key*)	BUFAln=*key*	*key*= D|F
DCB=(BUFIN=*num*)	BUFIN=*num*	
DCB=(BUFL=*num*)	BUFL=*num*	
DCB=(BUFMAX=*num*)	BUFMAX=*num*	
DCB=(BUFNO=*num*)	BUFNo=*num*	

© Copyright IBM Corp. 1993, 1995, 1996

Table 11 (Page 2 of 5). MVS Device Allocation Parameters and CLIO/S Equivalents

DD Statement Parameter	CLIO/S Parameter	Parameter Values
DCB=(BUFOFF=*num*)	BUFOFf=*num*	
DCB=(BUFOUT=*num*)	BUFOUt=*num*	
DCB=(BUFRQ=*num*)	BUFRq=*num*	
DCB=(BUFSIZE=*num*)	BUFSize=*num*	
DCB=(BUFTEK=*key*)	BUFTEK=*key*	*key*=A|E|R|S
DCB=(CODE=*key*)	CODe=*key*	*key*=A|B|C|F|I|N|T
DCB=(DEN=2)	D800	
DCB=(DEN=3)	D1600	
DCB=(DEN=4)	D6250	
DCB=(DIAGNS=TRACE)	TRAce	
DCB=(DSORG=*key*)	DSORG=*key*	*key*=CX|DA|DAU|GS|PO| PS|PSU|TQ|TX|TCAM|VSAM
DCB=(EROPT=ABE)	ABE	
DCB=(EROPT=ACC)	ACC	
DCB=(EROPT=SKP)	SKp	
DCB=(KEYLEN=*num*)	KEYlen=*num*	
DCB=(LIMCT=*num*)	LImct=*num*	
DCB=(LRECL=*num*)	LRec=*num*	
DCB=(MODE=*key*)	MODE=*key*	*key*=C|CO|CR|E|EQ|ER
DCB=(NCP=*num*)	NCP=*num*	
DCB=(OPTCD=*key*)	OPTCD=*key*	See *MVS/ESA JCL User's Guide* for a complete list of the OPTCD key values.
DCB=(PRTSP=*key*)	PRTsp=*key*	*key*=0|1|2|3
DCB=(RECFM=*key*)	RECFM=*key*	The RECFM values for fixed, undefined, and variable record formats are shown below. Additional RECFM values are listed in *MVS/ESA JCL User's Guide*.
DCB=(RECFM=F)	F	
DCB=(RECFM=FA)	FA	
DCB=(RECFM=FAS)	FAS	
DCB=(RECFM=FB)	FB	
DCB=(RECFM=FBA)	FBA	
DCB=(RECFM=FBAS)	FBAS	
DCB=(RECFM=FBM)	FBM	
DCB=(RECFM=FBS)	FBMS	DCB=(RECFM=FM)
	FM	
DCB=(RECFM=FMS)	FMS	
DCB=(RECFM=FS)	FS	

Table 11 (Page 3 of 5). MVS Device Allocation Parameters and CLIO/S Equivalents

DD Statement Parameter	CLIO/S Parameter	Parameter Values
DCB=(RECFM=U)	U	
DCB=(RECFM=V)	V	
DCB=(RECFM=VA)	VA	
DCB=(RECFM=VAS)	VAS	
DCB=(RECFM=VB)	VB	
DCB=(RECFM=VBA)	VBA	
DCB=(RECFM=VBAS)	VBAS	
DCB=(RECFM=VBM)	VBM	
DCB=(RECFM=VBMS)	VBMS	
DCB=(RECFM=VBS)	VBS	
DCB=(RECFM=VM)	VM	
DCB=(RECFM=VMS)	VMS	
DCB=(RECFM=VS)	VS	
DCB=(STACK=*num*)	STACK=*num*	
DCB=(TRTCH=*key*)	TRTch=*key*	*key*=C|E|ET|T
DISP=(MOD)	MOD	
DISP=(NEW)	NEW	
DISP=(OLD)	OLD	
DISP=(SHR)	SHr	
DISP=(,CATLG)	CAtlg	
DISP=(,DELETE)	DElete	
DISP=(,KEEP)	KEEp	
DISP=(,UNCATLG)	UNCatlg	
DISP=(,,CATLG)	CCatlg	
DISP=(,,DELETE)	CDelete	
DISP=(,,KEEP)	CKeep	
DISP=(,,UNCATLG)	CUncatlg	
DSN=...(*name*)	MEmber=*name*	
DSN=*dsname*	DSN=*dsname*	
DUMMY	DUMMY	
FCB=(*name*)	FORms=*name*	
FCB=(,ALIGN)	ALIgn	
FCB=(,VERIFY)	VERIFYF	
FREE=CLOSE	CLose	
HOLD=YES	Hold	
LABEL=(*num*)	DSSeq=*num*	
LABEL=(EXPDT=*yyddd*)	Expiry=*yyddd*	
LABEL=(RETPD=*num*)	RETention=*num*	
LABEL=(,AL)	AL	

Table 11 (Page 4 of 5). MVS Device Allocation Parameters and CLIO/S Equivalents

DD Statement Parameter	CLIO/S Parameter	Parameter Values
LABEL=(,AUL)	AUL	
LABEL=(,BLP)	BLP	
LABEL=(,LTM)	LTM	
LABEL=(,NL)	NL	
LABEL=(,NSL)	NSL	
LABEL=(,SL)	SL	
LABEL=(,SUL)	SUL	
LABEL=(,,IN)	Input	
LABEL=(,,NOPWREAD)	PASSWRite	
LABEL=(,,OUT)	OUTput	
LABEL=(,,PASSWORD)	PASSRead	
MGMTCLAS=*class*	MGMTCLAS=*class*	
MSVGP=*name*	MSVGP=*name*	
OUTLIM=*num*	OUTLim=*num*	
QNAME=*name*	QNAME=*name*	
SPACE=(*num*)	BLOck=*num*	
SPACE=(CYL)	CYL	
SPACE=(TRK)	TRK	
SPACE=(,(*num*))	PRIMary=*num*	
SPACE=(,(,*num*))	SECondary=*num*	
SPACE=(,(,,*num*))	DIRectory=*num*	
SPACE=(,,RLSE)	RLse	
SPACE=(,,,ALX)	ALX	
SPACE=(,,,CONTIG)	CONtig	
SPACE=(,,,MXIG)	MXIG	
SPACE=(,,,,ROUND)	ROund	
STORCLASS=*class*	STORCLAS=*class*	
SUBSYS=*name*	SUBSYS=*name*	Specifies a BatchPipes/MVS subsystem name
SYSOUT=*name*	SYSOUt=*name*	
SYSOUT=(,*name*)	SYSOUProg=*name*	
SYSOUT=(,,*name*)	SYSOUForms=*name*	
TERM=TS	TErmfile	
UCS=(,FOLD)	FOLdmode	
UCS=(,,VERIFY)	VERIFYC	
UNIT=*name*	UNIT=*name*	
UNIT=(,*num*)	UNITCount=*num*	
UNIT=(,P)	PARallel	
VOL=(,,*num*)	VOLSeq=*num*	

Table 11 (Page 5 of 5). MVS Device Allocation Parameters and CLIO/S Equivalents

DD Statement Parameter	CLIO/S Parameter	Parameter Values
VOL=(,,,*num*)	VOLCount=*num*	
VOL=(,,,,REF=*name*)	VOLRef=*name*	
VOL=(,,,,SER=(*name*))	VOLume=*name*	
VOL=(PRIVATE)	PRIVate	

Table 12. Miscellaneous Parameters

DD Statement Parameter	CLIO/S Parameter	Notes
DDNAME on DD Statement	DD=*name*	
PASSWORD	PASSWORD=*password*	
/*ROUTE *destination*	REMOTE=*destination*	
Assign the permanently allocated attribute to this resource	PERManent	For more information about the permanently allocated attribute, see *MVS/ESA Programming: Authorized Assembler Services Guide*.
Assign the convertible attribute to this resource	CONVert	For more information about the convertible attribute, see *MVS/ESA Programming: Authorized Assembler Services Guide*.
Turn on IDRC data compaction	COMP	CLIO/S-specific parameter.
Turn off IDRC data compaction	NOCOMP	CLIO/S-specific parameter.
Ensure that CLIO/S gives control back to the application when hitting End of tape on input or output	SUP_EOV	CLIO/S-specific parameter. If SUP_EOV is not specified, CLIO/S will automatically do end of volume processing when the user specifies a dataset name but no volser. (See "Effects of MVS Tape Management on CLIO/S" on page 241 for more information.)

Appendix F. IOCP, SMIT, TCP/IP, and CLIO/S Cross-reference

This appendix shows the relationships between various configuration items required to create ESCON connections between MVS and AIX. The requirements for both CLIO/S and TCP/IP are shown, although either will work without the other. (CLIO/S requires a TCP/IP connection between AIX and MVS, but that connection does not have be over the ESCON link.)

The sample configuration

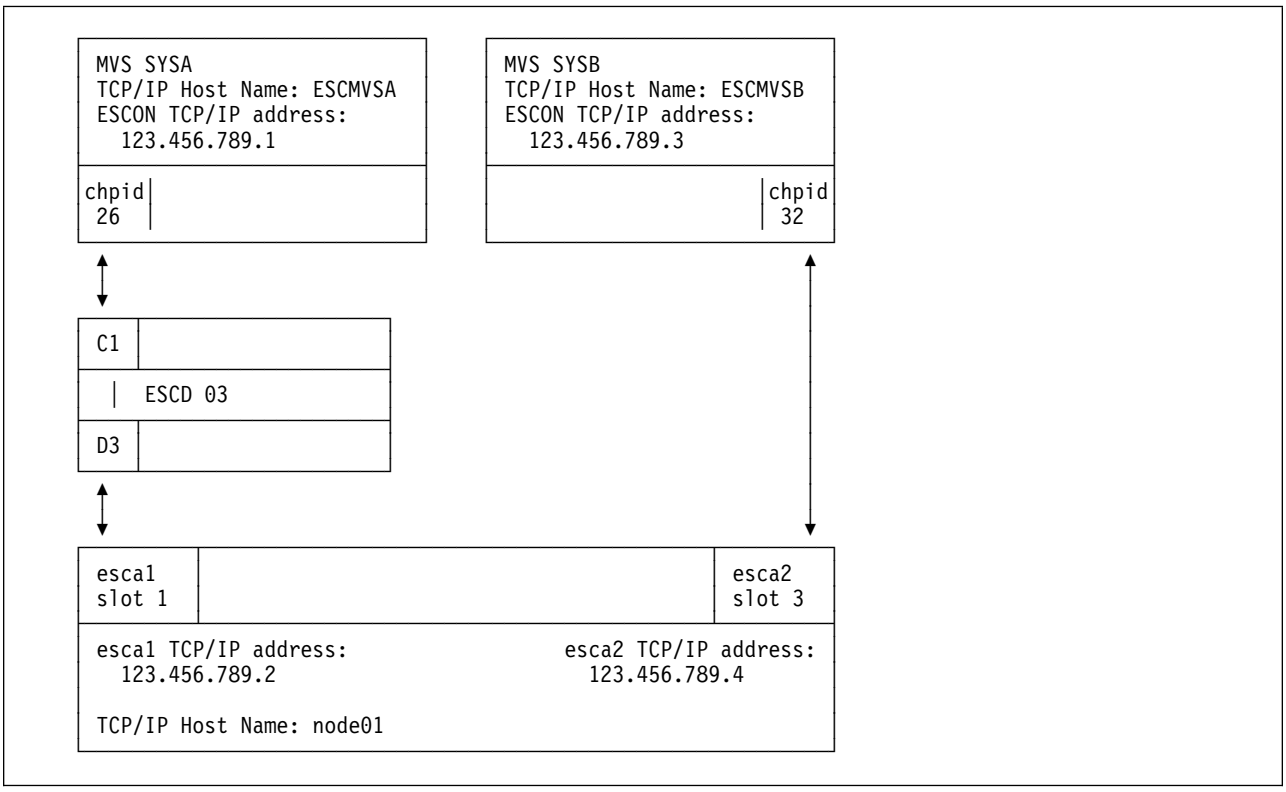


Figure 50. Configuration used for this section

The sample configuration files and displays

The required configuration data is shown here. For SMIT, display panels rather than input panels are shown for conciseness. The mapping from the display panels to the input panels is straightforward.

The reverse key (**a**) tags are used to show data that is referenced in more than one place. The next section contains discussion of the tagged items.

Sample IOCP and MVS Configuration Program statements for MVS A

© Copyright IBM Corp. 1993, 1995, 1996

```
CHPID  PATH=((26)),TYPE=CNC,SWITCH=03
CNTLUNIT CUNUMBER=226,PATH=26,UNIT=RS6K,UNITADD=((C0,32)),LINK=D3
                  a              b             c
IODEVICE UNIT=RS6K,ADDRESS=(6C0,32),CUNUMBER=226
             b            d               a
UNITNAME NAME=RS6K1,UNIT=((6C0,32))
             e           d
```

Sample IOCP and MVS Configuration Program statements for MVS B

```
CHPID  PATH=((32)),TYPE=CNC
CNTLUNIT CUNUMBER=404,PATH=32,UNIT=RS6K,UNITADD=((C0,32))
                  a              b             c
IODEVICE UNIT=RS6K,ADDRESS=(6C0,32),CUNUMBER=404
             b            d               a
UNITNAME NAME=RS6K1,UNIT=((6C0,32))
             e           d
```

Sample TCPIP.PROFILE statements for MVS A

```
DEVICE RS6K6C0 CLAW  6C0  MVSA     AIX0     NONE  40 40 4096 4096
       f            d   g        h
LINK  LINK6C0  IP  0  RS6K6C0
      i              f
HOME
   123.456.789.1   LINK6C0
                  i
GATEWAY
   123.456.789.2   =    LINK6C0  4096  HOST
                       i
START RS6K6C0
      f
```

Sample TCPIP.PROFILE statements for MVS B

```
DEVICE RS6K6C0 CLAW  6C0  MVSB     AIX1     NONE  40 40 4096 4096
       f             d   g        h
LINK  LINK6C0  IP  0  RS6K6C0
      i              f
HOME
   123.456.789.3   LINK6C0
                   i
GATEWAY
   123.456.789.4   =    LINK6C0  4096  HOST
                        i
START RS6K6C0
      f
```

Sample TCPIP.DATA statements for MVS A and B

```
SYSA: HostName escmvsa
                j
SYSB: HostName escmvsb
                k
```

Sample HOSTS.LOCAL statements for MVS A

```
HOST : 123.456.789.1 : escmvsa ::::
                       j
HOST : 123.456.789.2 : escaix0 ::::
                       k
```

Sample HOSTS.LOCAL statements for MVS B

```
HOST : 123.456.789.3 : escmvsb ::::
                       j
HOST : 123.456.789.4 : escaix1 ::::
                       k
```

Sample SMIT 'Show Defined Subchannels' data (partial)

```
 Subchannel        System Name     Address
 Set Name        Local    Remote   Lc Rm Gr Type     Path
 ------------------------------------------------------
 es0             ESCAIX0  ESCMVSA  c0 00 2  CLAW     c10
 l               h        g        m  c  n  o        p
 6c2                               c2 02 1  3088     c10
 l                                 m  c  n  o        p
 6c3                               c3 03 1  3088     c10
 6c4                               c4 04 1  3088     c10
 6c5                               c5 05 1  3088     c10
 6c6                               c6 06 1  3088     c10
 6c7                               c7 07 1  3088     c10
 6c8                               c8 08 1  3088     c10
 6c9                               c9 09 1  3088     c10
 6ca                               ca 0a 1  3088     c10
 6cb                               cb 0b 1  3088     c10
 es1             ESCAIX1  ESCMVSB  b0 00 2  CLAW     010
 l               h        g        m  c              p
 bc2                               b2 02 1  3088     010
 l                                 m  c              p
 bc3                               b3 03 1  3088     010
 bc4                               b4 04 1  3088     010
 bc5                               b5 05 1  3088     010
 bc6                               b6 06 1  3088     010
 bc7                               b7 07 1  3088     010
 bc8                               b8 08 1  3088     010
 bc9                               b9 09 1  3088     010
 bca                               ba 0a 1  3088     010
 bcb                               bb 0b 1  3088     010
```

Sample SMIT 'Show Defined Fibre' data

```
Fibre            Slot   Subchannel
Label            Number Sets
-----------------------------------------------------
cable1           1      es0,6c2,6c3,6c4,6c5,6c6,6c7,6c8,6c9,6ca,6cb
                        l
cable2           3      es1,bc2,bc3,bc4,bc5,bc6,bc7,bc8,bc9,bca,bcb
                        l
```

Sample SMIT 'Show Defined ESCON Network Interfaces' data

```
             Local    Internet        Remote   Internet
Name Status  System   Address         System   Address         Netmask
-------------------------------------------------------------------------
es0  up      AIX0     123.456.789.2   MVSA     123.456.789.1   255.255.255.0
 l           h        i               g        h
es1  up      AIX1     123.456.789.4   MVSB     123.456.789.3   255.255.255.0
 l           h        i               g        h
```

Sample CLIO/S /etc/parctc file

```
escmvsa  node01    /dev/esca0 1    6c2-6df
j        q                   e    c
escmvsb  node01    /dev/esca1 1    6c2-6df  Lb2
j        q                   e    c        m
```

Sample CLIO/S /etc/parconf file entries

```
escmvsa   2   4    123.456.789.1
j                  h
escmvsb   2   4    123.456.789.3
j                  h
```

Sample /etc/hosts file entry

```
987.654.321.1 node01
              q
123.456.789.1 escmvsa
              k
123.456.789.3 escmvsb
              k
```

Sample Missing Interrupt Handler from SYS1.PARMLIB(IECIOS00)

```
MIH TIME=00:00,DEV=(6C0-6DF)
                   c
```

Comments on the tagged items

a The CUNUMBER is used in the CNTLUNIT and IODEVICE IOCP macroinstructions.

b UNIT is used in the CNTLUNIT and IODEVICE IOCP macroinstructions. RS6K is the correct unit name for the latest levels of IOCP. Earlier values have included CTC and SCTC. For block mux channels, use 3088

c The unit address is referenced in the CNTLUNIT IOCP macroinstruction, in the SYS1.PARMLIB(IECIOS00) missing interrupt handler entry, and in the SMIT ESCA subchannel panels. In this sample configuration, the unit address is equal to the two rightmost digits of the device address, **d** , since there is no UNITADD parameter in the IODEVICE macroinstruction.

d The device address is referenced in the `IODEVICE` IOCP macroinstruction, in the `UNITNAME` MVS configuration program statement, in the TCPIP.PROFILE DEVICE statement, and in the CLIO/S /etc/parctc file. Note that the term 'subchannel' is often used on AIX to refer to device addresses on MVS.

e The esoteric name is referenced in the `UNITNAME` MVS configuration program statement and (in truncated form) in the /etc/parctc file. It must have the form RS6Knn, where nn is in the range 0-99 with no leading '0'. In this sample configuration the same esoteric name can be used for both units because they are on different MVS systems. If the esoteric name is missing or incorrect, CLIO/S will not be able to allocate a device/subchannel.

f The TCP/IP device name is referenced in the TCPIP.PROFILE DEVICE, LINK and START statements.

g The TCP/IP MVS host name is referenced in the TCPIP.PROFILE DEVICE statement, in the SMIT ESCA subchannel panels, in the SMIT ESCA network interfaces panels, in the CLIO/S /etc/parconf and /etc/parctc files, and possibly in the /etc/hosts file. The TCPIP.PROFILE and SMIT ESCA definitions require this to be in uppercase.

h The TCP/IP AIX host name is referenced in the TCPIP.PROFILE DEVICE and GATEWAY statements, in the SMIT ESCA subchannel panels, in the SMIT ESCA network interfaces panels and in the CLIO/S /etc/parconf and /etc/parctc files. The TCPIP.PROFILE and SMIT ESCA definitions require this to be in uppercase.

i The TCP/IP MVS link name is referenced in the TCPIP.PROFILE LINK, HOME, and GATEWAY statements.

l The subchannel set names are referenced in the SMIT ESCA subchannel and fibre panels. Note that the 'esn' format is required for the subchannel sets that are used by TCP/IP.

Note that only a portion of the full 32-address range for each adapter has been used. As configured here, no process will be able to use device addresses 6cc-6df.

m The local address is referenced in the SMIT ESCA subchannel panel and possibly in the CLIO/S /etc/parctc file. The value is arbitrary and must be unique. Commonly the local address is chosen to equal the remote (unit) address, but this is not required and sometimes the uniqueness requirement makes it impossible. CLIO/S expects the local address to be equal to the device address, which may or may not be equal to the unit address. If the local address is not equal to the device address, then the 'local' parameter must be used in the /etc/parctc entry.

n TCP/IP uses pairs of subchannels (with an even subchannel as the first of the pair), so the group value should always be 2. CLIO/S uses only one subchannel at a time, and requires that there be only one subchannel per group.

o The type is always CLAW for TCP/IP, alway 3088 for CLIO/S.

p The path refers to the ESCON Director (ESCD) port used to get back to the MVS channel. The first two characters of this value are the ESCD port number (01 is used if there is no ESCD involved). The third character is used to differentiate between different `CUADD` values, which are not used in this sample configuration.

- The host name is referenced in the /etc/parctc file and is defined in the /etc/hosts file. It is the value returned by the AIX **hostname** command, This is probably NOT the name of the ESCON adapter interface, but is commonly set to the name of the ethernet adapter.

Bibliography

- CLIO/S User's Guide and Reference
- AIX 4.1 ESCON Adapter: User's Guide and Service Information, SC31-8197
- IBM TCP/IP for MVS Customization and Administration Guide V3 R1, SC31-7134
- Enterprise System/9000 Enterprise System/3090 Input/Output Configuration Program User's Guide and ESCON Channel-to-Channel Reference, GC38-0097
- MVS Configuration Program, GC28-1615

Index

Special Characters

? subcommand of CLFTP 97
.netrc
  See ?

A

abend 768 136
about 273
  modifying 273
about the Application Linker GUI 63
accept() socket routine 167
accepting
  a CLIO/S connection from another process 173
  a TCP/IP connection 184
adapter
  sharing 43
adding a process ID to an alert group 195
ADDZEROEOF
  parameter for LOCSITE and SITE 115, 121
AL tape preformatting 247
ALERT DPI routine 194
ALERTGRP DPI routine 195
ALLOC
  parameter for LOCSITE and SITE 116, 121
allocating a CLIO/S pipe 145
allocating a CLIO/S tape device 84, 88
allocation device 303
allowing a process to change ownership of a connection 182
ALRTGRP DPI routine 195
anonymous FTP server xxi
ANSI Standard Label tape preformatting 247
APPEND subcommand of CLFTP 127
appending files
  using CLFTP 127
  using CLPLINK 149
Application Linker
  about the GUI 63
  changing a configuration 74
  clalink command 67
  complex example 69
  Menu Bar 64
  message boxes 66
  notebooks 66
  primary window 63
  running large configurations 79
  sample configurations 67
  simple example 68
  Start After IDs 67
  starting 67
Application Linker *(continued)*
  Template Area 65
  Tool Bar 64
  using 66
  Work Area 65
Application Linker GUI
  concepts 58
  configuration file 60
  learning to use 58
  model validation 62
  object IDs 62
  overview 57
  partial saves and loads 60
  password encryption 61
  Start After IDs 62
  system authorization 61
  terminology 58
  why use? 57
ASCII subcommand of CLFTP 111
audience of this book xvii

B

batch submission of CLFTP 133
BINARY subcommand of CLFTP 112
BLKWRITE
  parameter for LOCSITE and SITE 121
BLocks
  parameter for LOCSITE and SITE 116
BLOCKSIZE
  parameter for LOCSITE and SITE 116, 121
BMCA
  DPI support of 190
BMCA/PCA
  See BMCA
BMCA/PCA subchannels
  starting 275
buffer sizes
  receive 203
  send 203
buffering mechanism, changing 220

C

CANCEL function (MVS) 221
cancelling a CLIO/S job
  about 264
  with FCFCAN authorized program 264
  with FCFCANC REXX EXEC 265
CD subcommand of CLFTP 102
cdpi_async DPI routine 195

© Copyright IBM Corp. 1993, 1995, 1996

cdpi_check DPI routine 196
cdpi_s2c DPI routine 197
CDPIASY DPI routine 195
CDPICHK DPI routine 196
chained channel programs 213, 222
changing
  send and receive buffer sizes 203
  the CLIO/S tape pseudo device driver 49
changing defaults for environment variables 41
checking
  number of bytes transmitted by a group 196
  the number of bytes transmitted 174
CHPID IOCDS statement 19
clalink 67
CLAW 43
CLAW applications, sharing adapter with 43
clearing buffers written using SPANY 197
CLFLUSH DPI routine 197
CLFTP
  controlling termination of error 112
  customizing 95
  debugging 139
  deleting files on remote host 128
  internal error codes 136
  list of subcommands 96, 97
  list of subcommands by task 95
  messages 136
  overview 91
  return codes 135
  specifying user ID and password 93
  starting as a batch job 133
  terminating on error 112
  using 91
  using with multiple PARSRVs 91
CLFTP subcommands
  ? 97
  APPEND 127
  ASCII 111
  BINARY 112
  CD 102
  DELETE 128
  displaying file transfer status messages 98
  ECHO 98
  echoing subcommands 98
  ENDONERROR 112
  file transfer status messages, displaying 98
  GET 129
  HASH 98
  HELP 97
  LCD 107
  LLS 108
  LMKDIR 109
  LOCSTAT 99
  LPWD 111
  LS 104
  MKDIR 104
CLFTP subcommands *(continued)*
  PUT 131
  PWD 106
  QUIT 132
  SITE 114
  STATUS 100
  TRANSLATE 126
CLFTPINF file 94
CLFTPINPUT environment variable 37
CLFTPOUTPUT environment variable 37
clio_accept CMI routine 173
clio_async CMI routine 173
clio_check CMI routine 174
clio_close CMI routine 174
clio_connect CMI routine 175
clio_create CMI routine 175
clio_end CMI routine 176
clio_env CMI routine 176
clio_exp CMI routine 177
clio_h2i CMI routine 178
clio_init CMI routine 178
clio_kill CMI routine 178
clio_l2n
  CMI routine 179
  DPI routine 198
clio_m2i CMI routine 180
clio_map CMI routine 180
clio_n2h CMI routine 181
clio_n2i CMI routine 181
clio_n2l
  CMI routine 179
  DPI routine 199
clio_n2m CMI routine 182
clio_own CMI routine 182
clio_pid CMI routine 183
clio_read CMI routine 183
clio_t CMI structure 199
clio_write CMI routine 184
CLIO/S
  configuration files 25
  configuring 17
  conversion 259
  I/O 251
  man pages xx
  networking 17
  overview 1
  sockets programming interface 165
CLIO/S Cross System Pipe Link
  See also CLPLINK
  about 141
  accessing from files from an application 144
  clplink 148
  clpmk 145
  connecting pipes between remote hosts 145
  connecting to a CLIO/S pipe 147
  copying files to and from remote hosts 144

CLIO/S Cross System Pipe Link *(continued)*
creating a named pipe 145
CLIO/S FTP
See CLFTP
CLIOADEB environment variable 37
CLIOARB environment variable 37
CLIOBIN environment variable 37
CLIOBLK DPI routine 200
CLIOBLK environment variable 37
CLIOBUF DPI routine 200
CLIOBUF environment variable 37
CLIODEB DPI routine 201
CLIODEB environment variable 37
CLIODFILE environment variable 38
CLIOEFILE environment variable 38
CLIOENV DPI routine 202
CLIOMST DPI routine 202
CLIONODE environment variable 38
CLIOPATH environment variable 38
CLIOPSID environment variable 39
CLIOPSPT 91
CLIOPSPT environment variable 39
CLIOSHELL environment variable 39
CLIOSOCK DPI routine 203
cliostae 221
CLIOSTAE environment variable 222
CLIOSTRIPES environment variable 39
CLIOSYS environment variable 39
CLIOTBS DPI routine 203
CLIOTBS environment variable 39
CLIOTMO DPI routine 204
CLIOTMO environment variable 39
cliotrce 221
close() socket routine 167
closing
a CLIO/S connection 174
the DPI environment 204
the tape 230
CLPLINK
customizing 161
JCL for starting 267
using 148
clpmk 145
CLSFAM DPI routine 204
CLSPAR DPI routine 204
CLSSRV DPI routine 205
cltalloc 84, 88
cltclntd 53
cltterm 54
cltunall 89
CMI interface
about 171
clio_accept 173
clio_async 173
clio_check 174
clio_close 174
CMI interface *(continued)*
clio_connect 175
clio_create 175
clio_end 176
clio_env 176
clio_exp 177
clio_h2i 178
clio_init 178
clio_kill 178
clio_m2i 180
clio_map 180
clio_n2h 181
clio_n2i 181
clio_n2l 179
clio_n2m 182
clio_read 183
clio_write 184
tcp_accept 184
tcp_connect 185
codes
return, CLFTP 135
commands
clalink 67
clplink 148
clpmk 145
cltalloc 84, 88
cltclntd 52
cltterm 49, 54
cltunall 89
Communications Manager Interface 171
compiling the tape interface on MVS 242
compiling the tape interface on RS/6000 242
configuration files, creating and customizing 25
configuring CLIO/S 17, 25
configuring CLIO/S for striping 42
configuring CTCs on AIX 21
configuring CTCs on MVS 19
configuring the tape mount daemon
procedures for 46
with intermediate systems 47
confirming data was sent
from master to servant 207
from servant to master 216
connect() socket routine 167
connecting to a CLIO/S named pipe 148
CONV
parameter for LOCSITE and SITE 116, 122
conversion
CLIO/S 259
converting data
representations 179
to local format 199
to network format 198
converting data with CLFTP 126
copying files
from a remote host using CLFTP 129

copying files *(continued)*
from anonymous FTP server xxi
to a remote host using CLFTP 131
counting valid entries in PARCONF 208
cptexit 240
creating
local directory with CLFTP 109
remote directory with CLFTP 104
cross-reference connections for MVS to AIX 309
CTLUNIT IOCDS statement 19
ctpalloc tape interface routine 227
ctpclose tape interface routine 230
ctpopen tape interface routine 231
ctpread tape interface routine 232
ctprew tape interface routine 233
ctpskpf tape interface routine 233
ctpskpr tape interface routine 234
ctpunall tape interface routine 235
ctpvols tape interface routine 235
ctpweof tape interface routine 236
ctpwrite tape interface routine 237
ctpwrto tape interface routine 237
customizing
CLFTP 95
CLPLINK 161
JCL generation by PARSRV 279
CYLINDERS
parameter for LOCSITE and SITE 116

D

data conversion
CMI 171
DPI 187, 200, 206, 207, 216
data objects 59
data translation for CLFTP 126
data types
CMI 171
DPI 187
DCBDSN
parameter for LOCSITE and SITE 117
DD INPUT and OUTPUT 95
debug flags
setting 268
debugging
CLFTP 139
decr_pw 41
defaults
changing for PARSRV 41
establishing for PARSRV 41
DELETE subcommand of CLFTP 128
deleting files
from a remote host using CLFTP 128
device allocation 303
dialog boxes 66
directory
APPEND parameter 127
CD parameter 102
changing on remote MVS host 102
changing the directory on MVS 108
changing the directory on remote host 102
establishing for CLFTP 94
GET parameter 129
LCD parameter 107
local MVS host 108
LPWD parameter 111
MKDIR parameter 104
MVS CLFTP server 108
parameter for LOCSITE and SITE 117
PUT parameter 131
remote working directory 106
the local working directory 107
displaying
current working-level qualifier 111
file transfer status messages in CLFTP 98
local host status information for CLFTP 99
remote status information for CLFTP 100
the current remote working directory for CLFTP 106
displaying server information 268
Distributed Processing Interface
See DPI
documentation xix
downloading files from anonymous FTP server xxi
DPI
See also DPI C interface
See also DPI FORTRAN interface
about 190
sample master and servant code 192
using 187
DPI C interface
ALERT 194
ALERTGRP 195
cdpi_async 195
cdpi_check 196
cdpi_s2c 197
CLFLUSH 197
CLIOBLK 200
CLIOENV 202
CLIOSOCK 203
CLIOTBS 203
CLSFAM 204
CLSPAR 204
CLSSRV 205
FNDMACH 205
MS_flush 207
MS_xxxx (macros) 206
MSANY 206
MSxxxx 206
NMACHS 208
OPNFAM 208
OPNPAR 209

DPI C interface *(continued)*
OPNSRV 210
QMACH 211
QSLAVE 211
RP_xxxx (macros) 211
RPANY 211
rw_dpi 213
SCHEDUL 213
SETFAM 214
SETSRV 215
SM_flush 216
SM_xxxx 216
SMANY 215
SP_xxxx 218
SPANY 217
SYNCHRO 218
TSKEXIT 219
WAITFOR 219
WAITREQ 220
DPI FORTRAN interface
ALERT 194
ALRTGRP 195
CDPIASY 195
CDPICHK 196
CLFLUSH 197
CLIOBLK 200
CLIOBUF 200
CLIODEB 201
CLIOENV 202
CLIOMST 202
CLIOTMO 204
CLSFAM 204
CLSPAR 204
CLSSRV 205
FNDMACH 205
LNxxxx 198
MSFLSH 207
MSxxxx 206
NLxxxx 199
NMACHS 208
OPNFAM 208
OPNPAR 209
OPNSRV 210
RPxxxx 211
SCHEDUL 213
SETFAM 214
SETSRV 215
SM_xxxx (macros) 215
SMFLSH 216
SMxxxx 215
SP_xxxx (macros) 217
SPxxxx 217
SYNCHRO 218
WAITFOR 219
WAITREQ 220
DPI low level interface
rw_dpi 223
dynamic allocation
for CLPLINK and CLFTP 116, 121
for tape server 227
list of parameters 303

E

ECHO subcommand of CLFTP 98
encr_pw 41
ending a CLIO/S connection 176
ENDOFFSET
parameter for LOCSITE and SITE 117, 122
ENDONERROR subcommand of CLFTP 112
ENDRECNUM
parameter for LOCSITE and SITE 117, 122
ENDVLIST
parameter for LOCSITE and SITE 117, 122
environment variables
changing default values 41
CLFTP input 95
CLFTP output. 95
CLFTPINPUT 37
CLFTPOUTPUT 37
CLIOADEB 37
CLIOARB 37
CLIOBIN 37
CLIOBLK 37
CLIOBUF 37
CLIODEB 37
CLIODFILE 38
CLIOEFILE 38
CLIONODE 38
CLIOPATH 38
CLIOPSID 39
CLIOPSPT 39
CLIOSHELL 39
CLIOSTRIPES 39
CLIOSYS 39
CLIOTBS 39
CLIOTMO 39
list of reserved names 40
MVS characteristics for *user_ID*.CLIOENV 36
setting 36
TAPEMSG 38
error codes
internal, CLFTP 136
ESCON
DPI support of 190
ESCON cross-reference connections 309
esoteric names 19
establishing a TCP/IP connection 185
ESTAE support 221
example, tape interface 225

F

FCFCAN 264
FCFCANC 264
FCFCANC REXX EXEC 265
fcfcmi.h header file 171, 187
FCFCONF 41
fcfconf.c 41
fcfdpi.h header file 187
FCFFCLPR
  about 266
  JCL for starting 267
FCFFCLPW
  about 266
  JCL for starting 267
FCFFINDP 41
fcffindp.c 41
FCFFPIPR
  about 272
  JCL for starting 272
FCFFPIPW
  about 272
  JCL for starting 273
FCFFSERV 91
FCFIDUM 265
FCFPARDE 268
FCFPARIN 268
FCFPARPI 269
FCFPARSR
  See PARSRV
FCFPARST 271
FCFREXX
  See also JCL generation, REXX
  sample program 284
FCFSHELL 299
FCFTPX00 240
FCFTTEST sample program 242
FCFVARY
  using 275
file transfer types
  and CLFTP 111, 112
  and CLPLINK 148, 149
find_pw 41
finding related publications xxi
FNDMACH DPI routine 205
forcing data transfer to the servant ID 215
fs_accpt() socket routine 167
fs_close() socket routine 167
fs_conn() socket routine 167
fs_read() socket routine 167
fs_selct() socket routine 167
fs_shutd() socket routine 167
fs_write() socket routine 167

G

GDG data sets
  specifying tape allocation parameters 228
generating JCL
  See JCL generation
GET subcommand of CLFTP 129
getsockopt() 167
getting
  a list of tape volumes 235
  an Internet address 181
  HELP for CLFTP 97
  machine information 211
  servant machine information 211
  the environment 176
  the environment file 202
  the machine information ID 182

H

hardware, list of supported see CLIO/S *General Information Manual*
HASH subcommand of CLFTP 98
header file
  fcfcmi.h 171
help
  online (AIX man pages) xx
HELP subcommand of CLFTP 97
host
  disconnect (CLFTP) 132
  local status information for CLFTP 99
  remote 102
  remote status information 100
how to use this book xvii

I

I/O
  CLIO/S 251
indicating a servant is waiting for work 220
informing a master the servant process is finished 219
initializing
  CLIO/S 178
  the DPI environment 209
intermediate nodes
  CLPLINK example 158, 159
  configuring tape mount daemon for 47
  PARCONF entries for 26
  using CLFTP 92, 133
  using CLPLINK 150
  using tape API 228
IO
  parameter for LOCSITE and SITE 117, 122
IOCP generation
  esoteric names required for 19
  IOCDS statements 19

IODEVICE IOCDS statement 19

J

JCL generation
  customizing language for PARSVR 40
  migrating to CLIO/S Version 2 Release 1 279
  pseudo-JCL
  REXX
    debugging with TRACE and SAY 282
    generating JCL 280
    implementation of 282
    input to 280
    sample FCFREXX program 282
    SAY 282
    services available 280
    setting return code 282
    TRACE 282

L

large-volume transmission 165
LCD subcommand of CLFTP 107
LENFIELDS
  parameter for LOCSITE and SITE 117, 122
LENFIELDSADD
  parameter for LOCSITE and SITE 117, 122
LENFIELDSSUB
  parameter for LOCSITE and SITE 117, 122
limitations of sockets 168
linking the tape interface on MVS 242
linking the tape interface on RS/6000 242
listing files in local directory with CLFTP
listing files in remote directory with CLFTP
LLS subcommand of CLFTP 108
LMKDIR subcommand of CLFTP 109
LNxxxx DPI routine 198
local host
  creating directory using CLFTP 109
  listing files in directory using CLFTP 108
  specifying with CLIONODE 38
  status information 99
locating
  a machine by symbolic name 181, 205
  machine information using a machine number 180
  machine information using an Internet address 178
LOCSITE
  list of parameters for CLFTP 115
  subcommand of CLFTP 114
LOCSTAT subcommand of CLFTP 99
logging off CLFTP 132
LPWD subcommand of CLFTP 111
LRECL
  parameter for LOCSITE and SITE 118
LS subcommand of CLFTP 104

M

man pages xx
masking printed status messages 201
master/servant model 189
MAXBLOCKSIZE
  parameter for LOCSITE and SITE 122
MAXLRECL
  parameter for LOCSITE and SITE 118, 123
messages
  CLFTP 136
migrating
  FCFDEFS 279
  JCL generation programs 279
missing interrupt handler entry 20
MKDIR subcommand of CLFTP 104
monitoring
  tape mount daemon activity 48
MOUNT
  parameter for LOCSITE and SITE 123
MS_flush DPI routine 207
MS_xxxx DPI macros 206
MSANY DPI routine 206
MSFLSH DPI routine 207
MSxxxx DPI routine 206
multivolume tape support 241
MVS tape management 241
MVS UNITNAME 19
MVS user abend 768 136

N

netrc
  and CLFTP 93
  and CLPLINK 153
  bypassing for all intermediate CLFTP hosts 135
  bypassing for some intermediate CLFTP hosts 135
  identifying yourself to CLFTP 93
  specifying the downstream host for CLFTP 134
netrc file
  creating 34
networking CLIO/S 17
NLxxxx DPI routine 199
NMACHS DPI routine 208
NOADDZEROEOF
  parameter for LOCSITE and SITE 115, 121
NOBLKWRITE
  parameter for LOCSITE and SITE 121
node hopping 28, 61
NOLENFIELDS
  parameter for LOCSITE and SITE 117, 122
NOLENFIELDSADD
  parameter for LOCSITE and SITE 117, 122
NOLENFIELDSSUB
  parameter for LOCSITE and SITE 118, 122

non-CLIO/S applications, sharing adapters with 43
NONULSTOBLANKS
  parameter for LOCSITE and SITE 118, 123
NORAWWRITE
  parameter for LOCSITE and SITE 123
NORDW
  parameter for LOCSITE and SITE 118, 123
NORDWADD
  parameter for LOCSITE and SITE 118, 123
NORDWSUB
  parameter for LOCSITE and SITE 118, 123
NORECIO
  parameter for LOCSITE and SITE 119, 124
NOSTRIPBLANKS
  parameter for LOCSITE and SITE 119, 124
notation, syntax xxii
NOZEROEOF
  parameter for LOCSITE and SITE 120, 125
NULSTOBLANKS
  parameter for LOCSITE and SITE 118, 123

O

objectives for this manual xvii
OFFSET
  parameter for LOCSITE and SITE 118, 123
online 22, 275
online help (AIX man pages) xx
opening
  a family of processes 208
  the tape device for use 231
OPNFAM DPI routine 208
OPNPAR DPI routine 209
OPNSRV DPI routine 210
organization of this book xvii
overview of CLIO/S 1

P

parallel channel adapter (PCA)
  See BMCA
PARCONF file 26
PARCTC file 32
pardeb 268
parinfo 268
parping 269
PARSRV
  about 34
  customizing JCL for jobs started by 279
  customizing JCL generation language 40
  generating JCL for jobs started by 279
  JCL for starting 270, 271
  multiple instances i, 39, 41, 46, 50, 51, 91, 143, 150, 154, 279
    CLPLINK 155
  port number 35
PARSRV *(continued)*
  starting 35, 40, 269
  stopping 271
  testing whether running 269
  used by CLIO/S FTP 91
  used by DPI 190
  where it looks for servant programs 40
parstop 271
password
  supplying for multiple hosts for CLFTP 134
password security 41
PCA
  See BMCA
performance hints, socket 169
piping data between remote hosts
port number
  See PARSRV
prerequisites see CLIO/S *General Information Manual*
PRIMARY
  parameter for LOCSITE and SITE 118
Program Directory for CLIO/S 17
pseudo-JCL
  See JCL generation, pseudo-JCL
PUT subcommand of CLFTP 131
PWD subcommand of CLFTP 106

Q

QMACH DPI routine 211
QSLAVE DPI routine 211
QUIT subcommand of CLFTP 132

R

RAWWRITE
  parameter for LOCSITE and SITE 123
RDW
  parameter for LOCSITE and SITE 118, 123
RDWADD
  parameter for LOCSITE and SITE 118, 123
RDWSUB
  parameter for LOCSITE and SITE 118, 123
read() socket routine 167
readers of this book xvii
reading
  data 213
  data over a CLIO/S connection 183
  multiple tape vols 241
  the tape 232
receive buffer sizes 203
receiving data from a process 211
RECFM
  parameter for LOCSITE and SITE 119
RECIO
  parameter for LOCSITE and SITE 119, 124

RECNUM
  parameter for LOCSITE and SITE 119, 124
reconfiguring
  the CLIO/S tape pseudo device driver 50
recv() socket routine 167
related publications xxi
remote host
  creating directory using CLFTP 104
  listing files in directory using CLFTP 104
  sending information 114
  status information 100
removing
  the CLIO/S tape pseudo device driver 52
requesting a CLIO/S connection to another process 175
restarting
  the CLIO/S tape mount daemon 50
  the CLIO/S tape pseudo device driver after a reboot 50
return codes 135
return values, tape 239
returning
  a CLIO/S pointer 197
  a process ID 183
  the socket descriptor 203
rewinding a tape 233
REXX
  See JCL generation, REXX
routed nickname
  See intermediate nodes
routing
  See intermediate nodes
RP_xxxx DPI macros 211
RPANY DPI routine 211
RPxxxx DPI routine 211
rw_dpi 223
rw_dpi DPI routine 213

S

samples
  about sample JCL 263
  CLFTP as a batch job 133
  DPI master and servant programs 192
  FCFCANC 264
  FCFCONF 41
  FCFFCLPR 267
  FCFFCLPW 267
  FCFFINDP 41
  FCFFPIPR 272
  FCFFPIPW 272
  FCFIDUM 265
  FCFPARDE 268
  FCFPARIN 268
  FCFPARPI 269
  FCFPARSR 269
samples *(continued)*
  FCFPARST 271
  FCFREXX 284
  FCFTTEST 242
  FCFVARY 275
  list of JCL samples 263
SCHEDUL DPI routine 213
scheduling a task to start 213
SECONDARY
  parameter for LOCSITE and SITE 119
security, password 41
select() socket routine 167
send buffer sizes 203
send() socket routine 167
sending data to a process 217
SETFAM DPI routine 214
setsockopt() 167
SETSRV DPI routine 215
setting
  an alert group 194
  an asynchronous flag by group 195
  CLFTP transfer type to ASCII 111
  CLFTP transfer type to BINARY 112
  CLIO/S environment variables 36
  debug flags for PARSRV 268
  the asynchronous data flag 173
  the buffer bypass limit 200
  the environment 177
  the family ID 214
  the size of the internal buffer 200
  the time-out period 204
sharing a CTC adapter with other applications 43
shutdown() socket routine 167
SITE
  list of parameters for CLFTP 115
  subcommand for CLFTP 114
skipping
  tape files 233
  tape records 234
SM_flush DPI routine 216
SM_xxxx DPI macros 215
SMANY DPI routine 215
SMFLSH DPI routine 216
SMxxxx DPI routine 215
socket calls 165
socket inheritance 169
socket performance hints 169
sockets programming with CLIO/S 165
sockets, limitations 168
sockets, portability 167
software, list of supported see CLIO/S *General Information Manual*
SP_xxxx DPI macros 217
SP_xxxx DPI routine 218
SPANY DPI routine 217

specifying a tape drive 84, 88
specifying local host information for CLFTP 114
specifying the tape device you want 83
SPxxxx DPI routine 217
starting
  a master 193
  a process on a node 175
  a series of reads 223
  a series of writes 223
  a servant 193
  a servant process 210
  the CLIO/S server 34, 269
  the CLIO/S tape mount daemon 53
  the remote tape server 227
status information using CLFTP 98
STATUS subcommand of CLFTP 100
stopping
  a process on a node 178
  all processes in a family 204
  tape mount daemon 54
  the CLIO/S &tmoudt. 49
  the CLIO/S server 271
  the remote tape server 89, 235
  the servant process 205
STRIDE
  parameter for LOCSITE and SITE 119, 124
STRIPBLANKS
  parameter for LOCSITE and SITE 119, 124
striping function 42
striping, configuring CLIO/S for 42
subchannels
  configuring BMCA/PCA 21
  configuring ESCON 23
  varying on and off 273
Sun RPC 189
supported hardware and software see CLIO/S *General Information Manual*
SYNCHRO DPI routine 218
syntax notation xxii
SYS1.PARMLIB(IECIOS00) entries 20
system information using CLFTP 98

T

tape device 83
tape device note 225
tape driver 83
tape exit routine 240
tape I/O 248
tape interface 225
  clalink 67
  cltalloc 84, 88
  cltterm 54
  cltunall 89
  ctpalloc C function 227
  ctpclose C function 230
tape interface *(continued)*
  ctpopen C function 231
  ctpread C function 232
  ctprew C function 233
  ctpskpf C function 233
  ctpskpr C function 234
  ctpunall C function 235
  ctpvols C function 235
  ctpweof C function 236
  ctpwrite C function 237
  ctpwrto C function 237
  FCFTPX00 240
  return values 239
  tape exit routine 240
  tp utility 246
  tpalloc FORTRAN function 227
  tpclose FORTRAN function 230
  tpopen FORTRAN function 231
  tpread FORTRAN function 232
  tprew FORTRAN function 233
  tpskpf FORTRAN function 233
  tpskpr FORTRAN function 234
  tpunall FORTRAN function 235
  tpvols FORTRAN function 235
  tpweof FORTRAN function 236
  tpwrite FORTRAN function 237
  tpwrto FORTRAN function 237
  using the tpwrto program 246
tape interface example 225
tape mount daemon 45
  monitoring 48
  restarting 50
  signaled by tape pseudo device driver 53
  stopping 49
tape pseudo device driver
  changing 49
  making available 45
  reconfiguring 50
  removing 52
  restarting after reboot 50
  signals tape mount daemon 53
  stopping 54
  unconfiguring 51
tape return values 239
TAPEMSG environment variable 38
TAPESRV 45
tcp_accept CMI routine 184
tcp_connect CMI routine 185
TCP/IP
  ensuring complete installation 18
  selecting as CLIO/S protocol 26
telling OPNSRV to start independent masters 202
testing the CLIO/S Cross System Pipe Link 272
testing whether PARSRV is running 269
toggles
  ECHO—echoing subcommands 98

toggles *(continued)*
ENDONERROR—controlling termination on error 112
HASH—controlling file transfer status messages 98
LOCSITE 114
tp tape message utility 246
tpalloc tape interface routine 227
tpclose tape interface routine 230
tpopen tape interface routine 231
tpread tape interface routine 232
tprew tape interface routine 233
tpskpf tape interface routine 233
tpskpr tape interface routine 234
tpunall tape interface routine 235
tpvols tape interface routine 235
tpweof tape interface routine 236
tpwrite tape interface routine 237
tpwrto support 246
tpwrto tape interface routine 237
TRACKS
parameter for LOCSITE and SITE 119
trademarks xv
transferring data
APPEND subcommand of CLFTP 127
DELETE subcommand of CLFTP 128
from master to servant 206
from servant to master 215
GET subcommand of CLFTP 129
PUT subcommand of CLFTP 131
transferring files
using CLFTP 91
using CLIO/S Cross System Pipe Link 144
TRANSLATE subcommand of CLFTP 126
TSKEXIT DPI routine 219

U

unallocating a CLIO/S tape device 89
unconfiguring
the CLIO/S tape pseudo device driver 51
UNIT
parameter for LOCSITE and SITE 120
UNITNAME IOCDS statement 19
user abend 768 136
user ID
for CLFTP 93
for CLPLINK 153
supplying for multiple hosts for CLFTP 134
USERdata
parameter for LOCSITE and SITE 120, 124
using CLFTP
appending files on a remote host 127
copying files from remote host 129
copying files to remote host 131
creating directories 104, 109
deleting files from remote host 128
using CLFTP *(continued)*
deleting remote data 128
disconnect from the host 132
displaying local host status information 99
displaying status information 98
establishing an AIX working directory 94
establishing an MVS working directory 94
listing files in a local directory 108
listing files in a remote directory 104
specifying site parameters 114
transferring data 127, 129, 131
transferring files 91
with multiple PARSRVs 91
using the CLIO/S striping function 42
using this book xvii
UTILITY
parameter for LOCSITE and SITE 120, 124

V

varying on and off subchannels 273
verifying CLIO/S communication 265
VLIST
parameter for LOCSITE and SITE 120, 125
VOLUME
parameter for LOCSITE and SITE 120

W

WAITFOR DPI routine 219
waiting
until all or selected tasks are completed 218
until an alert is available 219
WAITREQ DPI routine 220
working directory
displaying name of 106
on AIX for CLFTP 94
on MVS for CLFTP 94, 107, 108
working-level qualifier 107, 111
write() socket routine 167
writing
data 213
data over a CLIO/S connection 184
multiple tape vols 241
tape marks 236
to operator 237
to tape 237

Z

ZEROEOF
parameter for LOCSITE and SITE 120, 125