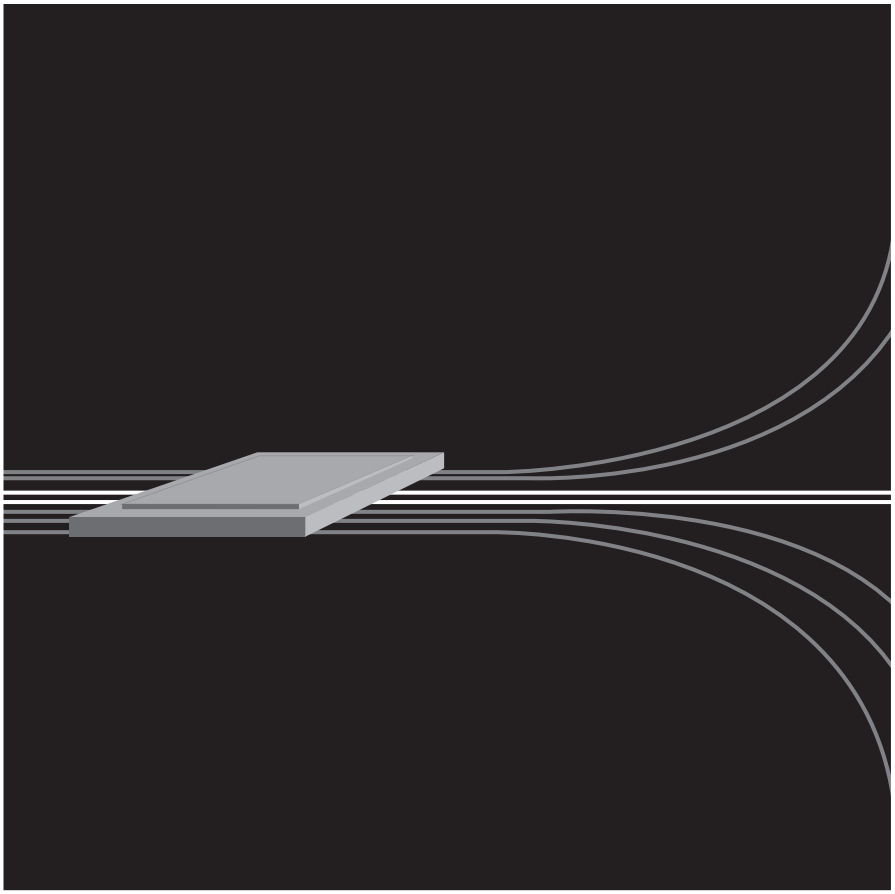
CLIO-user-01

IBM Client Input Output/Sockets

User's Guide and Reference

Version 2 Release 1.0



IBM

IBM Client Input Output/Sockets

CLIO-user-01

User's Guide and Reference

Version 2 Release 1.0

Note!

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

0 **Second Edition (May, 1995)**

This edition applies to Version 2 Release 1 of the IBM Client Input Output/Sockets (CLIO/S) Licensed Program, program number 5648-129, and to all subsequent releases and modifications until otherwise indicated in new editions. Significant changes or additions to the text and illustrations subsequent to CLIO/S Version 1 are indicated by a vertical bar (|) to the left of the change.
0 Changes to the manual added with APAR PN70092 are indicated with the number zero (0) to the left of the change. For a summary of changes, see “What's New in This Book” on page xvii. This document describes how to use CLIO/S.

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

IBM welcomes your comments. Please address your comments to:

| IBM Corporation, Department 55JA, Mail Station P384
| 522 South Road
| Poughkeepsie, New York 12601
United States of America

| FAX: Your International Access Code +1+914+432-9405
|
| IBMLink (United States customers only): D58PUBS at KGNVMC
| IBM Mail Exchange: USIB27BQ at IBMMAIL
| Internet: d58pubs@vnet.ibm.com

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

| © **Copyright IBM Corporation 1993, 1995. All rights reserved.**
Note to US Government users (documentation related to restricted rights): Use, duplication, or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corporation.

Contents

Special Notices xi
Programming Interfaces xi
Trademarks and Service Marks xi
Acknowledgement xii

About This Book xiii
Who Should Use This Book xiii
What's in This Book xiii
Where to Find Related Publications xv
General Information About This Book xvi
Special Terms xvi
Syntax Notation xvi
What's New in This Book xvii
0 Summary of Changes, May 1995 xvii
| Summary of Changes, April 1995 xvii
Summary of Changes, September 1994 xvii
Summary of Changes, May 1994 xvii

Chapter 1. Introduction 1
| An Overview of CLIO/S 1
CLIO/S's Major Functions 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
| The CLIO/S Library 7
Using CLIO/S Man Pages 8
| Downloading Files from the Anonymous FTP Server 9

| **Chapter 2. Migrating From CLIO/S Version 1** 11
| Installing CLIO/S 11
| On MVS 11
| On AIX 11
| Migrating CLIO/S Applications 12

Part 1. Configuring and Managing 13

Chapter 3. Configuring and Starting CLIO/S Networks 15
Planning CLIO/S Configuration and Startup 15
| Ensuring Complete TCP/IP Installation 17
Configuring Your Channel to Channel Adapters (CTCs) 17
Configuring CTCs On MVS 17
Configuring CTCs On AIX 19
| Creating and Customizing CLIO/S Configuration Files 23
The PARCONF File 24
The PARCTC File 26
The NETRC File 27
The CLIO/S Server—PARSRV 28

© Copyright IBM Corp. 1993, 1995

Selecting a Port Number . . . 29
Setting CLIO/S Environment Variables . . . 29
Specifying the JCL Generation Language . . . 33
Starting the PARSRV Server . . . 33
Refreshing JCL Generation . . . 34
Stopping the Server . . . 34
Changing Defaults for Some CLIO/S Environment Variables . . . 34
Running Multiple PARSRVs . . . 35
Ensuring Password Security . . . 35
Using the CLIO/S Striping Function in Your Application . . . 35
Enabling CLIO/S for Striping . . . 36
Monitoring Striping Activity . . . 36
Using CLIO/S and Non-CLIO/S Applications on the Same Adapter . . . 37
Sharing with Other CTC Applications . . . 37
Sharing with CLAW Applications . . . 37
Enabling TSO Access to the CLFTP and CLPLINK Commands . . . 37

Chapter 4. Configuring and Managing the CLIO/S Tape Pseudo Device Driver . . . 39
Configuring and Starting the CLIO/S Tape Mount Daemon . . . 39
Monitoring Tape Mount Daemon Activity . . . 42
Stopping the CLIO/S Tape Mount Daemon . . . 43
Changing the CLIO/S Tape Pseudo Device Driver . . . 43
Reconfiguring the Tape Pseudo Device Driver/Restarting the CLIO/S Tape Mount Daemon . . . 44
Restarting the Tape Pseudo Device Driver After a System Reboot . . . 44
Unconfiguring the CLIO/S Tape Pseudo Device Driver . . . 45
Removing the CLIO/S Tape Pseudo Device Driver . . . 46
Administrative Commands . . . 47
cltclntd—Starting the CLIO/S Tape Mount Daemon . . . 47
cltterm—Stopping the CLIO/S Tape Mount Daemon . . . 48

Part 2. Command Line Interfaces . . . 51

Chapter 5. Accessing CLIO/S Tapes With AIX Commands and Subroutines . . . 53
Allocating and Using Tapes . . . 53
User Commands . . . 54
cltalloc—Allocating a CLIO/S Tape Device . . . 54
cltunall—Unallocating a CLIO/S Tape Device . . . 57

Chapter 6. Transferring Files Using the CLIO/S File Transfer Program . . . 59
How CLIO/S FTP Works . . . 59
If There Are Multiple PARSRVs . . . 59
The CLFTP Command . . . 60
Identifying Yourself to CLFTP . . . 61
Establishing Your Working Directory . . . 62
Customizing CLFTP Defaults . . . 63
CLFTP Subcommands . . . 63
Displaying Help . . . 65
HELP or ?—Displaying Help . . . 65
Obtaining Status and System Information . . . 66
ECHO—Controlling the Redisplay of Subcommands . . . 66

HASH—Controlling the Display of File Transfer Status Messages 66
LOCSTAT—Retrieving Status Information From the Local Host 67
STATUS—Retrieving Status Information From a Remote Host 68
Working With Directories and Partitioned Data Sets on the Remote Host . . . 70
CD—Changing the Directory on the Remote Host 70
LS—Listing Files in a Remote Directory 72
MKDIR—Creating a Directory on the Remote Host 73
PWD—Displaying the Current Working Directory on the Remote Host . . . 74
Working With Directories and Partitioned Data Sets on the Local Host 75
LCD—Changing the Local Working Directory 75
LLS—Listing Files in a Local Directory 76
LMKDIR—Creating a Directory on the Local Host 77
LPWD—Displaying the Current Working Directory on the Local Host 79
Preparing the Environment 79
ASCII—Changing the Data Transfer Type to ASCII 80
BINARY—Changing the Data Transfer Type to Image 80
ENDONERROR—Controlling Termination on Error 80
LOCSITE—Specifying Site Information to the Local Host 81
SITE—Sending Site Specific Information to a Remote Host 82
SITE and LOCSITE Parameters 83
TRANSLATE—Controlling Which Host Performs Data Conversion 87
Transferring Data 87
APPEND—Appending to a Remote File or Data Set 88
DELETE—Deleting Files On the Remote Host 89
GET—Copying Files From the Remote Host 90
PUT—Copying Files to the Remote Host 91
Disconnecting from the Host 93
QUIT—Leaving the CLFTP Environment 93
Starting CLFTP as a Batch Job 93
Transferring Data Across Intermediate Hosts 94
CLFTP MVS Return Codes and AIX Exit Values 96
CLFTP Error Messages 97
CLFTP MVS Abends 97
Resolving Problems With CLIO/S File Transfer Program 97

Chapter 7. The CLIO/S Cross System Pipe Link 99
What Is a Pipe? 99
Transparent Access Across a CLIO/S Network 99
Using the CLIO/S Cross System Pipe Link 101
Copying Files To and From Remote Hosts 102
Accessing Remote Files From an Application 102
Connecting Pipes Between Remote Hosts 103
Creating a CLIO/S Pipe With clpmk 103
clpmk–Creating a CLIO/S Pipe 104
Connecting to a CLIO/S Pipe With CLPLINK 105
clplink–Connecting To a CLIO/S Named Pipe 106
Customizing CLPLINK Defaults 117

Part 3. Application Programming Interfaces 119

Chapter 8. Using the CLIO/S Sockets Interface 121
Using the CLIO/S Sockets Interface 121
Socket Calls Provided by the CLIO/S Sockets Interface 122

Using Socket Calls Not Provided by the CLIO/S Sockets Interface 123
Portability 123
Using Preexisting Socket Applications with the CLIO/S Sockets Interface 123
Using the CLIO/S Sockets Interface in Other Environments 123
Limitations of the CLIO/S Sockets Interface 124
CLIO/S Sockets Interface Performance Hints 125
Linking and Compiling Sockets 125

Chapter 9. Using the Communication Manager Interface (CMI) 127
CMI Header File fcfcmi.h 127
About CLIO/S Data Conversions 127
Using the CMI Functions 128
clio_accept—Accepting a CLIO/S Connection from Another Process . . . 128
clio_async—Setting the Asynchronous Data Flag 129
clio_check—Checking the Number of Bytes Transmitted 129
clio_close—Closing a CLIO/S Connection 130
clio_connect—Requesting a CLIO/S Connection to Another Process . . . 130
clio_create—Starting a Process on a Node 131
clio_end—Ending a CLIO/S Connection 132
clio_env—Getting the Environment 132
clio_exp—Setting the Environment 133
clio_h2i—Locating Machine Information Using an Internet Address 134
clio_init—Initializing CLIO/S 134
clio_kill—Stopping a Process on a Node 134
clio_l2n or clio_n2l—Converting Data Representations 135
clio_m2i—Locating Machine Information Using a Machine Number 136
clio_n2h—Getting an Internet Address 136
clio_n2i—Locating a Machine by Symbolic Name 137
clio_n2m—Getting the Machine Information ID 137
clio_own—Allowing a Process to Change Ownership of a Connection . . 138
clio_pid—Returning a Process ID 138
clio_read—Reading Data over a CLIO/S Connection 139
clio_write—Writing Data over a CLIO/S Connection 139
tcp_accept—Accepting a TCP/IP Connection 140
tcp_connect—Establishing a TCP/IP Connection 140

Chapter 10. Using the Distributed Processing Interface (DPI) 143
The DPI Header File fcfdpi.h 143
About CLIO/S Data Conversions 143
Overview of the DPI 144
The CLIO/S DPI Master/Servant Model 145
How the CLIO/S DPI Operates 146
The CLIO/S DPI Sample Program 148
Starting the Master 149
Starting the Servant 149
After the Servant is Started 150
Using the DPI Functions 150
ALERT—Setting an Alert Group 150
ALERTGRP or ALRTGRP—Adding a Process ID to an Alert Group 151
cdpi_async or CDPIASY—Setting an Asynchronous Flag By Group 151
cdpi_check or CDPICHK—Checking Number of Bytes Transmitted by a Group 152
cdpi_s2c—Returning a CLIO/S Pointer 152
CLFLUSH—Clearing Buffers Written Using SPANY 153

clio_l2n or LNxxxx—Converting Data to Network Format 153
clio_n2l or NLxxxx—Converting Data to Local Format 154
CLIOBLK—Setting the Buffer Bypass Limit 155
CLIOBUF—Setting the Size of the Internal Buffer 156
CLIODEB—Masking Printed Status Messages 157
CLIOENV—Getting Environment Information 157
CLIOMST—Telling OPNSRV to Start Independent Masters 158
CLIOSOCK—Returning the Socket Descriptor 158
CLIOTBS—Changing Send and Receive Buffer Sizes 159
CLIOTMO—Setting the Time-out Period 159
CLSFAM—Stopping All Processes in a Family 160
CLSPAR—Closing the DPI Environment 160
CLSSRV—Stopping the Servant Process 160
FNDMACH—Locating a Machine by Symbolic Name 161
MSANY or MSxxxx—Transferring Data from Master to Servant 161
MS_flush or MSFLSH—Confirming Data Was Sent from Master to Servant 163
NMACHS—Counting Valid Entries in PARCONF 163
OPNFAM—Opening a Family of Processes 164
OPNPAR—Initializing the DPI Environment 164
OPNSRV—Starting a Servant Process 165
QMACH—Getting Machine Information 166
QSLAVE—Getting Servant Machine Information 166
RPANY or RPxxxx—Receiving Data from a Process 167
rw_dpi—Reading and Writing Data 168
SCHEDUL—Scheduling a Task to Start 169
SETFAM—Setting the Family ID 170
SETSRV—Forcing Data Transfer to the Servant ID 170
SMANY or SMxxxx—Transferring Data from Servant to Master 171
SM_flush or SMFLSH—Confirming Data Was Sent from Servant to Master 172
SPANY or SPxxxx—Sending Data to a Process 172
SYNCHRO—Waiting Until All or Selected Tasks are Completed 173
TSKEXIT—Informing a Master the Servant Process is Finished 174
WAITFOR—Waiting Until an Alert is Available 175
WAITREQ—Indicating a Servant is Waiting for Work 175
Changing the Buffering Mechanism 176
Using CLIO/S ESTAE Support 177
Chained Channel Programs for BMCA and ESCON 178
Returning the Protocol Type 178
Starting a Series of Read or Write Operations 179
Using the rw_dpi Call 179

Chapter 11. Using the CLIO/S Tape Programming Interface and Support 181
The CLIO/S Tape Header File fcftpint.h 181
Sample CLIO/S Tape Program 181
Using the CLIO/S Tape Interface Functions 182
ctpalloc or tpalloc—Starting the Remote Tape Server 183
ctpclose or tpclose—Closing the Tape 186
ctpopen or tpopen—Opening the Tape Device for Use 187
ctpread or tpread—Reading the Tape 188
ctprew or tprew—Rewinding the Tape 189
ctpskpf or tpskpf—Skipping Tape Files 189
ctpskpr or tpskpr—Skipping Tape Records 190
ctpunall or tpunall—Stopping the Remote Tape Server 191
ctpvols or tpvols—Listing the Tape Volumes 191

ctpweof or tpweof—Writing Tape Marks . . . 192
ctpwrite or tpwrite—Writing to Tape . . . 193
ctpwrto or tpwrto—Writing to the Operator . . . 194
CLIO/S Tape Interface Return Values . . . 195
Tape Exit On MVS and AIX . . . 196
MVS Tape Exit FCFTPX00 . . . 196
AIX Tape Exit ctpexit . . . 197
Effects of MVS Tape Management on CLIO/S . . . 197
Reading and Writing Multiple Tape Volumes . . . 198
Compiling and Linking CLIO/S Tape Interface Programs . . . 199
On MVS/ESA . . . 199
On the RISC System/6000 . . . 199
FCFTTEST Sample Program . . . 199
Using TPWRTO Support On AIX . . . 200
Special Considerations for ANSI Standard Label (AL) Tapes . . . 201
Preformatting the AL Tape . . . 201
Writing the Data Set Name in the Tape Label Header File . . . 202
Performing Tape I/O . . . 202

Part 4. Appendixes . . . 203

Appendix A. Support Programs and Sample JCL . . . 205
CLIO/S Support Programs . . . 205
Cancelling a CLIO/S Job—FCFCAN, FCFCANC . . . 206
Verifying that CLIO/S Can Communicate Between Master and Servant—FCFIDUM . . . 207
Starting the CLPLINK program—FCFFCLPR, FCFFCLPW, clplink . . . 209
Switching Server Debug Flags—FCFPARDE, pardeb . . . 210
Displaying Server Information—FCFPARIN, parinfo . . . 210
Testing Whether PARSRV is Running—FCFPARPI, parping . . . 211
Refreshing PARSRV Defaults on MVS—FCFPARRE, parrefr . . . 212
Starting the CLIO/S Server–FCFPARSR, parsrv . . . 213
Stopping a CLIO/S Server—FCFPARST, parstop . . . 214
Testing the CLIO/S Cross System Pipe Link—FCFFPIPR, FCFFPIPW, fcffpipr, fcffpipw . . . 215
Varying On and Off Subchannels—FCFVARY . . . 216
Starting BMCA/PCA Subchannels on the Workstation With the online Program . . . 218

Appendix B. Generating JCL for Jobs Started by PARSRV . . . 221
Specifying the JCL Generation Language . . . 221
Migrating From CLIO/S Version 1 . . . 222
Generating JCL for started tasks . . . 222
Using REXX/MVS to Generate PARSRV JCL . . . 223
Services Available to FCFREXX . . . 223
Input to FCFREXX . . . 223
Setting the Return Code In FCFREXX . . . 225
Using the SAY and TRACE Commands . . . 225
Under the Covers of FCFREXX . . . 225
Sample FCFREXX Program . . . 225
Using Pseudo-JCL to Generate PARSRV JCL . . . 235
Specifying Pseudo-JCL Method . . . 235
Overview . . . 235

Specification 236
Customizing Your INSTDEF File 239
Sample FCFDEFS file 243

Appendix C. Dynamic Device Allocation on MVS 251

Appendix D. CLIO/S Performance and Throughput 257
Performance Considerations 257
Performance Comparisons 259
The Performance of CLIO/S DPI Interface 259
The Performance of CLIO/S Tape Server and CLIO/S File Transfer Program 261

Appendix E. Debugging CLIO/S 263
Debugging the Installation Process 263
Installation Problems 263
Debugging Intermittent, Hardware or Network Problems 266
General AIX debugging tips 267
Viewing Error Reports 267
Tracing an Adapter 267
Displaying Network Statistics 268

Index 269

Special 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 or other legally protectible rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, 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

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 and Service Marks

The following terms, denoted by an asterisk (*) in this publication, are trademarks or registered trademarks of the IBM Corporation in the United States and/or other countries:

AIX	RISC System/6000
AIX/ESA	S/390
Enterprise System/9000	Scalable POWERparallel Systems
ES/9000	System/390
IBM	9076 SP1
MVS/ESA	9076 SP2

The following terms, denoted by a double asterisk (**) in this publication, are trademarks of other companies:

Ethernet	Xerox Inc.
PostScript	Adobe Systems Incorporated
Sun RPC	Sun Corporation
UNIX	X/Open Company Limited

© Copyright IBM Corp. 1993, 1995

Acknowledgement

The product development team acknowledges Oddvar Gjerde and Oystein Skudal from the IBM European Petroleum Center, for originally developing CLIO/S. We also acknowledge John Hague, from the Scientific and Technical Systems and Solutions Group, IBM UK, Ltd., for his ideas on developing CLIO/S.

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 transfer functions
- 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 xv 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.

© Copyright IBM Corp. 1993, 1995

- **Chapter 2, Migrating From CLIO/S Version 1**

  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.
- Using CLIO/S's command line interfaces:
  - **Chapter 7, 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.
  - **Chapter 5, 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 6, Transferring Files Using the CLIO/S File Transfer Program**

    Tells you how to use the CLIO/S FTP to transfer files between networked hosts.
- Using CLIO/S application programming interfaces
  - **Chapter 8, Using the CLIO/S Sockets Interface**

    Provides information about how to use the CLIO/S sockets interface.
  - **Chapter 9, 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 10, 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 11, 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, Support Programs and Sample JCL**

    Describes the support programs that help you manage a CLIO/S network. Sample JCL coding is provided.
  - **Appendix B, Generating JCL for Jobs Started by PARSRV**

    Describes the two ways to generate JCL for starting servant processes on MVS.
  - **Appendix C, 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 D, CLIO/S Performance and Throughput**

    Discusses factors that influence the performance of CLIO/S and shows some results from IBM's performance testing.
  - **Appendix E, Debugging CLIO/S**

    Gives hints for resolving problems with CLIO/S.

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

  These three publications describe the AIX socket calls.
- *AIX Version 3.2 Communication Programming Concepts*, SC23-2206

  This publication is 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*, SC23-2427

  This publication describes the BMCA hardware.
- *IBM ESCON Channel Adapter User's Guide and Service Information*, SC23-2474

  This publication describes the ESCON CTC hardware.

- *TCP/IP Version 2 Release 2.1 for MVS: Programmer's Reference*, SC31-6087

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

- *IBM TCP/IP for MVS: Programmer's Reference*, SC09-1261

  This publication describes TCP/IP on MVS.

- *JCL Reference*, GC28-1654

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

- *BatchPipes/MVS 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 RISC System/6000 workstations and the IBM 9076 Scalable POWERparallel Systems SP1 and SP2.

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, May 1995

This edition documents the CLFTP ENDONERROR and DELETE subcommands and the CLPLINK –**n** option. It also contains minor corrections and enhancements.

Summary of Changes, April 1995

This edition documents new function added in CLIO/S Version 2 Release 1, including:

- Cross system pipe link (CLPLINK)
- CLFTP access to MVS and AIX tape data
- Ability to override default settings in CLFTP
- Additional CLFTP subcommands: APPEND, LS, and LLS.
- REXX interface for JCL generation on MVS
- AIX standard error logging policy

It contains a new appendix listing the MVS dynamic allocation parameters supported by CLIO/S. It also contains technical and editorial improvements.

Some introductory material was moved from this manual to the new document, *CLIO/S General Information*, GC23-3879.

Summary of Changes, September 1994

This edition documents enhancements added in Version 1 Release 2, including:

- CLIO/S Tape Pseudo Device Driver
- CLIO/S File Transfer Program

This edition also contains technical, organizational, and editorial improvements.

Summary of Changes, May 1994

This edition documents changes resulting from APAR PN55176, including these technical enhancements:

- The addition of the FCFVARY program to simplify varying channels online and offline
- Improved support for multiple instances of TCP/IP on a single machine
- Optional termination of MVS servant programs when an AIX master is stopped by <**ctrl-c**>
- An increase in the maximum number of stripes from 2 to 4
- Improved tape server error handling
- Tape server access to tape device data compaction

- Optional tape server suppression of automatic end of volume handling with cataloged tapes
- Support for multiple instances of PARSRV on a machine

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 9076 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 SP2)**. 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

CLIO/S's Major Functions

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."
- 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 RISC System/6000 ESCON adapter
- Serial Optical Channel Converter (SOCC) for the RISC System/6000

TCP/IP communications are also supported. This lets you create CLIO/S networks that include workstations or 9076 SP2s that do not have a SOCC attachment. 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 SP2 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 6, "Transferring Files Using the CLIO/S File Transfer Program" on page 59.

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 7, “The CLIO/S Cross System Pipe Link” on page 99.

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 8, “Using the CLIO/S Sockets Interface” on page 121 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 9, “Using the Communication Manager Interface (CMI)” on page 127 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 10, “Using the Distributed Processing Interface (DPI)” on page 143 for details.

The Tape Server

The CLIO/S tape server provides two ways to access any tape device on a RISC System/6000, 9076 SP2, 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 5, “Accessing CLIO/S Tapes With AIX Commands and Subroutines” on page 53 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 11, “Using the CLIO/S Tape Programming Interface and Support” on page 181 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. PARSRV creates the high-speed link between the hosts and then starts the servant program.

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 (FCFSERV). FCFSERV 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 23 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.

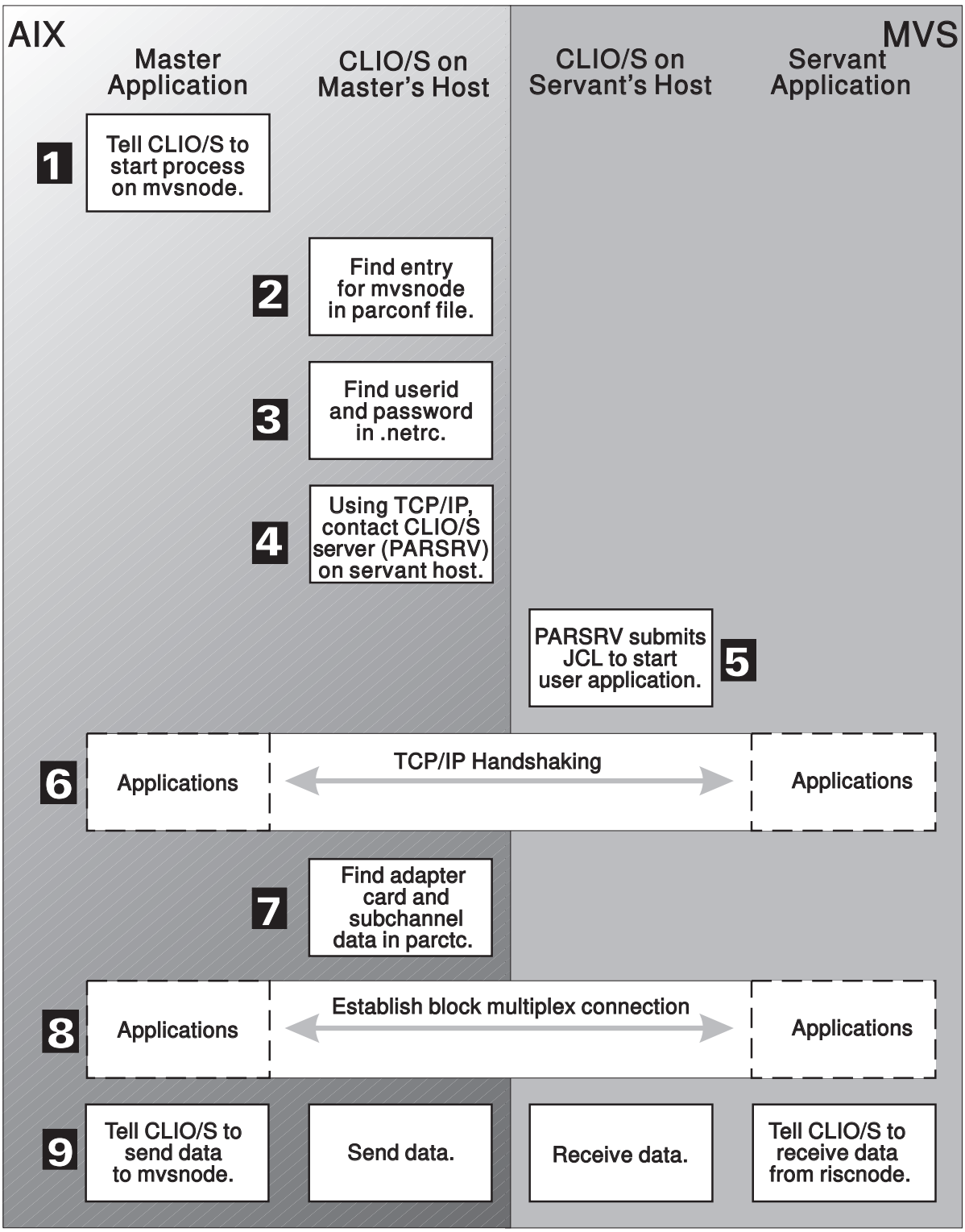


Figure 1. An Example of CLIO/S Communications

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 RISC System/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 A, “Support Programs and Sample JCL” on page 205 for more details on JCL generation.
6. The application at the MVS servant and the application at the AIX master establish TCP/IP communication. Before this, CLIO/S had set up all the communications.
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.

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.kgn.ibm.com** (129.40.10.220). The file name is **/pub/pps/clios/cliogi.ps.Z**. “Downloading Files from the Anonymous FTP Server” on page 9 describes how to connect to the server and download the file.

- *CLIO/S User's Guide and Reference* (no order number)

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/clios/clio_user_ref.ps**.

- You can download a PostScript listing from the anonymous FTP server **lscftp.kgn.ibm.com** (129.40.10.220). The file name is **/pub/pps/clios/clio_user_ref.ps.Z**. "Downloading Files from the Anonymous FTP Server" on page 9 describes how to connect to the server and download the file.

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.

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.kgn.ibm.com** (129.40.10.220) 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.

Chapter 2. Migrating From CLIO/S Version 1

This chapter describes the tasks involved in migrating to CLIO/S Version 2 Release 1 from Version 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 1, 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 1 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 1 on MVS, you must:

- Relink installation-maintained applications as described in “Migrating CLIO/S Applications” on page 12.

  No changes to the source code for your CLIO/S applications are required.
- Begin migrating any customized JCL generation programs to the new JCL generation language, REXX. See “Migrating From CLIO/S Version 1” on page 222 for more information about the migration process. “Using REXX/MVS to Generate PARSRV JCL” on page 223 describes how to create JCL generation programs using REXX.

On AIX

Before installing CLIO/S Version 2 Release 1 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**. 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.

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

After installing CLIO/S Version 2 Release 1 on the AIX system, you must:

© Copyright IBM Corp. 1993, 1995

- 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 now located in subdirectories of **/usr/lpp/fcf**. 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.

Part 1. Configuring and Managing

Chapter 3. Configuring and Starting CLIO/S Networks 15

Chapter 4. Configuring and Managing the CLIO/S Tape Pseudo Device Driver 39

© Copyright IBM Corp. 1993, 1995

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 17
  - "Configuring Your Channel to Channel Adapters (CTCs)" on page 17
  - "Creating and Customizing CLIO/S Configuration Files" on page 23
  - "The CLIO/S Server—PARSRV" on page 28
  - "Changing Defaults for Some CLIO/S Environment Variables" on page 34
  - "Running Multiple PARSRVs" on page 35
  - "Ensuring Password Security" on page 35
  - "Using the CLIO/S Striping Function in Your Application" on page 35
  - "Using CLIO/S and Non-CLIO/S Applications on the Same Adapter" on page 37
  - "Enabling TSO Access to the CLFTP and CLPLINK Commands" on page 37
- 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 23.
  - "Setting CLIO/S Environment Variables" on page 29

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 39.

© Copyright IBM Corp. 1993, 1995

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

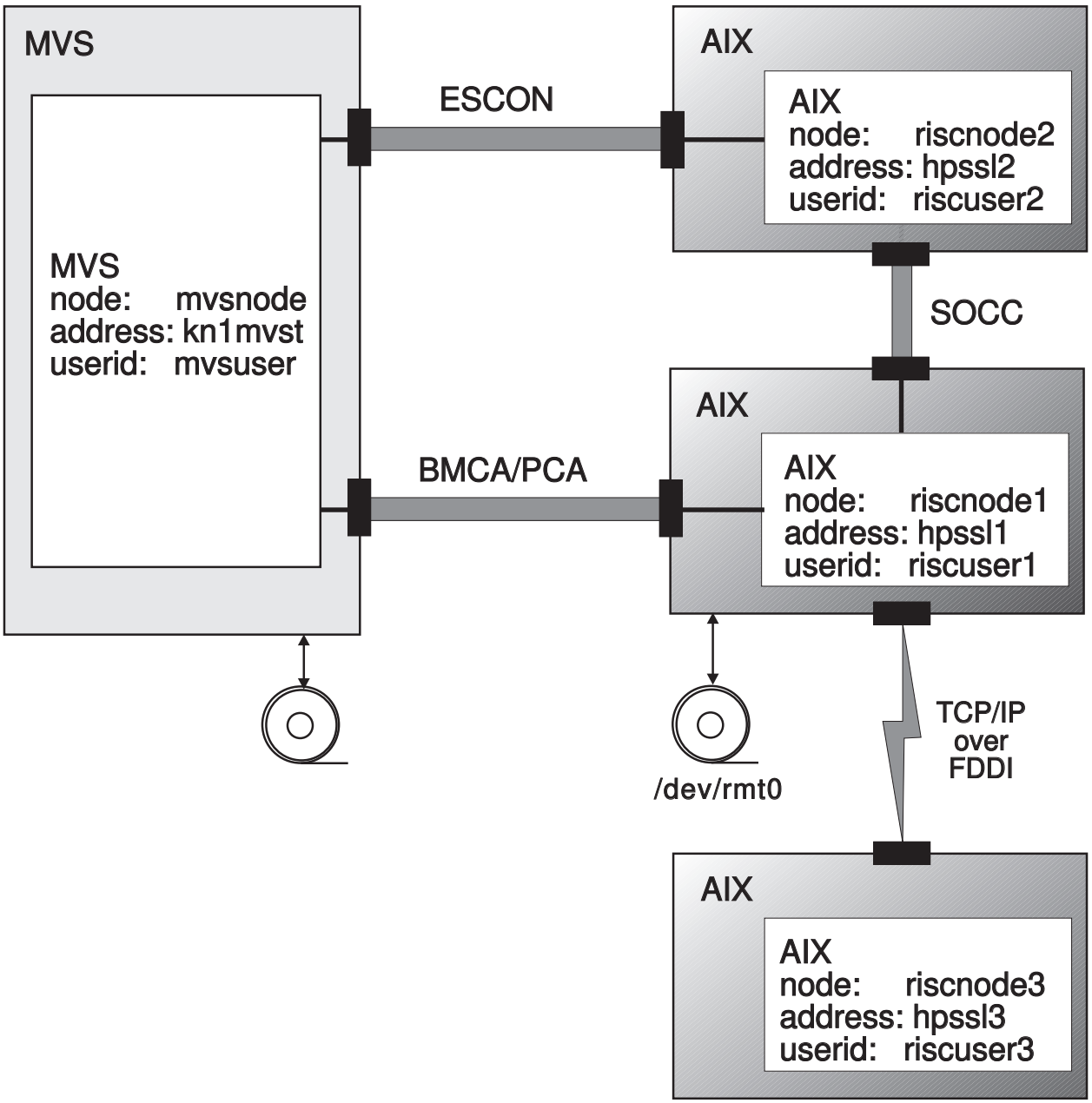


Figure 2. The Sample CLIO/S Configuration Used Throughout This Document

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 xv, 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 29 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 18

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 18 shows the definitions required to add CTC addresses 640 to 67F to CHPID 1E for four RISC System/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 RISC System/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 SCTC.

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=CTC,ADDRESS=(640,16),CUNUMBR=640,FEATURE=370
 IODEVICE  UNIT=CTC,ADDRESS=(650,16),CUNUMBR=650,FEATURE=370
 IODEVICE  UNIT=CTC,ADDRESS=(660,16),CUNUMBR=660,FEATURE=370
 IODEVICE  UNIT=CTC,ADDRESS=(670,16),CUNUMBR=670,FEATURE=370
*
**************************************************
*           ESCON CNTLUNIT/IODEVICE              *
**************************************************
 CNTLUNIT  CUNUMBR=6C0,PATH=(26),UNIT=3088,                             X
              UNITADD=((C0,64))               3088 CTC
 IODEVICE  UNIT=CTC,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 21
- "Allowing Access to SOCC Devices" 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 21.

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 18 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 18. In that example MVS defined 16 addresses for the CTC connection to this RISC System/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, SC23-2427.

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 216 and "Starting BMCA/PCA Subchannels on the Workstation With the online Program" on page 218.

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 RISC System/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. Update address (local) and Address (remote) to reflect the MVS subchannel address. This example configures the ESCON adapter in Figure 3 whose esoteric name is RS6K5, with the unit address of 6C0. Place the hex address '0C0' in both the Address (remote) and Address (local) fields. IBM recommends that you configure your ESCON subchannels with a single value for both remote and local addresses.

The local address of an ESCON adapter must be unique on each RISC System/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 RISC System/6000 as a way of connecting the application to the particular subchannel definition.

For example, a single ESCON adapter in the RISC System/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 21.

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 RISC System/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 21 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*, SC23-2474.

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. Among other data, the results should 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, for example:

```
no -o lowclust=40
```

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 additional mbufs.

Allowing Access to SOCC Devices

To enable CLIO/S applications to use the SOCC protocol, you must allow access to the SOCC device, /**dev**/**ops0**. On AIX, set the appropriate file permissions to allow users to access the device. For example, to give all users permission to use the SOCC device in their CLIO/S applications, use these commands:

```
su root
chmod guo+rw /dev/ops0
```

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	24	Each user needs access
PARCTC	Administrator, User	Each workstation using ESCON or BMCA/PCA	Used when creating a CTC connection	26	Not needed on MVS
NETRC	User	MVS and AIX	Making a connection	27	Used by other services, also.

The PARCONF File

The PARCONF file is the first place CLIO/S looks for configuration information. PARCONF contains the information CLIO/S uses to locate nodes and know how to connect to them. Each user ID in a CLIO/S network must have access to a PARCONF file on the host that starts the client program.

PARCONF is found in **$HOME**/**parconf** or /**etc**/**parconf** on AIX. If both files exist, CLIO/S uses **$HOME**/**parconf**. On MVS it is in *user_ID*.**PARCONF**, or it can be supplied with DD statements in the JCL for starting the application. Figure 25 on page 208 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.

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 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 RISC System/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
2 SOCC
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.

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**)
SOCC Consult your IBM representative about protocol-specific data

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 16:

- 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. In many cases, different user IDs on the same processor or on different processors 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**.

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 **$HOME/parctc** or **/etc/parctc** on RISC System/6000s. If both files are present, **$HOME/parctc** is used. 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*]

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. The address may 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 17 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.

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 16:

- 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 PARCONF file because does not have a CTC attachment to the mainframe.

To run CLIO/S in a striping environment, you must modify the PARCTC file. See "Using the CLIO/S Striping Function in Your Application" on page 35 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 RISC System/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 25 on page 208 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 **FCFNETRC** on MVS and in **/etc/netrc** 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 16:

- 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	29
Setting environment variables	29
Selecting the JCL Generation method (MVS only)	33
Starting the PARSRV server	33
Refreshing JCL Generation (MVS only)	34
Stopping the Server	34

The following table lists the customizable programs you use to start CLIO/S. Other CLIO/S support programs are described in Appendix A, “Support Programs and Sample JCL” on page 205.

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	28	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	34	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.” You can also run multiple PARSRVs on a single host, each with its own port number (see “Running Multiple PARSRVs” on page 35 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 34.

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:

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

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 34 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 155 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 156 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 RISC System/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 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
8192	0x2000	CTC or SOCC open or close to trace abnormal but acceptable situations
16384	0x4000	CTC or SOCC read or write to trace normal data flow
32768	0x8000	Print detailed error messages about CTC or SOCC read or write (always on in SOCC routines)

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 34 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 34.

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 34.

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 35 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/local/CLIO**. See "Changing Defaults for Some CLIO/S Environment Variables" on page 34 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 159 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 159 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

Specifying the JCL Generation Language

Before starting PARSRV on MVS, you must decide which program it uses to create the JCL that starts servant programs. When you are starting PARSRV on AIX, skip this step; PARSRV on AIX does not create JCL. You can write JCL generation programs in two languages:

- REXX
- Pseudo-JCL

To specify REXX JCL generation, include the REXXJCL DD statement in the JCL that starts PARSRV. This DD statement must contain the name of a partitioned data set containing member FCFREXX, a REXX program. To specify pseudo-JCL, make sure there is no REXXJCL DD statement in the JCL that starts PARSRV.

See Appendix B, "Generating JCL for Jobs Started by PARSRV" on page 221 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 30 through 32. 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
```

Refreshing JCL Generation

This section applies only to PARSRV servers running on MVS. PARSRV starts servant jobs on MVS by submitting JCL. The way you modify JCL generation depends on which method you're using to generate JCL. See Appendix B, “Generating JCL for Jobs Started by PARSRV” on page 221 for more information.

If PARSRV is using FCFREXX to create JCL for the servant jobs it starts, you can change the way JCL is created by modifying the REXX program in FCFREXX. This change takes effect immediately; the next servant job is submitted with your new JCL. For this reason, edit a *copy* of FCFREXX, rather than the actual PDS member being used by PARSRV.

If PARSRV is using pseudo-JCL to create JCL for the servant jobs it starts, changing the way JCL is created with these two steps:

1. Modifying the FCFDEFS files
2. Refreshing the server

Refresh PARSRV by submitting the FCFPARRE job on MVS or using the **parrefr** command on AIX. For more information about these commands, see “Refreshing PARSRV Defaults on MVS—FCFPARRE, parrefr” on page 212.

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 214 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 29 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. 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 16, you must duplicate the adapter's entries in the PARCTC file. The entries can define different subchannels; if they are identical, CLIO/S selects 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 29 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 uses 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 19 and "Configuring ESCON Subchannels" on page 21 describe using the SMIT configuration tool to configure subchannels on an adapter. CTC subchannels are used one at a time, but usually you 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*, SC23-2427, or the *IBM ESCON Channel Adapter User's Guide and Service Information*, SC23-2474.

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 RISC System/6000 systems using standard UNIX commands, for example, **tar**, **cpio**, or **backup**.

An administrator makes 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	39
Monitoring tape mount daemon activity	42
Stopping the tape mount daemon	43
Changing the tape pseudo device driver	43
Reconfiguring the tape pseudo device driver/restarting the tape mount daemon	44
Restarting the tape pseudo device driver after a system reboot	44
Unconfiguring the tape pseudo device driver	45
Removing the tape pseudo device driver	46

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

The sample CLIO/S configuration in Figure 2 on page 16 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 28.

   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
   ```

   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. 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 16, 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 24 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, make sure that a SOCC connection is available or that TCP/IP is set up to use a fast connection such as FDDI or HIPPI.

   b. Update the Allocation parameter / device name field. For more information about dynamic allocation parameters for tape devices, see the discussion of *allocparms* on page 184 and Appendix C, "Dynamic Device Allocation on MVS" on page 251.

      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 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.

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 40.

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
   ```

To restart the daemon, follow the procedures in "Reconfiguring the Tape Pseudo Device Driver/Restarting the CLIO/S Tape Mount Daemon" on page 44.

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 45. 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” on page 44.

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 39.

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 43.
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 44.

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 43.
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 39.

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
open	**ctpopen()**
write	**ctpwrite()**
read	**ctpread()**
allocate	**ctpalloc()**
unallocate	**ctpunall()**
close	**ctpclose()**

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:

STERASE	Erase the tape, leaving at load point
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 not automatically restarted when the system reboots. To restart the tape mount daemon, use SMIT or the **mkdev** –**l** command.

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 instantiation (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`

Related Topics

Commands: **cltclntd**

Part 2. Command Line Interfaces

Chapter 5. Accessing CLIO/S Tapes With AIX Commands and Subroutines 53

Chapter 6. Transferring Files Using the CLIO/S File Transfer Program . . 59

Chapter 7. The CLIO/S Cross System Pipe Link 99

© Copyright IBM Corp. 1993, 1995

Chapter 5. 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 39.

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 **ctalloc** command. Only that user can access that tape device. See “Configuring and Starting the CLIO/S Tape Mount Daemon” on page 39 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

See “Special Considerations for ANSI Standard Label (AL) Tapes” on page 201 for information about ANSI Standard Label (AL) tapes.

User Commands

cltalloc—Allocating a CLIO/S Tape Device

Syntax

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

–**c** For MVS tapes, 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**:

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

If the name is not known, use –**l bpl** parameter to bypass label processing. For AIX tapes, this parameter is ignored.

–**d** [**dsn**=]*dsname*
For MVS tapes, 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. For AIX tapes, this parameter is ignored.

The value you specify for *dsname* is converted to upper case.

–**e** [**ex**=]*yyddd*
For MVS tapes, specifies the expiration date for a **new** data set. *yyddd* is a Julian date. For AIX tapes, this parameter is ignored.

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*).

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

This is a required parameter.

–**l nl** | **al** | **blp** | **sl**
For MVS tapes, 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.

–**p new** | **old**

For MVS tapes, 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**. For AIX tapes, this parameter is ignored.

You may type **new** and **old** in upper, lower, or mixed case; it is converted to upper case.

0 –**r** [**ret**=]*nnnn*

For MVS tapes, specifies the retention period for a **new** data set, in number of days. For AIX tapes, this parameter is ignored.

Note: –**e** and –**r** are mutually exclusive. You can specify the
0 longevity of the tape as either an expiration date (–**e** *yyddd*) or
0 as a number of days from now (–**r** *nnnn*).

–**v** [**vol**=]*volser*

For AIX or MVS tapes, 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). You must have a **.netrc** file defined. See “The NETRC File” on page 27 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: 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] [-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
   ```

2. 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
   ```

3. 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
   ```

4. 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.

5. 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
```

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/pstn`

  A tape is already mounted.

- `cltunall: Tape on /dev/pstn 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 6. 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 28 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

```
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 **/usr/lpp/fcf/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 24 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 94 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 37.
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 27. 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 95.

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 70.

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 75.

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 63 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 70 and “LCD—Changing the Local Working Directory” on page 75
- 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 63 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
```

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	65
Obtaining status and system information	ECHO HASH LOCSTAT STATUS	66
Working with directories and partitioned data sets on the remote host	CD LS MKDIR PWD	70
Working with directories and partitioned data sets on the local host	LCD LMKDIR LPWD LLS	75
Preparing the environment	ASCII BINARY ENDONERROR LOCSITE SITE TRANSLATE	79
Transferring data	APPEND DELETE GET PUT	87
Disconnecting from the host	QUIT	93

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.	88
AScii	Sets the transfer type to ASCII. Use this to transfer text (non-binary) files between ASCII and/or EBCDIC hosts.	80
Binary	Sets the transfer type to IMAGE.	80
CD	Changes the working directory on the remote host.	70
DELETE	Deletes files and partitioned data sets on the remote host.	89
ECHO	Controls the redisplay of subcommands	66
ENDONERROR	Controls whether CLFTP quits after an error.	80
Get	Copies a file from the remote host to your local host.	90
HAsh	Controls the display of file transfer status messages.	66
HElp	Displays help information for CLFTP.	65

Table 5 (Page 2 of 2). CLFTP Subcommands

Subcommand	Description	Page
LCd	Changes the local working directory.	75
LLS	Lists the files in a local directory	76
LMkdir	Creates a directory on the local host.	77
LPwd	Displays the local current directory qualifier	79
LOCSIte	Specifies information that is used by the local host to provide service specific to that host system.	81
LOCSTat	Displays status information for the local host.	67
LS	Lists the files in a remote directory	72
MKdir	Creates a directory on the remote host.	73
PUt	Copies a file on your local host to the remote host.	91
PWd	Displays the current working directory.	74
QUIt	Leaves the CLFTP command environment.	93
SIte	Specifies information that is used by the remote host to provide service specific to that host system.	82
STatus	Displays status information for the remote host.	68
TRANSlate	Controls whether ASCII/EBCDIC translation is done on the local or the remote host.	87
?	Displays help information for CLFTP. Synonymous with HELP.	65

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 93 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'
0                       42093723 bytes transferred, Elapsed time: 10.01 sec., 4.01 Meg/s
0                       78088395 bytes transferred, Elapsed time: 20.09 sec., 3.71 Meg/s
0                       114582993 bytes transferred, Elapsed time: 30.11 sec., 3.63 Meg/s
0                       150577665 bytes transferred, Elapsed time: 40.11 sec., 3.58 Meg/s
0                       187072263 bytes transferred, Elapsed time: 50.39 sec., 3.54 Meg/s
0                       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
- ENDONERROR
- 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 83 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 67.

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 83 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 68.

SITE and LOCSITE Parameters

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 on page 84 lists the parameters provided for defining MVS data sets
- Table 7 on page 86 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 on page 84 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 3). 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'. See Appendix C, "Dynamic Device Allocation on MVS" on page 251 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'`
	
	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.
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.
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.

Table 6 (Page 2 of 3). SITE and LOCSITE Parameters for MVS Hosts

Parameters	Description
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.
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.
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.
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.
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.

Table 6 (Page 3 of 3). SITE and LOCSITE Parameters for MVS Hosts

Parameters	Description
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.
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.
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.

Use the AIX SITE and LOCSITE parameters to read from or write to raw devices or block devices on AIX systems. The AIX LOCSITE and SITE parameters are shown in Table 7.

Table 7. SITE and LOCSITE Parameters for AIX Hosts

Parameters	Description
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.
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.
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.
RAWwrite	By default, CLFTP and CLPLINK write only to tape and diskette raw devices. RAWWRITE specifies that CLFTP and CLPLINK should write to different types of raw devices such disk drives.
NORAWwrite	Resets the supported raw device types to the default: tape and diskette raw devices only.

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 62.

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 88.

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 62.

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 62.

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 88.
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 62.

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 88.

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 60.

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 9 on page 94, 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 9. 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 9, 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 16, ***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
0                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
0                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.

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 59.

- 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.

Refer to Appendix E, “Debugging CLIO/S” on page 263 for more information about problem solving.

Chapter 7. 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.

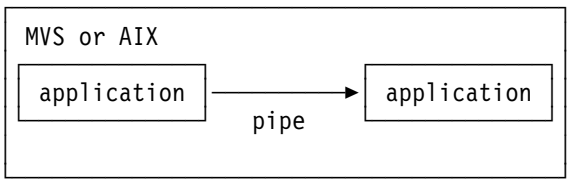


Figure 10. Piping eliminates intermediate output files.

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 SP2, while exploiting the data management capabilities of MVS.

© Copyright IBM Corp. 1993, 1995

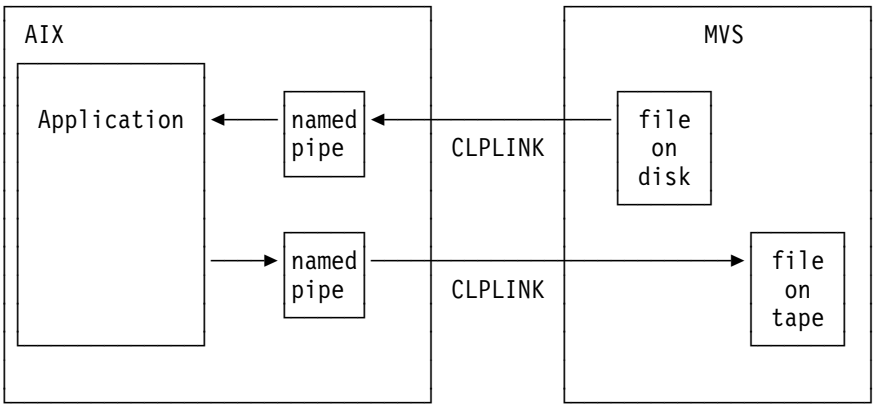


Figure 11. An AIX application transparently accessing files on a remote MVS system. You can use the CLIO/S Cross System Pipe Link with both named (as shown) and unnamed AIX pipes.

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.

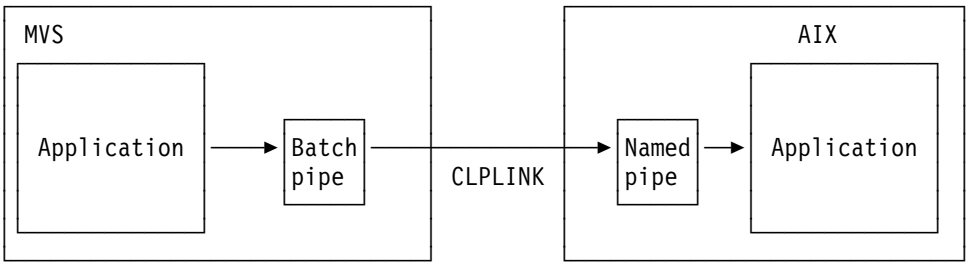


Figure 12. Piping across a CLIO/S network. The MVS application pipes data to an AIX application using named pipes.

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 SP2. 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.

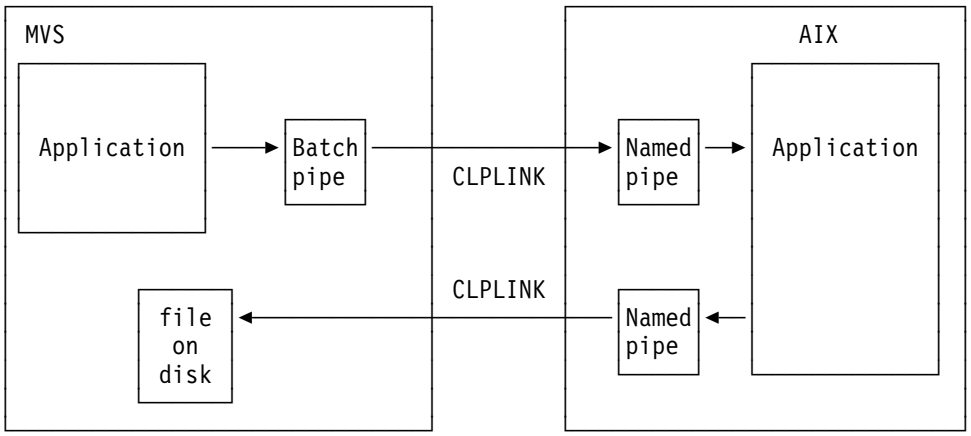


Figure 13. Piping across a CLIO/S network. The MVS application pipes data to an AIX application. Output created on AIX is saved in a file on MVS.

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 35 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 106.

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 102, 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 14. However, you don't always have to manually start both instances.

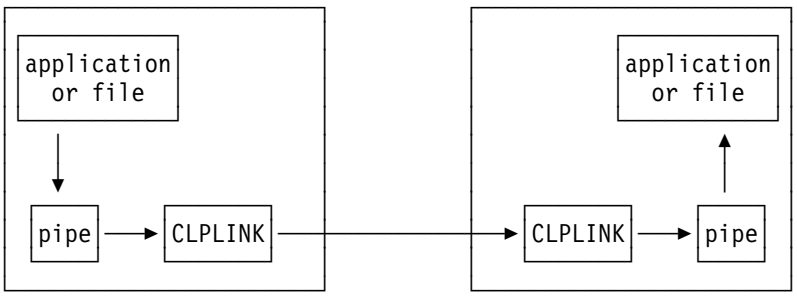


Figure 14. A CLPLINK instance on each host involved in a transfer.

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.

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* ...] [–**U** *userid*[**:***userid* ...] [–**P** *password*[**:***password* ...]]]
–**nx** *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 117.

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 the 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 35 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**.

−**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 83 for information about 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 83 for information about 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:

- 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** –**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 35 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.

–**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 B, “Generating JCL for Jobs Started by PARSRV” on page 221 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 37.

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 117.

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. CLPLINK uses QSAM for file I/O. Files that cannot be processed by QSAM are not read or written by CLPLINK.

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 SP2). 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 114 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:

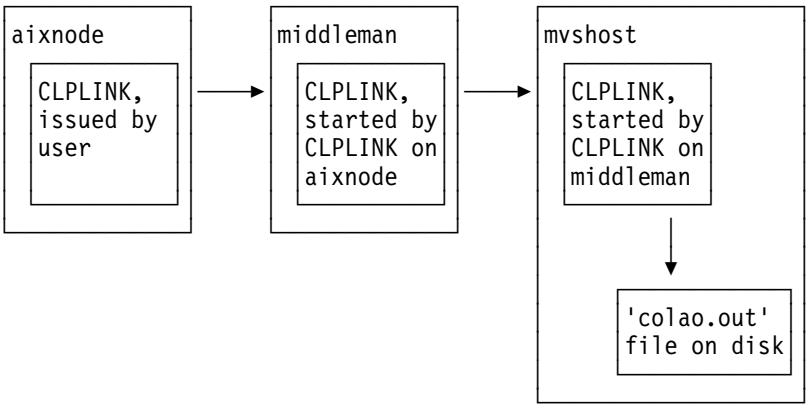
```
# Create the CLIO/S pipe
clpmk -w pipe1

# 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 116.



8. This example has the same end result as example 7 on page 115, 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————————

# Create a CLIO/S pipe
clpmk -w pipe1


# Connect to MVS through
# intermediate system
clplink -h middleman:mvshost \
        -p 50000             \
        -U jbecky -P secret  \
        -r pipe1

# Start the application
myprogram > pipe1
```

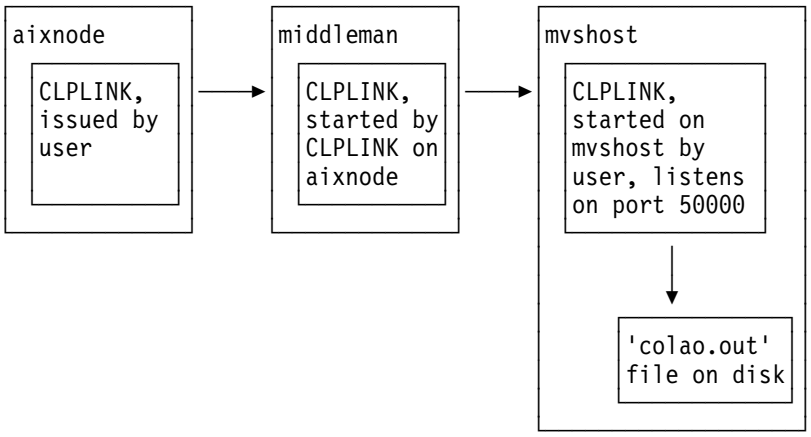
```
————————MVS side————————

# Listen on port 50000
# for someone to connect
clplink -p 50000  \
        -s "pri=10 sec=10" \
        -w 'colao.out'
```

   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 115.

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 63. 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 80.

Default Statement	Corresponding CLPLINK Option	Notes
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 80.
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 70.
HASH	–**v**	Controls the display of file transfer messages. For more information, see “HASH—Controlling the Display of File Transfer Status Messages” on page 66.
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 75.
LOCSITE	–**s**	Sets local site parameters. For more information, see “LOCSITE—Specifying Site Information to the Local Host” on page 81.
SITE	–**S**	Sets remote site parameters. For more information, see “SITE—Sending Site Specific Information to a Remote Host” on page 82.
TRANSLATE	(none)	Determines which host performs data translation. For more information, see “TRANSLATE—Controlling Which Host Performs Data Conversion” on page 87.

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 3. Application Programming Interfaces

Chapter 8. Using the CLIO/S Sockets Interface 121

Chapter 9. Using the Communication Manager Interface (CMI) 127

Chapter 10. Using the Distributed Processing Interface (DPI) 143

Chapter 11. Using the CLIO/S Tape Programming Interface and Support 181

© Copyright IBM Corp. 1993, 1995

Chapter 8. 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 RISC System/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 RISC System/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/fcf/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 15 on page 122 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

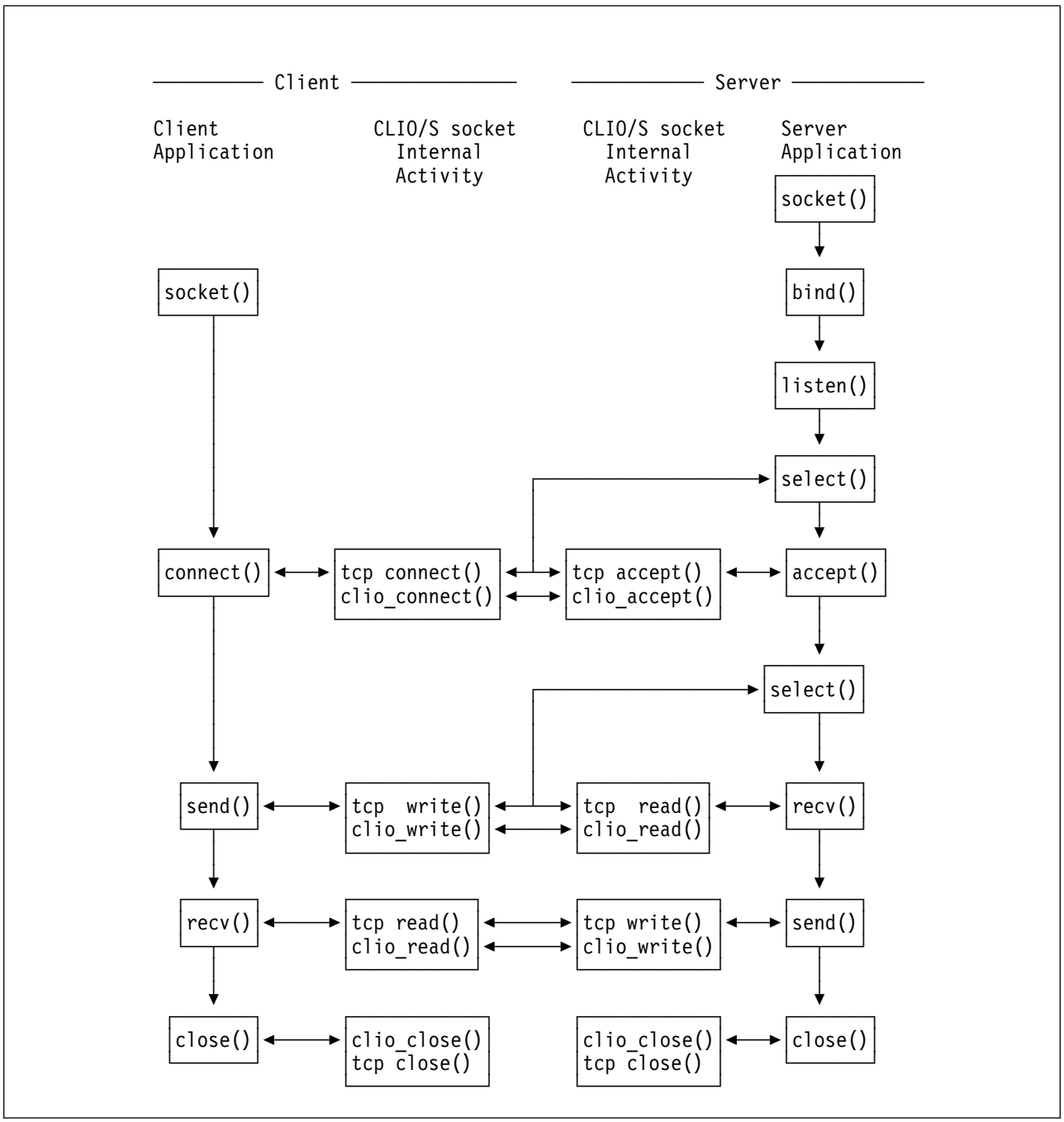


Figure 15. CLIO/S Sockets Interface Flow

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 124. 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. If the parent is closing the socket, to free the socket number for reuse in the parent process, a TCP/IP **close** must be issued, rather than a CLIO/S **close**. To do this, call TCP/IP **close** from the code that was not compiled with the **fcfsock.h** header file.
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 9. 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 10, “Using the Distributed Processing Interface (DPI)” on page 143.

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/fcf/inc/fcfcmi.h**

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

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	128
clio_async	Setting the Asynchronous Data Flag	129
clio_check	Checking the Number of Bytes Transmitted	129
clio_close	Closing a CLIO/S Connection	130
clio_connect	Requesting a CLIO/S Connection to Another Process	130
clio_create	Starting a Process on a Node	131
clio_end	Ending a CLIO/S Connection	132
clio_env	Getting the Environment	132
clio_exp	Setting the Environment	133
clio_h2i	Locating Machine Information Using an Internet Address	134
clio_init	Initializing CLIO/S	134
clio_kill	Stopping a Process on a Node	134
clio_l2n	Converting Data Representations, Local to Network	135
clio_n2l	Converting Data Representations, Network to Local	135
clio_m2i	Locating Machine Information Using a Machine Number	136
clio_n2h	Getting an Internet Address	136
clio_n2i	Locating a Machine by Symbolic Name	137
clio_n2m	Getting the Machine Information ID	137
clio_own	Allowing a Process to Change Ownership of a Connection	138
clio_pid	Returning a Process ID	138
clio_read	Reading Data over a CLIO/S Connection	139
clio_write	Writing Data over a CLIO/S Connection	139
tcp_accept	Accepting a TCP/IP Connection	140
tcp_connect	Establishing a TCP/IP Connection	140

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 RISC System/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, for example a Token Ring or Ethernet** connection and a SOCC connection. 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 137 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 29 discusses the environment variables used by CLIO/S. “clio_exp—Setting the Environment” on page 133 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 29 discusses the environment variables used by CLIO/S.

“clio_env—Getting the Environment” on page 132 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 B, “Generating JCL for Jobs Started by PARSRV” on page 221 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 RISC System/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 127 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_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 127 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 127 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 127 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 127 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 10. 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 9, "Using the Communication Manager Interface (CMI)" on page 127) 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/fcf/inc/fcfdpi.h**

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 127.

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

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 145. 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	150
ALERTGRP	ALRTGRP	Adding a Process ID to an Alert Group	151
cdpi_async	CDPIASY	Setting an Asynchronous Flag By Group	151
cdpi_check	CDPICHK	Checking Number of Bytes Transmitted by a Group	152
cdpi_s2c	N/A	Returning a CLIO/S Pointer	152
CLIOBLK	CLIOBLK	Setting the Buffer Bypass Limit	155
CLIOBUF	CLIOBUF	Setting the Size of the Internal Buffer	156
CLIODEB	CLIODEB	Masking Printed Status Messages	157
CLIOENV	CLIOENV	Getting Environment Information	157
CLIOMST	CLIOMST	Telling OPNSRV to Start Independent Masters	158
CLIOSOCK	N/A	Returning the Socket Descriptor	158
CLIOTBS	CLIOTBS	Changing Send and Receive Buffer Sizes	159
CLIOTMO	CLIOTMO	Setting the Time-out Period	159
CLSFAM	CLSFAM	Stopping All Processes in a Family	160
CLSPAR	CLSPAR	Closing the DPI Environment	160
CLSSRV	CLSSRV	Stopping the Servant Process	160
FNDMACH	FNDMACH	Locating a Machine by Symbolic Name	161
NMACHS	NMACHS	Counting Valid Entries in PARCONF	163
OPNFAM	OPNFAM	Opening a Family of Processes	164
OPNPAR	OPNPAR	Initializing the DPI Environment	164
OPNSRV	OPNSRV	Starting a Servant Process	165
QMACH	N/A	Getting Machine Information	166
QSLAVE	N/A	Getting Servant Machine Information	166
SCHEDUL	SCHEDUL	Scheduling a Task to Start	169

Table 8 (Page 2 of 2). DPI Master and Servant Control Subroutines

Subroutine			
C	**FORTRAN**	**Description**	**Page**
SETFAM	SETFAM	Setting the Family ID	170
SYNCHRO	SYNCHRO	Waiting Until All or Selected Tasks are Completed	173
TSKEXIT	N/A	Informing a Master the Servant Process is Finished	174
WAITFOR	WAITFOR	Waiting Until an Alert is Available	175
WAITREQ	WAITREQ	Indicating a Servant is Waiting for Work	175

Table 9. DPI Data Transfer and Conversion Subroutines

Subroutine			
C	**FORTRAN**	**Description**	**Page**
CLFLUSH	CLFLUSH	Clearing Buffers Written Using SPANY	153
clio_l2n	LN*xxxx*	Converting Data to Network Format	153
clio_n2l	NL*xxxx*	Converting Data to Local Format	154
MSANY	MS*xxxx*	Transferring Data from Master to Servant	161
MS_flush	MSFLSH	Confirming Data Was Sent from Master to Servant	163
RPANY	RP*xxxx*	Receiving Data from a Process	167
rw_dpi	N/A	Reading and Writing Data	168
SETSRV	SETSRV	Forcing Data Transfer to the Servant ID	170
SMANY	SM*xxxx*	Transferring Data from Servant to Master	171
SM_flush	SMFLSH	Confirming Data Was Sent from Servant to Master	172
SPANY	SP*xxxx*	Sending Data to a Process	172

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 RISC System/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 16 on page 147. 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 RISC System/6000-to-RISC System/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 RISC System/6000).
ESCON	Channel-to-Channel between MVS and a workstation with ESCON (ESCON to RISC System/6000).
SOC	Serial Optical Channel (SOCC) between workstations with Serial Optical Channel Converters (SOCC to SOCC).

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.

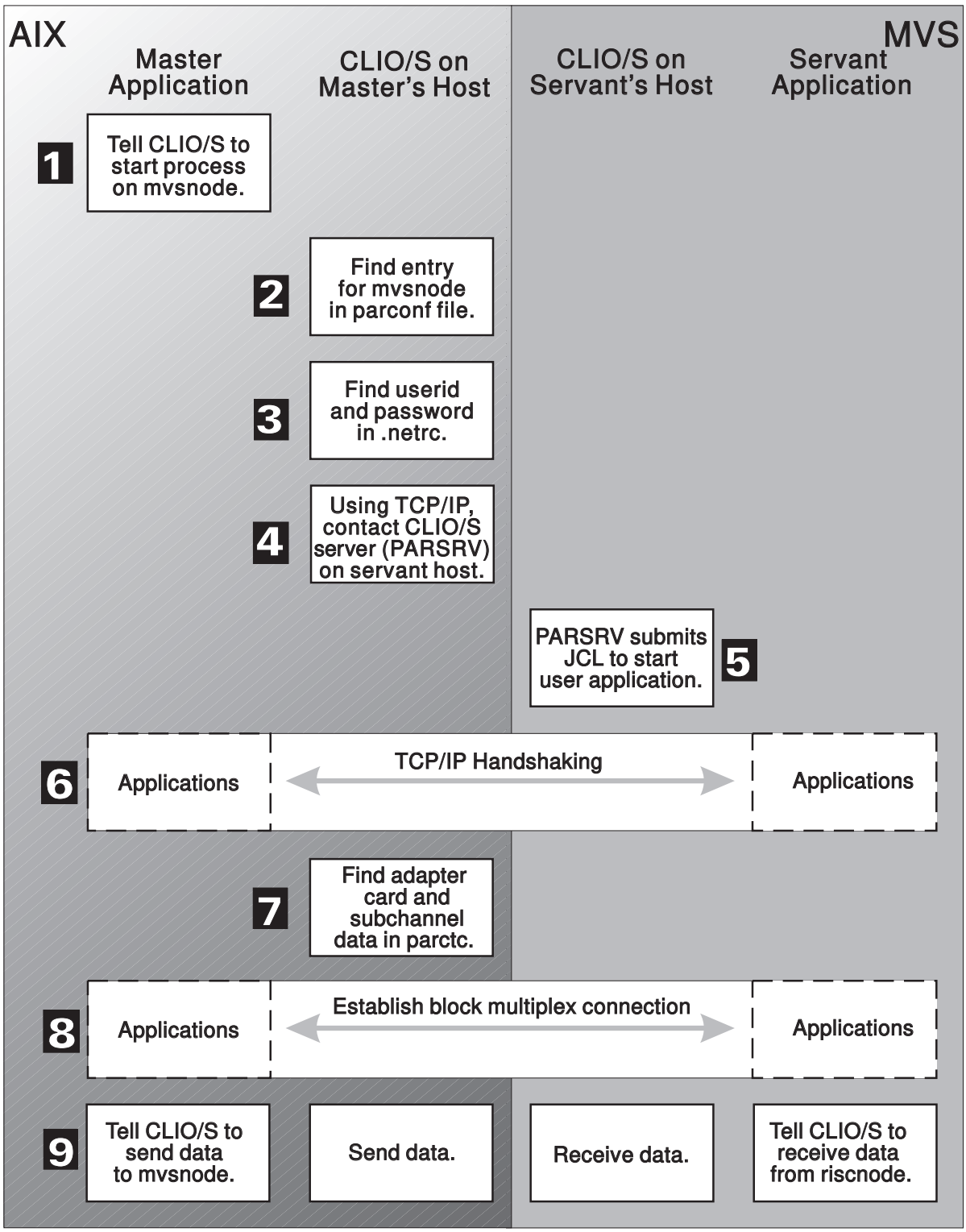


Figure 16. How the DPI Works

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 17. 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 18. 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 17 on page 148),
`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 required to start the new password. 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 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 B, “Generating JCL for Jobs Started by PARSRV” on page 221 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 17 on page 148), `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 129 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 135.

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 135.

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 176 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 176 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 29 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 133 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.

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 143.

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 178 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 175).

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***.

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 161 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 19, 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 19. 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 and SOCC connections and runs on the RISC System/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 152.

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 168 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:

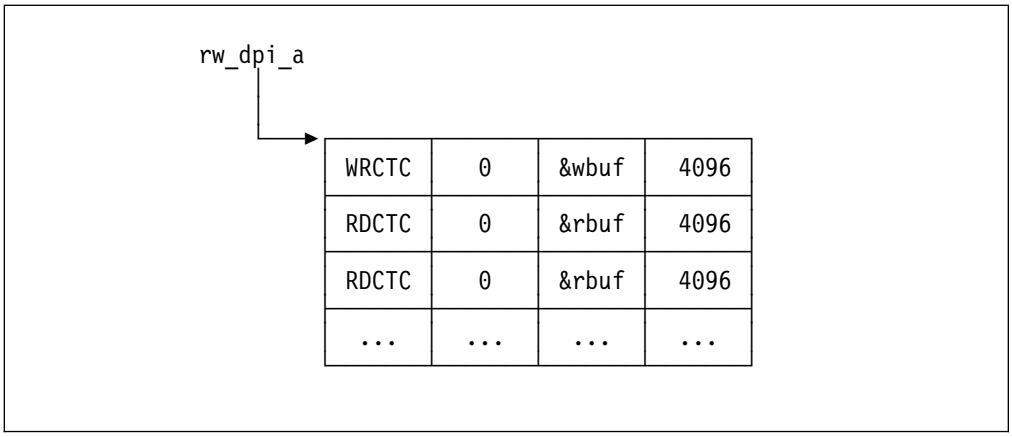


Figure 20. Example of Array of RW_DPI Structures for Chained Reads and Writes (WRCTC=write RDCTC=read)

Using the rw_dpi Call

Figure 21 on page 180 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 21. Sample Code for Using the rw_dpi Call.

Chapter 11. 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 RISC System/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 "FCFTTEST Sample Program" on page 199 for more information.

Chapter 4, "Configuring and Managing the CLIO/S Tape Pseudo Device Driver" on page 39 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 148 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 10, "Using the Distributed Processing Interface (DPI)" on page 143. 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

```
 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 22. 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	183

Subroutine			
C	**FORTRAN**	**Description**	**Page**
ctpclose	tpclose	Closing the Tape	186
ctpopen	tpopen	Opening the Tape Device for Use	187
ctpread	tpread	Reading the Tape	188
ctprew	tprew	Rewinding the Tape	189
ctpskpf	tpskpf	Skipping Tape Files	189
ctpskpr	tpskpr	Skipping Tape Records	190
ctpunall	tpunall	Stop the Remote Tape Server	191
ctpvols	tpvols	Getting a List of Tape Volumes	191
ctpweof	tpweof	Writing Tape Marks	192
ctpwrite	tpwrite	Writing to tape	193
ctpwrto	tpwrto	Writing to Operator	194

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 RISC System/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 24. 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 C, “Dynamic Device Allocation on MVS” on page 251.

```
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 197 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 196 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 RISC System/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 RISC System/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 195 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 197 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 195 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 195 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 195 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 195 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 195 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 195 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 197 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 195 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 195 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 195 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 200 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:

00 Called before allocation
01 Called before open
02 Called after close
03 Called after unallocation

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 C, “Dynamic Device Allocation on MVS” on page 251 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**/**sample** 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:

00 Called before allocation
01 Called before open
02 Called after close
03 Called after unallocation

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.
	Multivolume support is done using multiple opens and close, as shown in Figure 23 on page 199.	MVS automatically mounts a new tape when you reach the end of the current tape. A single open and close is required.

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 23. 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 RISC System/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.

FCFTTEST Sample Program

FCFTTEST is a program that reads C-like commands from standard input and interprets them as function calls to the CLIO/S tape interface. It is shipped with the CLIO/S sample programs, along with some sample input files and startup JCL for MVS and a makefile for AIX. On AIX, it is located in the file **/usr/lpp/fcf/samplet/fcfttest.c**. On MVS, it's in the data set

SYS1.SFCFSAMP(FCFTTEST). Comments at the top of FCFTTEST contain directions for using and modifying the sample.

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 4. Appendixes

© Copyright IBM Corp. 1993, 1995

Appendix A. 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. Be sure to change the default programs for both REXX and pseudo-JCL.

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	206
FCFIDUM	fcfidum	Verifying that CLIO/S can communicate between master and servant programs	207
FCFFCLPR, FCFFCLPW	clplink	Starting the CLPLINK program	209
FCFFPIPR, FCFFPIPW	fcffpipr, fcffpipw	Testing the CLIO/S Cross System Pipe Link	215

© Copyright IBM Corp. 1993, 1995

Table 10 (Page 2 of 2). CLIO/S Support Programs

This program on ...			
MVS	**AIX**	**Helps you with this task...**	**Page**
FCFPARDE	pardeb	Switching server debug flags	210
FCFPARIN	parinfo	Displaying server information	210
FCFPARPI	parping	Tests whether PARSRV is running	211
FCFPARRE	parrefr	Refreshing MVS installation defaults	212
FCFPARSR	parsrv	Starting a DPI server	213
FCFPARST	parstop	Stopping a DPI server	214
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	216
n/a	online	Starting subchannels on the workstation	218

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 24. 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

Refer to “Debugging the Installation Process” on page 263 if the program doesn't work between the mainframe and the workstation.

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 25. 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 106 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 215

```
//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 26. 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 27. 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 28. 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 29. 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 more 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 30. JCL for Testing If PARSRV is Running (FCFPARPI)

Refreshing PARSRV Defaults on MVS—FCFPARRE, parrefr

FCFPARRE causes PARSRV on MVS to reread, or *refresh*, the JCL generation defaults when using the pseudo-JCL language. FCFPARRE may be run from a workstation or submitted as an MVS job.

A PARSRV server on AIX does not need to refresh because it does not generate JCL; specifying an AIX node's name has no effect.

An MVS PARSRV using REXX to generate JCL does not need to be refreshed because PARSRV picks up changes to the FCFREXX program immediately. Only MVS PARSRVs that use the pseudo-JCL method of generating JCL are refreshed by FCFPARRE; PARSRVs using REXX to generate JCL are not affected.

FCFPARRE accepts two parameters:

nodename Specifies the name of the MVS node whose PARSRV should be refreshed.

"*filename***"** Specifies the file from which new defaults are read. On AIX, the file name must be enclosed in double quotation marks ("); escape the double quotes to prevent the shell from interpreting them.

If you omit *filename*, the current value of INSTDEF will be used.

To run FCFPARRE on AIX, use the command in **/usr/lpp/fcf/bin**:

```
parrefr nodename \"filename\"
```

For MVS, the sample JCL for starting FCFPARRE, provided in **SYS1.SFCFSAMP(FCFPARRE)**, is the following:

```
//FCFPARRE JOB 1,PP.NUMBER.5648.129,MSGLEVEL=1,REGION=4096K
//***---------------------------------------------------
//* CLIO/S - REFRESH INSTALLATION DEFAULTS FILE NAME
//*---------------------------------------------------
//FCFPARRE  EXEC PGM=FCFPARRE,PARM='MVSNODE'
//*FCFPARRE EXEC PGM=FCFPARRE,PARM='MVSNODE FILENAME'
//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
```

Figure 31. JCL for Refreshing Installation Defaults (FCFPARRE)

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 28. “Running Multiple PARSRVs” on page 35 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 30 through 32.

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 32. 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 33. 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 more 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 34. 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 35. 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 36. 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 RISC System/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 218) 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 RISC System/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 RISC System/6000 subchannel, in hexadecimal

risclast Is the device number of the last RISC System/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 RISC System/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 RISC System/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 18, the sample PARCONF file shown in "The PARCONF File" on page 24 and the sample PARCTC file shown in "The PARCTC File" on page 26.

Using FCFVARY

To start or stop subchannels on your RISC System/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 RISC System/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 37. 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 216. The **online** program is available only on the RISC System/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 B. Generating JCL for Jobs Started by PARSRV

This chapter describes two languages for customizing the JCL that is submitted to start servant process. 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.

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. The two languages available for creating JCL are:

- REXX is the preferred language for generating JCL. FCFPARSR calls a REXX program. You modify the REXX program to control the JCL that starts the servant job. "Using REXX/MVS to Generate PARSRV JCL" on page 223 describes how to use REXX to customize the JCL created by PARSRV.
- Pseudo-JCL is supported to ease migration from earlier version of CLIO/S. FCFPARSR interprets definition files containing the CLIO/S-specific pseudo-JCL language. FCFDEFS is the default definition file. "Using Pseudo-JCL to Generate PARSRV JCL" on page 235 describes how to customize the JCL generated by PARSRV with pseudo-JCL.

Generating JCL for a program is a fairly intricate process. If you install CLIO/S with the default data set names, the JCL generation programs in both REXX and pseudo-JCL will generate JCL that is sufficient for most programs. The sample FCFREXX and pseudo-JCL programs shipped with CLIO/S produce similar JCL.

Note: If you use a security product other than RACF, you need to make the appropriate changes to the default JCL generation programs. Be sure to change the default programs for both REXX and pseudo-JCL.

Specifying the JCL Generation Language

You control which language a PARSRV server uses to generate JCL by including or omitting the REXXJCL DD statement in the JCL that starts FCFPARSR. If there is a REXXJCL DD statement, the REXX program in the FCFREXX member of the partitioned data set named by the statement is run. If there is no REXXJCL DD statement, the pseudo-JCL method is used. The sample JCL for starting FCFPARSR includes the REXXJCL DD statement.

Note: The REXXJCL DD statement, if it exists, must point to a partitioned data set containing a member named FCFREXX.

You select the JCL generation method for each PARSRV server. If you have multiple PARSRVs running on an MVS system, one PARSRV can use the default REXX program, a second PARSRV can use a customized REXX program, a third PARSRV can use the default pseudo-JCL program, a fourth PARSRV can use a customized pseudo-JCL program, and so on.

© Copyright IBM Corp. 1993, 1995

Migrating From CLIO/S Version 1

This section describes considerations for customers migrating from CLIO/S Version 1. If you did not use CLIO/S Version 1, you can skip this section.

CLIO/S Version 1 provided only the pseudo-JCL language for JCL generation. CLIO/S Version 2 uses REXX programs, but supports the pseudo-JCL language to help you migrate from the earlier version.

Note

The CLIO/S pseudo-JCL language is supported for migration purposes. Support for it may be withdrawn.

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).

If your installation customized JCL generation by changing the FCFDEFS file in CLIO/S Version 1, you can continue to use your modifications while migrating your customized program from pseudo-JCL to REXX. Make sure the JCL that starts PARSRV does not 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 35 for more information.

Generating JCL for started tasks

Either method of generating JCL can start servant jobs as started tasks rather than submitted batch jobs. To do this, your program must generate JCL consists of a $VS statement starting the procedure, followed by parameters containing the address of the master process. The JCL generation programs FCFREXX (233) and FCFDEFS (249) contain examples of this.

The generated result looks like this:

```
/*$VS,'START PROCNAME.JCASEY,PRM=(4047,9.117.20.21)'
```

PROCNAME is a proclib member, and PRM is a variable in that proclib that is to pass the socket number (4047) and internet address (9.117.20.21) of the master process to the servant.

Because CLIO/S environment variables cannot be supplied to the servant process as usual when started tasks are used, the location of the master is passed in arguments. In addition, the servant program must explicitly make these arguments accessible to CLIO/S routines that need them. To do this, any C program that runs as a started task should have the following code at the beginning:

```
Clioargc = argc;   /* argc is argument count passed to main() */
Clioargv = argv;   /* argv is argument ptr passed to main()   */
```

Starting tasks requires a higher level of MVS authority than submitting batch jobs.

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 32 on page 213 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 33 on page 214.

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 29 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 29), as well as variables created by the master application with the **clio_exp** routine (see page 133).

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 38 on page 226 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 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.        */
   p.i = arg(i)
   end
call initialize                       /* Do some preparation.       */

/*********************************************************************
* Now set some variables that will be used during JCL generation.
* Change these as desired.
*********************************************************************/
account_info = '(572260,6MM,202-3)'
class        = 'CLASS=A'
region       = 'REGION=2M'
msgclass     = 'MSGCLASS=4'
msglevel     = 'MSGLEVEL=(1,1)'
notify       = 'NOTIFY=' || MVSUSER
time         = 'TIME=1000'
passwd       = 'PASSWORD=' || MVSPW
jobname = MVSUSER||substr(jobstring,(SRVNUM-1)//length(jobstring)+1,1)
                                      /* Choose jobletter based on  */
                                      /*   SRVNUM, append to userid */
                                      /*   to create jobname.       */
/*********************************************************************
* The following STEPLIB datasets should reflect your own installation.
*********************************************************************/
clioload = 'SYS1.SFCFLOAD'        /* CLIO/S load modules             */
cliosamp = 'SYS1.SFCFSAMP'        /* CLIO/S sample code              */
tapeload = 'SYS1.SFCFLOAD'        /* CLIO/S load modules             */
sedclink = 'SYS1.SEDCLINK'        /* C runtime libraries             */
seqalink = 'SYS1.SIBMLINK'        /* PL1 runtime libraries           */
samprun2 = 'VSPASCAL.VSPV1R2.SAMPRUN2'  /* Pascal runtime libraries  */
vsf2load = 'SYS1.VSF2LOAD'        /* Fortran runtime libraries       */
```

Figure 38 (Part 1 of 10). 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.)
*
* In this sample code, different methods of filling the stem variables
* with JCL statements are demonstrated.
*
* The JCL produced closely matches that produced by the older CLIO/S
* JCL generation routine.
*
* 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.
*
*********************************************************************/
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 == 'C*A*N'    /* CLIO/S Cancel routine.              */
     then call fcfcanc

  otherwise call default     /* Default JCL.                        */
  end;                       /* end select                          */

/*********************************************************************
*
* Write the JCL statements into the JES internal reader.
*
*********************************************************************/
'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.                                     */
exit
```

Figure 38 (Part 2 of 10). Sample JCL Generation Program using REXX (FCFREXX)

```
/*********************************************************************/
default:
/*********************************************************************
*
* This subroutine calls another subroutine to store the JCL strings.
* The 'j' subroutine maintains it's own count, so we don't have to.
*
*********************************************************************/
call j("//"jobname  "JOB" account_info",'"MVSUSER"',"class","region",")
call j("//   "msgclass","msglevel","notify",")
call j("//   "time","passwd)
call j("//"SRVPROG"   EXEC PGM="SRVPROG",")
call j("// PARM='"SRVPARMS"'")
call j("//STEPLIB  DD   DISP=SHR,DSN="clioload)
call j("//         DD   DISP=SHR,DSN="sedclink)
call j("//         DD   DISP=SHR,DSN="samprun2)
call j("//         DD   DISP=SHR,DSN="vsf2load)
call j("//SYSPRINT DD   SYSOUT=*")
call j("//SYSUDUMP DD   SYSOUT=*")
call j("//SYSDUMP  DD   SYSOUT=*")

if datatype(start_env, 'W') = 1        /* Any environment variables? */
   then do
   call j("//CLIOENV  DD   *")         /* Create CLIOENV DD card,    */
   do i = start_env to numargs         /*   write variables into it. */
      call j(p.i)
      end i
   call j("/*")                        /* Terminate the DD card.     */
   end
call j("//* JCL generated by FCFREXX") /* Information only.          */

call print_jcl                  /* Copy JCL to SYSTSPRT, if desired. */

return
```

Figure 38 (Part 3 of 10). Sample JCL Generation Program using REXX (FCFREXX)

```
/*********************************************************************/
fcffserv:
/*********************************************************************
*
* This subroutine determines the position of each JCL string explicitly.
* Be sure to keep the numbering correct if this code is modified.
*
*********************************************************************/

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). */
jcl.1  = "//"jobname  "JOB" account_info",'"MVSUSER"','"class","
jcl.2  = "//   REGION=17M,"msgclass","msglevel","notify","
jcl.3  = "//   "time","passwd
jcl.4  = "//"SRVPROG"  PROC"
jcl.5  = "//"SRVPROG"  EXEC PGM=IKJEFT01"
jcl.6  = "//STEPLIB  DD   DISP=SHR,DSN="clioload
jcl.7  = "//SYSIN    DD   DUMMY"
jcl.8  = "//INSPLOG  DD   SYSOUT=4,DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)"
jcl.9  = "//REXXLST  DD   DSN="cliosamp",DISP=SHR"
jcl.10 = "//SYSUDUMP DD   SYSOUT=*"
jcl.11 = "//SYSPRINT DD   SYSOUT=*"
jcl.12 = "//SYSTSPRT DD   SYSOUT=4,DCB=BLKSIZE=3429"
jcl.13 = "//    PEND"
jcl.14 = "//P1  EXEC "SRVPROG
jcl.15 = "//SYSTSIN  DD   *"
jcl.16 = "   CALL '"clioload"("SRVPROG")' '"SRVPARMS"'"

jcl_lines = 16
call add_env                    /* Add environment variables */
call print_jcl                  /* Copy JCL to SYSTSPRT, if desired. */

return
```

Figure 38 (Part 4 of 10). Sample JCL Generation Program using REXX (FCFREXX)

```
/*********************************************************************/
clplink:
/*********************************************************************
*
* This subroutine determines the position of each JCL string explicitly.
* Be sure to keep the numbering correct if this code is modified.
*
*********************************************************************/
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). */
jcl.1  = "//"jobname  "JOB" account_info",'"MVSUSER"',"class","
jcl.2  = "//   REGION=17M,"msgclass","msglevel","notify","
jcl.3  = "//   "time","passwd
jcl.4  = "//"SRVPROG"  EXEC PGM="SRVPROG","
jcl.5  = "// PARM='"SRVPARMS"'"
jcl.6  = "//STEPLIB  DD   DISP=SHR,DSN="clioload
jcl.7  = "//SYSUDUMP DD   SYSOUT=*"
jcl.8  = "//SYSPRINT DD   SYSOUT=*"
jcl_lines = 8

if CLPLINKX <> "CLPLINKX" 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_env                    /* Add environment variables */
call print_jcl                  /* Copy JCL to SYSTSPRT, if desired. */

return
```

Figure 38 (Part 5 of 10). Sample JCL Generation Program using REXX (FCFREXX)

```
/*********************************************************************/
clftp:
/*********************************************************************
*
* This subroutine determines the position of each JCL string explicitly.
* Be sure to keep the numbering correct if this code is modified.
*
*********************************************************************/
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). */
jcl.1  = "//"jobname  "JOB" account_info",'"MVSUSER"','class","
jcl.2  = "//   REGION=17M,"msgclass","msglevel","notify","
jcl.3  = "//   "time","passwd
jcl.4  = "//"SRVPROG"  EXEC PGM="SRVPROG","
jcl.5  = "// PARM='"SRVPARMS"'"
jcl.6  = "//STEPLIB  DD   DISP=SHR,DSN="clioload
jcl.7  = "//SYSUDUMP DD   SYSOUT=*"
jcl.8  = "//SYSPRINT DD   SYSOUT=*"
jcl_lines = 8

call add_env                    /* Add environment variables */
call print_jcl                  /* Copy JCL to SYSTSPRT, if desired. */

return
```

Figure 38 (Part 6 of 10). Sample JCL Generation Program using REXX (FCFREXX)

```
/*********************************************************************/
tapesrv:
/*********************************************************************
*
* This subroutine reads the REXX source for comments containing the
* JCL statements, then substitutes !-delimited variables with their
* values. In this implementation '!' can only be used a delimiter,
* and there can be no comment delimiters in the jcl.
*
*********************************************************************/
jcl_in = _sigl() + 1    /* _sigl returns line number of caller       */
/** start of comment ************************************************
//!jobname!  JOB !account_info!,!MVSUSER!,!class!,!region!,!
//   !msgclass!,!msglevel!,!notify!,
//   !time!,!passwd!
//!SRVPROG!   EXEC PGM=!SRVPROG!,PARM='NOSTAE,NOSPIE'
//STEPLIB  DD   DISP=SHR,DSN=!clioload!
//         DD   DISP=SHR,DSN=!sedclink!
//         DD   DISP=SHR,DSN=!samprun2!
//SYSUDUMP DD   SYSOUT=*,DCB=BLKSIZE=3429
//SYSPRINT DD   SYSOUT=*
*** end of comment ************************************************/

line = sourceline(jcl_in)              /* Read the FCFREXX source.   */
if pos('/*', line) = 0 then say 'Comment expected'

maxlines = sourceline()                /* Last line in program.      */
do until jcl_in >= maxlines            /* Stop at end of file.       */
   jcl_in = jcl_in + 1
   line = strip(sourceline(jcl_in), 'T')
   if pos('*/', line) <> 0 then leave  /* Stop at end of comment.    */
   if pos('!', line) <> 0 then line = change(line)
   jcl_lines = jcl_lines + 1
   jcl.jcl_lines = line
   end

if datatype(start_env, 'W') = 1        /* Any environment variables? */
   then do
   jcl_lines = jcl_lines + 1
   jcl.jcl_lines = "//CLIOENV  DD   *" /* Create CLIOENV DD card,    */
   do i = start_env to numargs         /*   write variables into it. */
      jcl_lines = jcl_lines + 1
      jcl.jcl_lines = p.i
      end i
   jcl_lines = jcl_lines + 1
   jcl.jcl_lines = "/*"                /* Terminate the DD card.     */
   end

jcl_lines = jcl_lines + 1
jcl.jcl_lines = "//* JCL generated by FCFREXX"  /* Info only.        */

call print_jcl                 /* Copy JCL to SYSTSPRT, if desired. */

return
```

Figure 38 (Part 7 of 10). Sample JCL Generation Program using REXX (FCFREXX)

```
/*********************************************************************/
fcfcanc:
/*********************************************************************
* This subroutine determines the position of each JCL string explicitly.
* Be sure to keep the numbering correct if this code is modified.
*
* Notes about FCFCANC:
*    -  Default MSGCLASS is Z, meaning job output is discarded
*    -  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.
*********************************************************************/
msgclass = 'MSGCLASS=Z'
jcl.1  = "//FCFCANC JOB" account_info",'"ID"',"class","region","
jcl.2  = "//   "msgclass","msglevel","notify","
jcl.3  = "//   "time","passwd
jcl.4  = "//FCFCANC   EXEC PGM=IKJEFT01"
jcl.5  = "//SYSPROC  DD   DISP=SHR,DSN="cliosamp
jcl.6  = "//SYSTSPRT DD   SYSOUT=*"
jcl.7  = "//SYSTSIN  DD   *"
jcl.8  = "%FCFCANC" SRVPARMS
jcl.9  = "/*"

jcl_lines = 9

jcl_lines = jcl_lines + 1
jcl.jcl_lines = "//* JCL generated by FCFREXX"  /* Info only.        */

call print_jcl                 /* Copy JCL to SYSTSPRT, if desired. */

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.
*
*********************************************************************/
jcl.1  = "//FCFSTASK JOB" account_info",'"ID"',"class","region","
jcl.2  = "//   "msgclass","msglevel","notify","
jcl.3  = "//   "time","passwd
jcl.4  = "/*$VS,'S "SRVPROG"."MVSUSER",PRM=("MstPort","MstHid")'"

/* Started tasks don't get the CLIOENV DD card. See User's Guide.    */

call print_jcl                 /* Copy JCL to SYSTSPRT, if desired. */

return
```

Figure 38 (Part 8 of 10). Sample JCL Generation Program using REXX (FCFREXX)

```
/*********************************************************************/
_sigl: return sigl   /* get the source line number of calling line    */
/*********************************************************************/
change:  /* Substitute !-delimited varaibles with their values.       */
parse arg line
do until pos('!', line) = 0
   parse var line first '!' var '!' rest
   if var <> '' then line = first || value(var) || rest
   else line = first
   end
return line

/*********************************************************************/
add_env:   /* Add environment variables to jcl.                       */
/*********************************************************************/
if datatype(start_env, 'W') = 1        /* Any environment variables? */
   then do
   jcl_lines = jcl_lines + 1
   jcl.jcl_lines = "//CLIOENV  DD   *" /* Create CLIOENV DD card,    */
   do i = start_env to numargs         /*   write variables into it. */
      jcl_lines = jcl_lines + 1
      jcl.jcl_lines = p.i
      end i
   jcl_lines = jcl_lines + 1
   jcl.jcl_lines = "/*"                /* Terminate the DD card.     */
   end

jcl_lines = jcl_lines + 1
jcl.jcl_lines = "//* JCL generated by FCFREXX"  /* Info only.        */

return

/*********************************************************************/
print_jcl: /* Copy JCL to 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 38 (Part 9 of 10). Sample JCL Generation Program using REXX (FCFREXX)

```
/*********************************************************************/
initialize:
/*********************************************************************/

parse source . . . . . environment . /* Just for information,      */
                                     /*    could be 'MVS' or 'TSO'.*/
jcl_lines = 0
jobstring = 'BCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'
do i = 1 to numargs                  /* Process the arguments.     */
   if substr(p.i, 1, 7) == 'MstPort' /* Start of Environment data, */
      then start_env = i             /*    capture for later use.  */
   interpret p.i                     /* Execute the parameter,     */
                                     /*    if desired.             */
   end
return
```

Figure 38 (Part 10 of 10). Sample JCL Generation Program using REXX (FCFREXX)

Using Pseudo-JCL to Generate PARSRV JCL

The pseudo-JCL method of generating JCL for servant processes uses a special language, whose constructs are similar to those of JCL itself (hence the name *pseudo-JCL*). This section describes CLIO/S's pseudo-JCL language.

Pseudo-JCL provides upward compatibility for customers migrating from CLIO/S Version 1. IBM recommends that you not write new JCL-tailoring code in this language and that you convert existing code to REXX.

Note

The CLIO/S pseudo-JCL language is supported for migration purposes. Support for it may be withdrawn.

Specifying Pseudo-JCL Method

To use the pseudo-JCL method of generating JCL for servant processes, make sure the JCL that starts FCFPARSR does not contain a REXXJCL DD statement. The default JCL for FCFPARSR includes the REXXJCL DD statement; you must modify the default JCL to remove it.

The DEFAULTS DD statement in the JCL that starts FCFPARSR specifies the data set that contains the pseudo-JCL program that control JCL generation. If your JCL contains both REXXJCL and DEFAULTS DD statements, PARSRV will use REXXJCL.

Overview

Customization, such as job cards, data set naming standards, etc., can be taken care of by inspecting the input file read by PARSRV at startup (DD statement DEFAULTS). The defaults provided by PARSRV can be easily modified by the installation. If desired, these defaults can further be modified based on program name or MVS user ID. Further, unless disabled by the installation, one or more libraries may be searched for explicit JCL for a particular program. Explicit JCL may be general in the sense that it can contain variables.

The main input file has three primary functions:

- Define layout of default JCL
- Determine which files are scanned for user defaults
- Specify how to locate specific JCL.

PARSRV learns about the various defaults by scanning files that contain assignment statements. At startup, it reads the variable INSTDEF, which names the file containing installation defaults. The default value of INSTDEF is `dd:DEFAULTS`. It allows the installation to specify the general layout of the JCL to be produced, to specify which files, if any, hold user defaults, where specific JCL is located, and so on. See "Sample FCFDEFS file" on page 243 for an example of the INSTDEF file.

Further, for each program that is to be started, the following steps are taken:

1. Read the file(s) **$(USERDEF)**, where the user can set his own defaults.
2. Process the first existing file in **$(USERJCL)**. This file normally contains JCL for a particular program, but can also contain instructions as in **$(USERDEF)**.
3. If step 2 produced no JCL, expand the sections mentioned in **$($(PROG).SECTIONS)**.
4. If **$($(PROG).CLIOENV)** has value the YES, expand a special section which allows a servant to contact its master.

Note: The variables USERDEF and USERJCL may hold more than one data set name. The names must be separated by blanks.

Specification

The INSTDEF, USERDEF and USERJCL files hold assignment statements, keyword statements and plain text. The latter is scanned for variables and added to the JCL. The keywords are:

EXPAND sections	Expand the sections mentioned
.EXPAND	Same as EXPAND $($(PROG).SECTIONS)
.INCLUDE file(s)	Include the files mentioned
.INLINE delim	Start an inline section
.INLINE	Same as INLINE $(.DELIM)
delim	End of inline section (default /*PROCESS*/)
.ECHO line	Echo line to stderr
.DEBUG flag(s)	Update debug flags

Any output that would be produced by INSTDEF is suppressed. The **USERJCL** file (and files included by it) is entered with INLINE active.

In an inline section, assignment and keyword statements are treated as plain text. Variables are expanded unless the value of **$(INLINE)** is NOSUBST. The inline section ends when the delimiter is found on a line by itself, starting in column one.

To prevent recursion, .INCLUDE of a file that is active will be ignored. The offending list of files is written to **stderr**.

The possible arguments of the .DEBUG keyword are:

(NO)JOB	Print a message when submitting a job
(NO)ECHO	Echo JCL to stdout

(NO)SUBST	Indicate how variables are substituted
(NO)CURFILE	Print a message when opening a file
OFF	Turn off JOB, ECHO, SUBST and CURFILE (default)

Variables

Variables are referenced by surrounding the name with $(and). Any spaces will be taken as part of the name. The name may itself be specified by a variable, for example, $($(VAR)EXT). Almost any character is permitted in a name. Names are case sensitive. It is possible to define a variable in a way that leads to recursion. This is detected, and the value returned is **RECUR**. If the value of a variable is a valid arithmetic expression, it is replaced by the numerical value of the expression. Undefined variables return an empty string.

Prefixing the name with a question mark (?) when referencing a variable will return the first blank-delimited substring of the value of the variable. This is useful when assigning defaults to data set names. For an example, see the definition of PROGLIB in Figure 39 on page 244.

Sections

A section is a group of JCL statements. Inline sections are defined by using the .INLINE keyword. All other sections have a name. Multiple lines can be assigned to the name by treating it as a stem variable, i.e. by appending .1, .2, etc. to the name and assigning values to the resulting variables. Several examples are shown below. Note: Many sections have default values. If name.1 is redefined, the whole section is redefined. In some cases it may be desirable to redefine only parts of the section and keep the rest. This can be achieved by treating the desired component, for example, name.2, as a stemmed variable and assigning values to name.2.1, name.2.2, etc.

Examples: Define a new section SECT:

```
SECT.1 = "//DD1      DD   DISP=SHR,DSN=$(USER).NEWSAMP.LIB"
SECT.2 = "//DD2      DD   DISP=SHR,DSN=$(USER).SAMPLE.LIB"
```

Redefine STEPLIB.1 and keep STEPLIB.2, STEPLIB.3 etc.

```
STEPLIB.1.1 = "//STEPLIB  DD   DISP=SHR,DSN=$(USER).TEST.LOAD"
STEPLIB.1.2 = "//         DD   DISP=SHR,DSN=$(PROGLIB)"
```

Special sections

STC	Can be used for started tasks
CLIOENV	Enables a servant to contact master

Assignments

The format of an assignment statement is:

```
lhs = rhs
```

Leading and trailing spaces surrounding *lhs* and *rhs* are removed. Any variables in *lhs* are expanded. Variables in *rhs* are expanded unless they are surrounded by quotes. One layer of quotes is stripped off. Quotes can be single or double. Single quotes inside double quotes, and vice versa, or quotes inside parentheses are treated like a normal character. Unbalanced quotes are retained. The consequence of this is that the evaluation of variables inside quotes is delayed until the new variable is expanded, effectively turning it into a macro. This is very useful

when defining defaults. A semicolon can be used to separate multiple assignment statements on the same line.

Examples

```
c = "'name.tst$(a)'"   # Value of $(c) is 'name.tst'
a = 5; b = $(a) + 1    # Value of $(b) is 6
a$(b) = $(a) * $(b)    # Value of $(a6) is 30
a = 7+1                # Value of $(c) is 'name.tst8'
a = $(a)               # Value of $(c) is 'name.tst8'
a = '$(a)'             # Value of $(c) is 'name.tst**RECUR**'
t = Don't remove quote # The quote is kept
a = first second
t = $(?a)              # Value of $(t) is first
t = $(a)               # Value of $(t) is first second
```

Special variables: There are some special variables:

- **PROJECT** - Used to construct file names.

  For user files, it can be overridden by setting $(PROG).PROJECT to the desired value in INSTDEF. Default is CLIO.

- **SECTIONS** - Specifies sections to be expanded.

  This variable is used to define which sections are expanded by an explicit .EXPAND (with no args) or when producing default JCL (as in step 3 on page 236). Default is START JES JOB EXEC STEPLIB SYSPRINT END.

- **$(PROG).SECTIONS** - Specify JCL sections for $(PROG).

  When default JCL is produced (in step 3 on page 236). it is checked if this variable exists. If so, it is used instead of $(SECTIONS).

  For an example, see the definition of TAPESRV.SECTIONS in Figure 39 on page 244.

- **CLIOENV**

  If YES, append the special section CLIOENV at the end of the job. This section allows the servant to contact its master. Initialized to YES by PARSRV.

- **$(PROG).CLIOENV** - Override $(CLIOENV) for $(PROG).

- **INLINE** - Affects variable expansions in inline sections

  ```
  SUBST   - variables are expanded (default)
  NOSUBST - variables are not expanded
  ```

- **CURFILE** - Name of the file currently being processed

- **USER** - Current MVS userid

- **PW** - Current MVS password

- **PROG** - Used to select program to be run

- **$PARM** - Holds argument string from OPNSRV or clio_create.

- **APPL** - Set to CLIO by PARSRV.

- **JOBLETTERS** -

  Defines the set of characters that are appended to $(USER) to produce the jobname.

- **JC** - Current element of $(JOBLETTERS)
- **MSTPORT** - Master's TCP/IP port number
- **MSTHID** - Master's TCP/IP internet address
- **MSTPID** - Master's Process ID.
- **MSTUSER** - Master's User ID.
- **DATE** - Current date (*day dd/mm/yy*)
- **TIME** - Current time (*hh:mm:ss*)
- **NAME** - May hold the name of the current user.

  Default is "$(USER)" or "$($(USER).NAME)" if it exists.
- **$(USER).NAME** - Specify name corresponding to $(USER)

Customizing Your INSTDEF File

The following sections contains examples of modifying the INSTDEF file to customize JCL generation for your programs. The following tasks are demonstrated:

- Modifying the job name
- Defining and modifying sections
- Adding additional job steps
- Making jcl globally available
- Starting a process as a started task
- Passing environment variables to a servant

Modifying the Job Name

The default job name given to a job submitted by PARSRV is created by appending a job letter to the MVS user ID. Since the default pool of possible job letters consists of the string “BCDEFGHIJKLMNOPQRTUVWXYZ0123456789”, this naming convention allows each user to have a maximum of 35 jobs active at any one time.

You can modify the INSTDEFS file to create jobs names that identify the master process and to allow a maximum of 35 active jobs for each master (instead of 35 for each user). The modified lines could be included in the INSTDEFS file named by the DD: DEFAULTS statement in the JCL used to start PARSRV, or in the USERDEFS file named by the USERDEF variable.

Replace the line in FCFDEFS (see page 245) that defines the job name:

```
$NAME    ="$(USER)$(JC)"
```

with the following lines:

```
TMP      = '-2128057592'           # Address of riscnode1
NN.$(TMP)= 'R1'                    # Identifier for riscnode1
TMP      = '-2128082132'           # Address of riscnode2
NN.$(TMP)= 'R2'                    # Identifier for riscnode2
NN       = '$(NN.$(MstHid)) XX'    # NN = 'R1 XX' or 'R2 XX'
                                   # or just 'XX' if MstHid unknown
$NAME    = $(?NN)$(MSTPORT)$(JC)   # Job name = R11494B or similar,
                                   #    XX1494B if workstation unknown
```

TMP is an arbitrary variable used to hold the decimal representation (-2128057592) of the internet address of a workstation (in this example, 129.40.107.8). Then the expression NN.$(TMP)= 'R1' associates a two-character nickname (R1) with the internet address (the value of $(TMP)). This technique is similar to the associative arrays available in the REXX language. Thus the first two lines:

```
TMP      = '-2128057592'           # Address of riscnode1
NN.$(TMP)= 'R1'                    # Identifier for riscnode1
```

are equivalent to the assignment:

```
NN.-2128057592 = 'R1'
```

Repeat the process to assign the nickname R2 to another workstation.

MstHid, a variable made available by CLIO/S, contains the master's internet address in decimal format. Thus, in the assignment:

```
NN       = '$(NN.$(MstHid)) XX'
```

NN becomes 'R1 XX', 'R2 XX', or 'XX', depending on which workstation (R1, R2, or an unknown one) is the master.

Finally, $NAME receives the job name in the assignment:

```
$NAME    = $(?NN)$(MSTPORT)$(JC)
```

which is resolved as follows:

$(?NN) Is the first word of the string in variable NN, namely R1, R2, or XX.

$(MSTPORT) Is the socket number that the master obtained for communicating with PARSRV, and that the servant will eventually use to communicate back to the master. Since the socket number is unique for a particular workstation, the nickname-socket combination will uniquely identify the master.

$(JC) Appends a job letter, allowing the master to have up to 35 servants simultaneously.

Defining and Modifying Sections

The default JCL is specified by the following lines in $(INSTDEF):

```
######## Define the default sections
# SECTIONS = START JOB JES EXEC STEPLIB SYSPRINT END

######## This is the default JCL produced
# JOB.1 = "//$($NAME) JOB ($($ACCT)),'$($PID)',$($CLASS)$($REGION)"
# JOB.2 = "//   $($MSGCLASS)$($MSGLEVEL)$($NOTIFY)$($TYPRUN)"
# JOB.3     = "//   $($TIME)$($USER)$($PW)"
# EXEC.1    = "//$(PROG)  EXEC PGM=$(PROG)$(COND)$(PARM)"
# STEPLIB.1 = "//STEPLIB  DD   DISP=SHR,DSN=$(?PROGLIB)"
# STEPLIB.2 = "//         DD   DISP=SHR,DSN=$(SEDCLINK)"
# STEPLIB.3 = "//         DD   DISP=SHR,DSN=$(SAMPRUN2)"
# STEPLIB.4 = "//         DD   DISP=SHR,DSN=$(VSF2LOAD)"
# SYSPRINT.1=  //SYSPRINT DD   SYSOUT=*
# SYSPRINT.2=  //FT06F001 DD   SYSOUT=*
```

The number symbol (#) shows these as comment lines. They are left as comments because they are identical to the built-in defaults.

Some of the lines in the file are enclosed in double quotes ("). This ensures that the variables are evaluated at JCL generation time.

Three of the sections (START, JES, END) are not defined in the sample file. To insert statements between JOB and EXEC, it is only necessary to define JES, for example:

```
JES.1 = "//*"
JES.2 = "//*    Submitted at $(TIME) on $(DATE)."
JES.3 = "//*"
```

This can be done in any of the files INSTDEF, USERDEF, or USERJCL. A section cannot be deleted, but it can be overridden or modified. If the first (JES.1) is redefined, the whole section (JES) is redefined. If any of the other (JES.2) is redefined, it does not affect JES.1, JES.3 etc. To modify JES.1 and still keep JES.2 etc., type something like:

```
JES.1.1 = "//* Modified"
JES.1.2 = "//* This line added before JES.2"
```

The same technique can be used to add another line between JES.2 and JES.3

```
JES.2.1 = "$(JES.2)"
JES.2.2 = "//* This line added before JES.3"
```

Adding Additional Job Steps

If a jobstep is required after the main step of a servant job, it is necessary to ensure that the CLIOENV DD statement is placed in the main step. In explicit JCL, this may be achieved as follows:

```
//MYJOB    JOB .....
//MAINSTEP EXEC PGM=SERVANT
//STEPLIB  DD   DISP=SHR,DSN=...
//SYSPRINT DD   SYSOUT=*
/*PROCESS*/
.EXPAND CLIOENV        # Expand the CLIOENV section here
.INLINE
//LASTSTEP EXEC PGM=SECOND
//STEPLIB  DD   DISP=SHR,DSN=...
//SYSPRINT DD   SYSOUT=*
```

Another approach, if the default JCL is acceptable for each step, could be

```
/*PROCESS*/
.EXPAND $(SECTIONS) CLIOENV  # Build main step
JOB.1 = "//* STEP TWO"       # Deactivate the JOB section
PROG  = SECOND               # Name of second program
.EXPAND                 # Build second step, ( no CLIOENV statement)
```

If the installation wants an extra step added to most (all) jobs, it might consider something like

```
SECTIONS = START JOB JES EXEC STEPLIB SYSPRINT CLIOENV END EXTRA
EXTRA.1  = "//STEP2   EXEC PGM=SECOND,PARM='$(USER) $(PROG)'"
EXTRA.2  = "//STEPLIB DD   DISP=SHR,DSN=..."
```

Making JCL Globally Available

It is likely that some (most) programs are to be available to more than one user. There are two ways to make JCL for these programs globally available. Assume one such program is GLOBAL.

1. Set GLOBAL.SECTIONS = JOB GLOBAL (if the default JOB card is OK). Set GLOBAL.1, GLOBAL.2 etc. to hold the appropriate JCL skeleton.
2. Create a PDS data set, for example, SYSTEM.JCL. Create and edit a member GLOBAL in SYSTEM.JCL, so that it specifies the JCL required for program GLOBAL. Extend the definition of USERJCL to include "SYSTEM.JCL($(PROG))".

Starting a Process as a Started Task

If you want to start a servant as a started task instead of as a batch program, the internal reader must have the appropriate authorization (see the $TRDI JES2 command). The CLIO definitions might be as follows:

```
MYSTC.SECTIONS = STC; MYSTC.CLIOENV = NO;
```

To run a program as a started task instead of as a batch job, you create a command similar to the following. The sample FCFDEFS file has this command commented out in the STC.1 section.

```
$VS,'S $(PROG).$(USER),PRM=($(MSTPORT),$(MSTHID))'
```

If you want to run a servant as a started task, JCL for that servant must exist in the system PROCLIB concatenation. For example, add the following JCL to let the FCFIDUM servant run as a started task:

```
//FCFIDUM  PROC PRM=
//*-------------------------------------------------------------
//* FUNCTION:
//*   STARTED PROGRAM FCFIDUM.
//*
//* INSTRUCTIONS:
//*   1. IF NECESSARY, CHANGE SYS1.SFCFLOAD TO BE THE NAME OF THE
//*      FILE CONTAINING THE FCFIDUM LOAD MODULE.
//*   2. CHANGE USER.CLIOENV TO BE THE NAME OF THE FILE CONTAINING
//*      CLIOENV VARIABLES.
//*-------------------------------------------------------------
//FCFIDUM  EXEC PGM=FCFIDUM,PARM=&PRM
//STEPLIB  DD DSN=SYS1.SFCFLOAD,DISP=SHR
//CLIOENV  DD DSN=USER.CLIOENV,DISP=SHR
//SYSPRINT DD SYSOUT=4
//   PEND
```

In addition, any program that runs as a started task must set the Clioargc and Clioargv global variables to contain the argc and argv values passed into that program. This C code segment from the FCFIDUM program shows how to do that:

```
#include <fcfcmi.h>

main(argc, argv)
int argc;
char *argv[];
{

#if defined(MVS)
  Clioargc = argc;
  Clioargv = argv;
#endif
        .
        .
        .
```

Passing Environment Variables to a Servant

A master program starting a servant program on MVS can pass values to JCL generation using the **clio_exp** function. All variables created using **clio_exp** are available during JCL generation. For example, assume the JCL generation contains this line:

```
EXEC.1 = "//$(VAR1) EXEC PGM=$(VAR1)$(PARM)"
```

The master program could set the $(VAR1) value for JCL generation using this call to the **clio_exp** function:

```
clio_exp("VAR1","PROG1");
```

If a master program wanted to set the END section for JCL generation to contain a new DD statement for input, it could set values as follows.

```
clio_exp("END.1","//INPFILE DD *");
clio_exp("END.2","Line 1 of input");
clio_exp("END.3","Line 2 of input");
clio_exp("END.4","/*");
```

The above statements would then replace the END section during JCL generation.

Sample FCFDEFS file

The installation defaults (INSTDEF) file is used for generating JCL in the pseudo-JCL language. This file is shipped in **SYS1.SFCFSAMP(FCFDEFS)**. Figure 39 on page 244 shows the sample code. The system administrator who installs CLIO/S on MVS must modify this file. Instructions for modifying it are contained in comments in the program and in the Program Directory.

```
# ****************************************************************
# When PARSRV starts a program on MVS the following steps are taken:
#
# 1. Read the file(s) $(USERDEF), where the user can set his own defaults.
# 2. Process the first existing file in $(USERJCL). This file normally
#    contains JCL for a particular program, but can also contain
#    instructions as in $(USERDEF).
# 3. If step 2 produced no JCL, expand the sections mentioned in
#    $(SECTIONS).
# 4. If $(CLIOENV) has value YES, expand a special section
#    which allows a slave to contact its master.
#
# Note: The variables USERDEF and USERJCL may hold more than one
# dataset name. The names must be separated by blanks.
#
# Additional instructions are in INSTDEFS file on MVS.

#
#    S T A R T   O F   I N S T A L L A T I O N   D E F A U L T S
#
.ECHO "Processing INSTDEF: $(CURFILE) $(DATE) $(TIME)."
.DEBUG CURFILE JOB               # Echo message when opening files

### Variables that are set in each invocation before processing
### user files
# USER     # MVS userid
# PW       # MVS password for $(USER)
# PROG     # Identifies program to be run
# $PARM    # Command line arguments  ( PARM field in JCL )
# APPL     # Set to CLIO by PARSRV
# JC       # Set to next character in $(JOBLETTERS) if it exists
#          # else next character in "BCDEFGHIJKLMNOPQRTUVWXYZ0123456789"
# MSTPORT  # Master's port number         ( from PARSRV )
# MSTHID   # Master's internet address    ( from PARSRV )
# MSTUSER  # Master's user id
# MSTPID   # Master's process id
# PROJECT  = "$($(PROG).PROJECT)"  # If defined in INSTDEF
# NAME     = "$($(USER).NAME)"     # If defined in INSTDEF
```

Figure 39 (Part 1 of 6). Sample JCL Generation Program in Pseudo-JCL (FCFDEFS)

```
######## CLIO/S data set defaults. Set these to match the installation
######## of your CLIO/S data sets.
  CLIOHLQ   = SYS1                # HLQ of CLIO datasets
  PROJECT   = CLIO                # Default project
  CLIODSN   = $(CLIOHLQ)          # CLIO datasets

######## Job letters that can be used in job names. Set these to values
######## that can be used in job names.
  JOBLETTERS= BCDEFGHIJKLMNOPQRTUVWXYZ0123456789

######## Variables in the job card. Set these to match the values needed
######## for your installation.
# $NAME    ="$(USER)$(JC)"
  $NAME    ="$(USER)$(JC)"
# $ACCT    = EPAC,1000
# $PID     = "$($(USER).NAME)"
# $CLASS   = CLASS=X,
# $MSGCLASS= MSGCLASS=R,
# $MSGLEVEL= MSGLEVEL=(1,1),
# $REGION  = REGION=0K,
  $REGION  = REGION=2M,
# $NOTIFY  =
# $TYPRUN  =
# $TIME    =
  $TIME    = TIME=1000,
# $USER    ="USER=$(USER),"
# $PW      ="PASSWORD=$(PW)"

######## STEPLIB datasets. Set these to match the corresponding data
######## sets at your installation.
# PROGLIB  ="$($(PROG).PROGLIB) $(USER).$(PROJECT).LOAD"
  PROGLIB  = SYS1.SFCFLOAD
  SEDCLINK = SYS1.SEDCLINK             # C370 SEDCLINK
  SEQALINK = SYS1.SEQALINK             # PLI SEQALINK
  SAMPRUN2 = VSPASCAL.VSPV1R2.SAMPRUN2 # VSPASCAL SAMPRUN2
  VSF2LOAD = SYS1.VSF2LOAD             # VSF2LOAD

######## Define files holding defaults and JCL for specific programs.
# INSTDEF   = "dd:DEFAULTS"               # Initial file processed
  INSTDEF   = "'$(CLIODSN).SFCFSAMP(FCFDEFS)'" # For PARREFR, if used
  INSTJCL   = "'$(CLIODSN).JCLGEN($(PROG))'"
  USERFILE  = "$(USER).$(PROJECT).JCLGEN"   # For user use
  USERFIL2  = "$(USER).API.CNTL"       # For compatibility
  USERJCL1  = "'$(USERFILE)($(PROG))'" # For user use
  USERJCL2  = "'$(USERFIL2)($(PROG))'" # For user use
  USERDEF   = "'$(USERFILE)($DEF$)' '$(USERFIL2)($DEF$)'"
  USERJCL   = "$(USERJCL1) $(USERJCL2) $(INSTJCL)"
```

Figure 39 (Part 2 of 6). Sample JCL Generation Program in Pseudo-JCL (FCFDEFS)

```
######## JCL for TAPESRV
TAPESRV.SECTIONS = START JOB JES TAPESRV END
TAPELOAD = "$(CLIODSN).SFCFLOAD"
TAPESRV.1 ="//TAPESRV  EXEC PGM=$(PROG),PARM='NOSTAE,NOSPIE'"
TAPESRV.2 ="//STEPLIB  DD   DISP=SHR,DSN=$(TAPELOAD)"
TAPESRV.3 ="//         DD   DISP=SHR,DSN=$(SEQALINK)"
TAPESRV.4 ="//         DD   DISP=SHR,DSN=$(SEDCLINK)"
TAPESRV.5 ="//         DD   DISP=SHR,DSN=$(SAMPRUN2)"
TAPESRV.6 = //SYSUDUMP DD   SYSOUT=*,DCB=BLKSIZE=3429
TAPESRV.7 = //SYSPRINT DD   SYSOUT=*
TAPESRV.8 = //FT06F001 DD   SYSOUT=*

######## JCL for FCFFSERV
FCFFSERV.SECTIONS = START JES FCFFSERV END

######## JCL for CLFTP
CLFTP.SECTIONS = START JES CLFTP END

| ######## JCL for CLPLINK
| CLPLINK.SECTIONS = START JES CLPLINK END

| FCFFLD = "$(CLIODSN).SFCFLOAD"
| FCFFSERV.1 = "//$($NAME) JOB ($($ACCT)),'$($PID)',$($CLASS)REGION=17M,"
| FCFFSERV.2 = "//   $($MSGCLASS)$($MSGLEVEL)$($NOTIFY)$($TYPRUN)"
| FCFFSERV.3 = "//   $($TIME)$($USER)$($PW)"
| FCFFSERV.4 ="//FCFFSERV PROC"
| FCFFSERV.5 ="//FCFFSERV EXEC PGM=IKJEFT01"
| FCFFSERV.6 ="//STEPLIB  DD   DISP=SHR,DSN=$(FCFFLD)"
| FCFFSERV.7 ="//SYSIN    DD   DUMMY"
| FCFFSERV.8 ="//INSPLOG  DD   SYSOUT=4,DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)"
| FCFFSERV.9 ="//REXXLST  DD   DSN=$(CLIODSN).SFCFSAMP,DISP=SHR"
| FCFFSERV.10 = //SYSUDUMP DD   SYSOUT=4
| FCFFSERV.11 = //SYSPRINT DD   SYSOUT=4
| FCFFSERV.12 = //SYSTSPRT DD   SYSOUT=4,DCB=BLKSIZE=3429
| FCFFSERV.13 = //    PEND
| FCFFSERV.14 = //P1  EXEC FCFFSERV
| FCFFSERV.15 = //SYSTSIN  DD   *
| FCFFSERV.16 ="   CALL '$(FCFFLD)(FCFFSERV)' '$(PARM2)'"


| CLFLD = "$(CLIODSN).SFCFLOAD"
| CLFTP.1 = "//$($NAME) JOB ($($ACCT)),'$($PID)',$($CLASS)REGION=17M,"
| CLFTP.2 = "//   $($MSGCLASS)$($MSGLEVEL)$($NOTIFY)$($TYPRUN)"
| CLFTP.3 = "//   $($TIME)$($USER)$($PW)"
| CLFTP.4 = "//CLFTP EXEC PGM=$(PROG)$(PARM)"
| CLFTP.5 = "//STEPLIB  DD   DISP=SHR,DSN=$(CLFLD)"
| CLFTP.6 = //SYSUDUMP DD   SYSOUT=4
| CLFTP.7 = //SYSPRINT DD   SYSOUT=4
```

| *Figure 39 (Part 3 of 6). Sample JCL Generation Program in Pseudo-JCL (FCFDEFS)*

```
| CLPLD = "$(CLIODSN).SFCFLOAD"
| CLPLINK.1 = "//$($NAME) JOB ($($ACCT)),'$($PID)',$($CLASS)REGION=17M,"
| CLPLINK.2 = "//   $($MSGCLASS)$($MSGLEVEL)$($NOTIFY)$($TYPRUN)"
| CLPLINK.3 = "//   $($TIME)$($USER)$($PW)"
| CLPLINK.4 = "//CLPLINK EXEC PGM=$(PROG)$(PARM)"
| CLPLINK.5 = "//STEPLIB  DD   DISP=SHR,DSN=$(CLPLD)"
| CLPLINK.6 = //SYSUDUMP DD   SYSOUT=4
| CLPLINK.7 = //SYSPRINT DD   SYSOUT=4
  ######## This is the default JCL produced
  # JOB.1     = "//$($NAME) JOB ($($ACCT)),'$($PID)',$($CLASS)$($REGION)"
  # JOB.2     = "//   $($MSGCLASS)$($MSGLEVEL)$($NOTIFY)$($TYPRUN)"
  # JOB.3     = "//   $($TIME)$($USER)$($PW)"
  # EXEC.1    = "//$(PROG)  EXEC PGM=$(PROG)$(COND)$(PARM)"
    STEPLIB.1 = "//STEPLIB  DD   DISP=SHR,DSN=$(?PROGLIB)"
    STEPLIB.2 = "//         DD   DISP=SHR,DSN=$(SEDCLINK)"
    STEPLIB.3 = "//         DD   DISP=SHR,DSN=$(SAMPRUN2)"
    STEPLIB.4 = "//         DD   DISP=SHR,DSN=$(VSF2LOAD)"
  # SYSPRINT.1=  //SYSPRINT DD   SYSOUT=*
  # SYSPRINT.2=  //FT06F001 DD   SYSOUT=*

  ######## Assign names to some userids. Default is "$(USER)".
  # NAME         = "$(USER)"             # Default name
    EPAC201.NAME = O. Skudal
    EPAC202.NAME = T. Rugland
    EPAC203.NAME = O. Gjerde
    EPAC312.NAME = F. Harloff

  ######## Sundry defaults
  # .DELIM    = /*PROCESS*/         # End of inline block
  # INLINE    = SUBST               # Substitute inline variables
  # CLIOENV   = "$($(PROG).CLIOENV) YES"  # Expand CLIOENV DD

  ######## Define the default sections
  # SECTIONS = START JOB JES EXEC STEPLIB SYSPRINT END
  #
  # Add ARGS to SECTIONS if arguments are wanted in a file
    ARGSDD    = "ARGS    "                  # Name of DD card
    ARGS.1    = "//$(ARGSDD) DD   *"
    ARGS.2    = "$(PROG) $($PARM)"
    ARGS.3    = "/*"
```

Figure 39 (Part 4 of 6). Sample JCL Generation Program in Pseudo-JCL (FCFDEFS)

```
######## EXEC PARM field
# PARM     =",PARM='$($PARM)'"
PARM2    ="$($PARM)"
# Use PARM3 instead of PARM or PARM2 when a CLIOENV DD statement is
# not generated for FCFFSERV, CLFTP, and CLPLINK. This will occur
# when CLIOENV = NO.
PARM3    =",PARM='-c $(MstPort),$(MstHid) $(PARM2)'"

######## JCL for WTOSRV
WTOSRV.SECTIONS = $(TAPESRV.SECTIONS)

######## Sections for the CLIOCAN program
# C*A*N.SECTIONS = C*A*N;C*A*N.CLIOENV = NO    # Supress CLIOENV DD

# This should work if INTRDR has system authority (JES2:$T RDI,A=3 )
# JTYPE = "$($PARM)/100000;JOBNO = "$($PARM)-$(JTYPE)*100000"
# LETTER.2=J;LETTER.3=S;LETTER.4=T   # JOB/STC/TSO
# C*A*N.1 = "/*$C $(LETTER.$(JTYPE))$(JOBNO)"

######## JCL for running CLIOCAN
  C*A*N.1 = "//$($CANNAME) JOB ($($CANACCT)),'$($CANPID)',$($CANCLASS)"
  C*A*N.2 = "//   $($CANMSGCLASS)$($CANMSGLEVEL)$($CANNOTIFY)"
  C*A*N.3 = "//   $($TIME)$($USER)$($PW)"
######## Using the authorized program
# C*A*N.4 = "//FCFCAN   EXEC PGM=$(CANPROG),PARM='$($PARM)'"
# C*A*N.5 = "//STEPLIB  DD   DISP=SHR,DSN=$(CANPROGLIB)"
# C*A*N.6 = "//SYSPRINT DD   SYSOUT=*"

######## Using the REXX version of CLIOCAN
  C*A*N.4 = "//FCFCANC  EXEC PGM=IKJEFT01"
  C*A*N.5 = "//SYSPROC  DD   DISP=SHR,DSN=$(CLIODSN).REXX"
  C*A*N.6 = "//SYSTSPRT DD   SYSOUT=*"
  C*A*N.7 = "//SYSTSIN  DD   *"
  C*A*N.8 = "%FCFCANC $($PARM) $(USER)"  #Job ID and USERID
  C*A*N.9 = "/*"

######## Variables in the job card for CLIOCAN
# $CANNAME     = FCFCAN
# $CANACCT     = CLIO,1000
# $CANPID      = FCFCAN
  $CANCLASS    = CLASS=A,
  $CANMSGCLASS = MSGCLASS=4,
  $CANMSGLEVEL = MSGLEVEL=(0,0),
  $CANREGION   = REGION=2M,

  $CANPROG     = FCFCAN                  # Program name
  $CANPROGLIB  = SYS1.APFLINK            # Needs APF Authorization
```

Figure 39 (Part 5 of 6). Sample JCL Generation Program in Pseudo-JCL (FCFDEFS)

```
######## Sections which can be used for started tasks ( $T RDI,A=3 )
# STC.1    = "/*$VS,'S $(PROG).$(USER),PRM=($(MSTPORT),$(MSTHID))'"
#
# The following should be used for FCFFSERV, CLFTP, and CLPLINK.
# STC.1    = "/*$VS,'S $(PROG).$(USER),PRM=($(PARM3))'"
#
#### Try to run TAPESRV as a started task
# TAPESRV.SECTIONS = STC; TAPESRV.CLIOENV = NO
# WTOSRV.SECTIONS  = STC; WTOSRV.CLIOENV = NO
#
# Note: This presumes that you have JCL for TAPESRV in one of your
#    PROCLIB datasets.  (Try 'S TAPESRV' from the console or sdsf if
#    you want to make sure TAPESRV is there.)
#
#    If you want the task name to be FCFTAPES, try the following:
#    TAPESRV.SECTIONS = TAPESTC; TAPESRV.CLIOENV = NO
#    TAPESTC.1= "/*$VS,'S FCFTAPES.$(USER),PRM=($(MSTPORT),$(MSTHID))'"
#
# Ensure that you have no TAPESRV member in JCLGEN which overrides this
# definition.
#
# Any C program that runs as a started task should have the following
# code at the beginning of it.
# #if defined(MVS)
#    Clioargc = argc;   /* argc is argument count passed to main() */
#    Clioargv = argv;   /* argv is argument ptrs passed to main() */
# #endif
#
#### Try to run FCFIDUM as a started task. The FCFIDUM proc will be
#    started rather than submitting JCL. The FCFIDUM proc must be
#    available in one of your PROCLIB datasets. (Try 'S FCFIDUM' from
#    the console or sdsf if you want to make sure FCFIDUM is there.)
# FCFIDUM.SECTIONS = STC; FCFIDUM.CLIOENV = NO
#
# The following JCL should not be uncommented. It is an example PROC
# that can be used to start FCFIDUM. Any PROC used to start a started
# task must be included in the system PROCLIB concatenation.
#
#   Example PROC to start FCFIDUM:
#   //FCFIDUM  PROC PRM=
#   //*--------------------------------------------------------------
#   //* FUNCTION:
#   //*   STARTED PROGRAM FCFIDUM.
#   //*
#   //* INSTRUCTIONS:
#   //*   1. IF NECESSARY, CHANGE SYS1.SFCFLOAD TO BE THE NAME OF THE
#   //*      FILE CONTAINING THE FCFIDUM LOAD MODULE.
#   //*   2. CHANGE USER.CLIOENV TO BE THE NAME OF THE FILE CONTAINING
#   //*      CLIOENV VARIABLES.
#   //*--------------------------------------------------------------
#   //*
#   //FCFIDUM  EXEC PGM=FCFIDUM,PARM=&PRM
#   //STEPLIB  DD DSN=SYS1.SFCFLOAD,DISP=SHR
#   //CLIOENV  DD DSN=USER.CLIOENV,DISP=SHR
#   //SYSPRINT DD SYSOUT=4
#   //   PEND

######## Variables that are redefined before generating default JCL
# SECTIONS = "$($(PROG).SECTIONS)" # If defined
```

Figure 39 (Part 6 of 6). Sample JCL Generation Program in Pseudo-JCL (FCFDEFS)

Appendix C. 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=.

Table 11 (Page 1 of 4). 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*	
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*	

© Copyright IBM Corp. 1993, 1995

Table 11 (Page 2 of 4). MVS Device Allocation Parameters and CLIO/S Equivalents

DD Statement Parameter	CLIO/S Parameter	Parameter Values
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	
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	

Table 11 (Page 3 of 4). MVS Device Allocation Parameters and CLIO/S Equivalents

DD Statement Parameter	CLIO/S Parameter	Parameter Values
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*	not available	
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*)	Expdt=*yyddd*	
LABEL=(RETPD=*num*)	RETpd=*num*	
LABEL=(,AL)	AL	
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	

Table 11 (Page 4 of 4). MVS Device Allocation Parameters and CLIO/S Equivalents

DD Statement Parameter	CLIO/S Parameter	Parameter Values
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*	
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 197 for more information.)

Appendix D. CLIO/S Performance and Throughput

This section contains a discussion of factors affecting performance and throughput, and shows performance data for CLIO/S.

Performance and throughput are complex problems. There are no hard and fast rules. This section points out factors that are important in CLIO/S performance. You must determine how each factor applies to your application and its environment, using both analysis and experimentation.

IBM does not guarantee a particular level of performance in your environment.

Performance Considerations

Many factors affect CLIO/S performance and throughput. Factors that can be controlled by the application writer include:

- Size of the transmit/receive buffer
- Alignment of the transmit/receive buffer
- Concurrent processes

Factors that are controlled by systems administrators and/or hardware planners include:

- Configuration of the adapter
- TCP/IP performance
- Support levels
- Number of adapters
- System workload and processor speed

Each of these factors is discussed in the following sections.

Note: Most of the performance tuning effort will take place on the workstation rather than the mainframe.

Size of the Transmit/Receive Buffer

CLIO/S sends data in 60K chunks. Your application gets best performance when it transmits data in blocks that are a multiple of this size. In general, the larger the transmission size, the better.

Alignment of the Transmit/Receive Buffer

If your data is being transmitted from or received into storage that is not aligned on a 4K page boundary, system throughput can be degraded. CLIO/S uses 4K-aligned storage for data transfer; if your data is not aligned, the adapter device driver must copy it to 4K-aligned storage.

Concurrent Processes

CLIO/S I/O operations proceed in a step-by-step manner. Inevitably, some system resources will be idle while CLIO/S is using others. To get the most out of all resources, you can create several concurrent CLIO/S operations.

© Copyright IBM Corp. 1993, 1995

There are several ways this can be accomplished.

- Multiple processes using CLIO/S. This might occur naturally in some environments.
- Single process forking and multiple child processes using CLIO/S.
- Using the CLIO/S stripes function. See “Using the CLIO/S Striping Function in Your Application” on page 35 for details.

Configuration of the Adapter

The key configuration variable is the number of buffers allocated to transmit (from workstation to the mainframe) and receive (by workstation from mainframe).

The PCA adapter uses 52 4K buffers, 26 for transmit and 26 for receive. You can improve transmit performance without impacting receive performance by increasing the number of buffers allocated to transmission. Try 36 transmit buffers and 16 receive buffers as a possible starting point. You can make this change using either the SMIT menu interface or the AIX **chdev** command.

This example shows how to use the **chdev** command to reallocate the buffers:

```
su root
chdev -l cat0 -a xmitno=36 -a recvno=16 &
```

The ESCON adapter dynamically allocates its buffers. You should not attempt to tune it.

TCP/IP Performance

All CLIO/S interfaces use TCP/IP for initialization. However, the CLIO/S sockets interface uses TCP/IP as part of each **send** or **recv** call. Poor TCP/IP performance has a noticeable impact on the performance of the CLIO/S sockets interface.

Multi-threading can minimize the impact of poor TCP/IP performance.

Support Levels

You **must** have the correct level of device driver and adapter microcode to use CLIO/S. Having back-level microcode or device driver code can cause poor performance or outright failure, especially as more data is driven through the adapter.

Contact your IBM representative for the most current information about system software and microcode requirements.

Number of Adapters

CLIO/S can use more than one channel adapter. Adding an adapter can increase performance. Consider whether your workload justifies the additional hardware.

System Workload and Processor Speed

While efficient, CLIO/S still draws on system resources, and competition for resources among the various processes in the system has an effect on CLIO/S performance and throughput. On the RISC System/6000, high transmission rates will result in high CPU use, with the exact ratio depending on the particular machine in use. If multiple adapters are used, it is possible for CPU use to be a gating factor.

AIX commands such as `vmstat 5` can be used to observe CPU utilization.

Mainframe use will typically be only a fraction of that seen on the workstation, so mainframe processor speed is not a key consideration.

Performance Comparisons

Remember that many factors affect performance. IBM does not guarantee that you will obtain any particular transmission rate or CPU overhead. These performance statistics are presented “as is.”

This section provides performance statistics for the CLIO/S DPI interface and discusses the level of performance you can expect from the CLIO/S tape server, file transfer, and piping interfaces.

The Performance of CLIO/S DPI Interface

Table 13 on page 260 shows performance statistics for the CLIO/S DPI interface. These results indicate that the DPI achieves near-channel speed on memory-to-memory data transfer between channel-attached workstations and mainframes. For comparison, the table also shows the performance when the DPI is forced to use TCP/IP instead of a channel adapter.

This data was collected from runs sending a 1200-kilobyte buffer 900 times (for a total transmission of slightly more than 1 gigabyte) in each direction.

CPU use statistics for ESCON and PCA are provided by the C language **clock** function. For the TCP/IP examples, TSO RMFMON was also used to capture statistics regarding CPU utilization in the MVS TCP/IP address space.

On systems with no other workload there was very little variation between runs.

Table 14 on page 260 describes the performance of the CLIO/S DPI for transferring data between workstations using TCP/IP. The data shows that CLIO/S achieves good performance using the TCP/IP protocol when TCP/IP uses a fast connection such as FDDI.

Table 13. CLIO/S Performance for Mainframe-to-Workstation Data Transfer. The CLIO/S Distributed Processing Interface (DPI) provides near-channel speed between channel-attached workstations and mainframes.

Using 4.5 M/sec Parallel Channel on model 980 workstation				
Direction	**Transfer rate (M/sec)**	**CPU Use (MVS)**	**CPU Use (RISC)**	**Comments**
from MVS	4.1	1%	9%	
to MVS	2.2	1%	14%	xmit/rcv buffer ratio 26/26
to MVS	3.6	1%	24%	xmit/rcv buffer ratio 36/16
Using 10 M/sec ESCON Channel on model 980 workstation				
Direction	**Transfer rate (M/sec)**	**CPU Use (MVS)**	**CPU Use (RISC)**	**Comments**
from MVS	9.8	2%	33%	one stripe
to MVS	6.9	1%	18%	one stripe
from MVS	12.7	2%	76%	two stripes
to MVS	12.3	2%	36%	two stripes
Using 10 M/sec ESCON Channel on SP2 wide node				
Direction	**Transfer rate (M/sec)**	**CPU Use (MVS)**	**CPU Use (RISC)**	**Comments**
from MVS	3.0	15%	1%	DPI over TCP/IP using ESCON
to MVS	1.8	27%	15%	DPI over TCP/IP using ESCON
from MVS	9.1	3%	9%	DPI using ESCON directly
to MVS	8.3	3%	9%	DPI using ESCON directly
Using 10 M/sec ESCON Channel on model 550 workstation				
Direction	**Transfer rate (M/sec)**	**CPU Use (MVS)**	**CPU Use (RISC)**	**Comments**
from MVS	0.4	10%	8%	DPI using TCP/IP over token ring
to MVS	0.3	9%	3%	DPI using TCP/IP over token ring
from MVS	1.3	11%	9%	DPI using ESCON directly
to MVS	0.9	25%	33%	DPI using ESCON directly

Table 14. CLIO/S Performance for Workstation-to-Workstation Data Transfer. The CLIO/S Distributed Processing Interface (DPI) gets better performance with the TCP/IP protocol when TCP/IP uses FDDI instead of token ring.

Using TCP/IP Protocol Between Model 580 and Model 980 Workstations				
Direction	**Transfer rate (M/sec)**	**CPU Use (580)**	**CPU Use (980)**	**Comments**
580 ⇒ 980	0.6	8%	4%	via token ring
580 ⇒ 980	3.1	16%	13%	via FDDI
580 ⇐ 980	0.6	5%	4%	via token ring
580 ⇐ 980	3.0	14%	10%	via FDDI

The Performance of CLIO/S Tape Server and CLIO/S File Transfer Program

Performance of the tape and CLIO/S FTP interfaces is influenced by many factors in addition to those affecting the CMI, DPI, and socket interfaces. Performance is often gated by the speed with which data can be read from or written to the device. The CLIO/S tape server performs “handshaking” on each data transfer to ensure data integrity; this also affects performance. In addition, a small amount of overhead is introduced because the CLIO/S tape server is an application written on top of the DPI interface.

A reasonable goal for data transfer by the CLIO/S tape server and CLIO/S FTP is about half of CLIO/S's memory-to-memory data transfer rate, or 70% of the device read/write rate, whichever is less.

Appendix E. Debugging CLIO/S

This section describes debugging CLIO/S problems in an MVS-to-AIX environment. Debugging in the AIX-to-AIX environment is similar; ignore the material covering CTC connections.

This is not an exhaustive list of possible problems. It focuses on finding and correcting common problems.

Don't forget that parts of your CLIO/S application run on both the MVS host and the RISC System/6000 workstation or SP2 node. It's important to debug from *both* sides of the network. This is especially important when you suspect hardware configuration problems or hardware errors. The MVS file **SYS1.LOGREC**, created by the Environmental Record Editing and Printing Program (EREP), can help debug hardware problems.

Debugging the Installation Process

Because most CLIO/S problems are the result of incorrect or incomplete installation, this section concentrates on errors resulting from installation problems.

If you can get the sample programs provided with CLIO/S to run successfully in both directions between AIX and MVS over a CTC connection, all the necessary components of CLIO/S are working properly. Testing the installation between RISC System/6000s could miss problems that exist in the MVS-RISC System/6000 configuration.

If the sample programs work, but your own applications do not, the problem is probably with your applications.

We will use FCFIDUM as a representative member of the samples. You should be able to successfully run FCFIDUM from both MVS and AIX.

IBM recommends that you turn on the debug messages while you install CLIO/S. See “Setting CLIO/S Environment Variables” on page 29 for more information on setting the CLIODEB debug flags.

Installation Problems

Symptom The FCFIDUM executable doesn't exist or won't run.

Try This Check the installation procedure. Repeat FCFSAMP (MVS) or **make** (AIX) as required, checking for errors. On AIX, make sure your $PATH environment variable lists the directory containing the **fcfidum** executable, or that you invoke the command by its full path name.

Symptom FCFIDUM runs but generates error message `fcfidum: Error -1 from OPNSRV.`

Try This Both **parsrv** (AIX) and **FCFPARSR** (MVS) must be running. See “The CLIO/S Server—PARSRV” on page 28 for more information on starting these programs.

© Copyright IBM Corp. 1993, 1995

Or This On MVS, a likely explanation is that the servant copy of FCFIDUM was started but failed before it could contact the master. Inspect the operator log or held output for traces of the servant job. One particular source of trouble is JCL errors; if these occur, modify the FCFREXX or FCFDEFS file to correct the problem.

Or This For the AIX-to-AIX environment, or during the Installation Verification Program on AIX, make sure that **rexec** is working properly. Try an **rexec** command such as

```
rexec targetnode date
```

If this fails, find and fix the problem with **rexec** before proceeding with CLIO/S debugging.

Symptom FCFIDUM runs but generates error message `Sorry, can't find info about` *targetnode*

Try This The local PARCONF file does not contain an entry for *targetnode* or the entry has an invalid address. Check the debug messages that are generated when CLIO/S reads the PARCONF file; they often contain clues to the exact problem.

The log contains messages from many stages of processing. Focus on messages from the **readconf** routines.

Can you **ping** the internet address specified in the PARCONF file? This demonstrates that you have a valid network address in the PARCONF file.

Symptom FCFIDUM runs and debug messages indicate attempted contact with target, but connection fails with message `OPNSRV select failed. rc=0`

Try This The local NETRC file may have an incorrect password for the target system. You can test this by **ftp**ing to the target host. If you are prompted for your password when you **ftp** to the target with the current NETRC file, your password is not in the NETRC file. If you receive error messages when you **ftp**, there is a problem with your NETRC; read the messages for a clue.

Or This PARSRV on the remote system must be able to locate the FCFIDUM executable or program. On AIX, make sure that the executable is in a directory listed in one of $CLIOBIN, $CLIOPATH, or $CLIOSYS. On MVS, make sure the JCL has been modified to match your installation's values. Also, make sure that FCFREXX or FCFDEFS names the data set containing the MVS executable of FCFIDUM. The CLIO/S Program Directory describes how to modify the JCL for FCFIDUM. For more information on generating JCL with FCFREXX or FCFDEFS, see Appendix B, “Generating JCL for Jobs Started by PARSRV” on page 221.

Symptom FCFIDUM runs successfully but debug messages say `Protocol=TCPIP`

Try This CLIO/S didn't read the PARCTC file. Check the debug messages from the AIX side for one of these problems:

- CLIO/S can't find the PARCTC file in your **$HOME** directory or in **/etc/parctc**.

  Put PARCTC in one of these directories.

- CLIO/S is not looking for the PARCTC file because the PARCONF entry specifies the TCP/IP protocol.

  Fix the PARCONF entry.

Or This CLIO/S read the PARCTC file, but didn't find a valid entry for the CTC connection desired. Messages indicating this problem include:

```
CTC_OPN_0: readconf- No /dev/cat entries found in parctc
```

or

```
CTC_OPN_0: readconf- No /dev/esca entries found in parctc
```

Possible causes for this include incorrect syntax in the PARCTC entry, or an incorrect local or remote node name. The local name should be the host name of the local machine.

Fix the PARCTC entry and try again.

Or This Correct the ESCON adapter subchannel address

When configuring the adapter card, make sure that the remote address matches the address in the UNITNAME statement of the IOCDS on the mainframe. IBM recommends that the local and remote addresses be identical. A mismatch between your PARCTC file and the adapter's configuration can cause CTC initialization to fail. A message similar to the following is issued:

```
ctc_opn - Stripe 1: WS failed to start subchannel 0xCDF
```

Using SMIT, correct the adapter subchannel definition, using the configuration information in “Configuring ESCON Subchannels” on page 21.

Symptom CLIO/S uses the TCP/IP protocol when the PARCONF file specifies SOCC (P_SOCC or P_DYNAMIC).

Try This Make sure your user ID has read and write permission to the SOCC device driver, **/dev/ops0**. “Allowing Access to SOCC Devices” on page 23 describes how to set the correct permissions.

Symptom CLIO/S reads the PARCTC file and attempts to open a subchannel, but fails.

Try This Check the debug messages from both AIX and MVS for one of these conditions:

- Can't find the adapter on the AIX side.

  Is the adapter defined and available?

- No available subchannels on the AIX side.

  Is some other application using the adapter? The CLIO/S utility **online** is one possible culprit.

- Can't allocate subchannels on MVS side.
  - Are correct subchannels defined in PARCTC?
  - Are the AIX adapter definitions correct? Use SMIT to verify.
  - Are IOCDS definitions correct?
  - Are subchannels online and available?

Symptom FCFIDUM runs successfully over the channel, but performance is poor.

Try This See Appendix D, “CLIO/S Performance and Throughput” on page 257 for discussion of factors affecting CLIO/S performance.

Debugging Intermittent, Hardware or Network Problems

This section assumes that the installation verification program has succeeded and that CLIO/S is able to run at least some of the time over CTC connections between MVS and AIX.

Symptom FCFIDUM sometimes runs successfully over the channel, but intermittent errors occur. Intermittent errors can be detected by:

- One or both ends of the transmission ‘hanging’ while waiting for I/O from the other.
- CLIO/S debug messages indicating a mismatch between the actual number of bytes transmitted and what was expected.
- Explicit CLIO/S debug messages indicating channel or adapter problems.

  For example, the message “resource temporarily unavailable” indicates that you need additional mbufs for an ESCON adapter.

These errors are more likely to occur with large data transfers or rapid switching between reading and writing.

Try This Make sure the logic in your application is correct. When using the DPI or CMI, the amount of data read by one side must be the same as the amount sent by the other side. Otherwise, hangs can occur.

Or This Make sure there are sufficient mbufs and clusters. 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. Among other data, the results 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, for example:

```
no -o lowclust=40
```

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 additional mbufs.

Or This Disable the missing interrupt handler.

If the missing interrupt handler is not disabled, channel retries occur. Debug messages containing the following are produced on MVS:

```
OCTC   _0: octc ADDR=0023D000 - CCW=C3000000 20000001 - IDAW=0000 0000
OCTC   _0: octc - WT rc=4
OCTC   _0: octc - CSW=0023D008 0E000001
OCTC   _0: octc - SENSE=00800000
OCTC   _0: octc - WT retrying
```

Disable the missing interrupt handler by adding an entry as described in "Adding a Missing Interrupt Handler Entry" on page 18.

Or This Verify that the correct microcode levels are installed by contacting your IBM representative.

Or This Verify that you have the correct level of AIX installed.

To check the AIX level on your machine, use the **oslevel** command. The result should be `3240`, indicating level 3.2.4, or `3250`, indicating level 3.2.5.

Install the correct AIX level on your machine.

General AIX debugging tips

The AIX operating system has commands that may be helpful in debugging intermittent, hardware, or network errors. When you call IBM service, you may be asked to provide this information. See AIX reference pages for more information about these commands. Note that interpreting the output from these commands may take considerable expertise.

Viewing Error Reports

CLIO/S writes error and informational log records to the AIX system error log using the AIX error logging facility. Hardware errors are also written to this log.

To view the system error log, use the **errpt** command:

```
errpt -a | more
```

Errors recorded by CLIO/S have an ERROR LABEL starting with the string "CLIO," for example, CLIO_INFO102_TR.

You can capture the output in a file using:

```
errpt -a > outfile
```

Tracing an Adapter

The trace data contains log entries from the ESCON and BMCA/PCA adapter device drivers. If you suspect a hardware problem or a network problem with your adapter, the trace entries may contain clues. You can also examine this data to learn if the adapter is being used; if your application is not set up correctly or there is contention for the adapter, CLIO/S may use TCP/IP instead of the adapter.

To collect problem determination information about a channel adapter, use the **trace** command, located in **/usr/lib/ras/ras_smutil**.

To trace the use of ESCON adapters by an application, use the commands:

```
trace -a -j 452 -o tracedata
(run the application)
trcstop
trcrpt tracedata > tracereport
```

The value 452 specifies the ESCON adapter event. “tracedata” and “tracereport” are arbitrary file names.

To trace BMCA/PCA adapters:

```
trace -a -j 350 -o tracedata
(run the application)
trcstop
trcrpt tracedata > tracereport
```

The value 350 specifies the BMCA/PCA adapter event.

Displaying Network Statistics

AIX has many ways to generate network statistics. Examining this data can indicate whether the network is using an adapter. To display information about active sockets for the ESCON adapter **es0**, use this command:

```
netstat -n -I es0
```

Similarly, display information about the BMCA/PCA adapter /**dev**/**cat0** with:

```
netstat -n -I ca0
```

Display routing information about your network configuration using this command:

```
netstat -rn
```

This command lets you see that data (packets) are being transmitted on your network:

```
netstat -i
```

Examine the attributes of an adapter with SMIT or the **lsattr** command. For example, this command shows the state of the BMCA/PCA adapter /**dev**/**cat0**:

```
lsattr -E -l cat0
```

To show the state of the ESCON adapter /**dev**/**esca0**, issue:

```
lsattr -E -l esca0
```

Index

Special Characters

? subcommand of CLFTP 65
.netrc
  See netrc file

A

abend 768 97
accept() socket routine 123
accepting
  a CLIO/S connection from another process 129
  a TCP/IP connection 140
access methods
  used by CLFTP 61
  used by clplink 113
access, allowing to SOCC device 23
adapter
  sharing 37
  tracing an 267
adding a process ID to an alert group 151
AIX environment, debugging the 267
AIX level 267
AL tape preformatting 201
ALERT DPI routine 150
ALERTGRP DPI routine 151
ALLOC
  parameter for LOCSITE and SITE 84
allocating a CLIO/S pipe 104
allocating a CLIO/S tape device 54
allocation device 251
allowing a process to change ownership of a connection 138
ALRTGRP DPI routine 151
anonymous FTP server 9
ANSI Standard Label tape preformatting 201
APPEND subcommand of CLFTP 88
appending files
  using CLFTP 88
  using CLPLINK 106
ASCII subcommand of CLFTP 80
audience of this book xiii

B

batch submission of CLFTP 93
BINARY subcommand of CLFTP 80
BLKWRITE
  parameter for LOCSITE and SITE 86
BLocks
  parameter for LOCSITE and SITE 84
BLOCKSIZE
  parameter for LOCSITE and SITE 84, 86
BMCA
  DPI support of 146
BMCA/PCA
  See BMCA
BMCA/PCA subchannels
  starting 218
buffer sizes
  receive 159
  send 159
buffering mechanism, changing 176

C

CANCEL function (MVS) 177
cancelling a CLIO/S job
  about 206
  with FCFCAN authorized program 206
  with FCFCANC REXX EXEC 207
CD subcommand of CLFTP 70
cdpi_async DPI routine 151
cdpi_check DPI routine 152
cdpi_s2c DPI routine 152
CDPIASY DPI routine 151
CDPICHK DPI routine 152
chained channel programs 168, 178
changing
  send and receive buffer sizes 159
  the CLIO/S tape pseudo device driver 43
changing defaults for environment variables 34
checking
  number of bytes transmitted by a group 152
  the number of bytes transmitted 130
CHPID IOCDS statement 17
CLAW 37
CLAW applications, sharing adapter with 37
clearing buffers written using SPANY 153
CLFLUSH DPI routine 153
CLFTP
  controlling termination of error 80
  customizing 63
  debugging 97
  deleting files on remote host 89
  displaying file transfer status messages 66
  echoing subcommands 66
  file transfer status messages, displaying 66
  internal error codes 97
  list of subcommands 64, 65
  list of subcommands by task 63
  messages 97
  overview 59
  return codes 96
  specifying user ID and password 61

© Copyright IBM Corp. 1993, 1995

CLFTP *(continued)*
starting as a batch job 93
terminating on error 80
using 59
using with multiple PARSRVs 59
CLFTP subcommands
? 65
APPEND 88
ASCII 80
BINARY 80
CD 70
DELETE 89
ECHO 66
ENDONERROR 80
GET 90
HASH 66
HELP 65
LCD 75
LLS 76
LMKDIR 77
LOCSTAT 67
LPWD 79
LS 72
MKDIR 73
PUT 92
PWD 74
QUIT 93
SITE 82
STATUS 68
TRANSLATE 87
clio_accept CMI routine 129
clio_async CMI routine 129
clio_check CMI routine 130
clio_close CMI routine 130
clio_connect CMI routine 131
clio_create CMI routine 131
clio_end CMI routine 132
clio_env CMI routine 132
clio_exp CMI routine 133
clio_h2i CMI routine 134
clio_init CMI routine 134
clio_kill CMI routine 134
clio_l2n
CMI routine 135
DPI routine 154
clio_m2i CMI routine 136
clio_n2h CMI routine 136
clio_n2i CMI routine 137
clio_n2l
CMI routine 135
DPI routine 155
clio_n2m CMI routine 137
clio_own CMI routine 138
clio_pid CMI routine 138
clio_read CMI routine 139
clio_t CMI structure 155
clio_write CMI routine 140
CLIO/S
configuration files 24
configuring 15
debugging 263
man pages 8
networking 15
overview 1
sockets programming interface 121
CLIO/S Cross System Pipe Link
See also CLPLINK
about 99
accessing from files from an application 102
clplink 106
clpmk 104
connecting pipes between remote hosts 103
connecting to a CLIO/S pipe 105
copying files to and from remote hosts 102
creating a named pipe 103
CLIO/S FTP
See CLFTP
CLIOADEB environment variable 30
CLIOBIN environment variable 30
CLIOBLK DPI routine 156
CLIOBLK environment variable 30
CLIOBUF DPI routine 156
CLIOBUF environment variable 30
CLIODEB DPI routine 157
CLIODEB environment variable 30
CLIOENV DPI routine 157
CLIOMST DPI routine 158
CLIONODE environment variable 32
CLIOPATH environment variable 32
CLIOPSID environment variable 32
CLIOPSPT 59
CLIOPSPT environment variable 32
CLIOSOCK DPI routine 158
cliostae 177
CLIOSTAE environment variable 178
CLIOSTRIPES environment variable 32
CLIOSYS environment variable 32
CLIOTBS DPI routine 159
CLIOTBS environment variable 33
CLIOTMO DPI routine 159
CLIOTMO environment variable 33
cliotrce 177
close() socket routine 123
closing
a CLIO/S connection 130
the DPI environment 160
the tape 186
CLPLINK
customizing 117
JCL for starting 209, 210
using 106

clpmk 104
CLSFAM DPI routine 160
CLSPAR DPI routine 160
CLSSRV DPI routine 161
cltalloc command 54
cltclntd 47
cltclntd command 47
cltterm command 48
cltunall command 57
CMI interface
  about 127
  clio_accept 129
  clio_async 129
  clio_check 130
  clio_close 130
  clio_connect 131
  clio_create 131
  clio_end 132
  clio_env 132
  clio_exp 133
  clio_h2i 134
  clio_init 134
  clio_kill 134
  clio_m2i 136
  clio_n2h 136
  clio_n2i 137
  clio_n2l 135
  clio_n2m 137
  clio_read 139
  clio_write 140
  tcp_accept 140
  tcp_connect 141
codes
  return, CLFTP 96
commands
  clplink 106
  clpmk 104
  cltalloc command 54
  cltclntd 47
  cltterm 43
  cltterm command 48
  cltunall 57
Communications Manager Interface 127
compiling the tape interface on MVS 199
compiling the tape interface on RISC System/6000 199
configuration files, creating and customizing 23
configuring CLIO/S 15, 23
configuring CLIO/S for striping 36
configuring CTCs on AIX 19
configuring CTCs on MVS 17
configuring the tape mount daemon
  procedures for 40
  with intermediate systems 41
confirming data was sent
  from master to servant 163
confirming data was sent *(continued)*
  from servant to master 172
connect() socket routine 123
connecting to a CLIO/S named pipe 106
converting data
  representations 135
  to local format 155
  to network format 154
converting data with CLFTP 87
copying files
  from a remote host using CLFTP 90
  from anonymous FTP server 9
  to a remote host using CLFTP 92
counting valid entries in PARCONF 163
creating
  local directory with CLFTP 77
  remote directory with CLFTP 73
CTLUNIT IOCDS statement 17
ctpalloc tape interface routine 183
ctpclose tape interface routine 186
ctpexit tape exit 197
ctpopen tape interface routine 187
ctpread tape interface routine 188
ctprew tape interface routine 189
ctpskpf tape interface routine 189
ctpskpr tape interface routine 190
ctpunall tape interface routine 191
ctpvols tape interface routine 191
ctpweof tape interface routine 192
ctpwrite tape interface routine 193
ctpwrto tape interface routine 194
customizing
  CLFTP 63
  CLPLINK 117
  JCL generation by PARSRV 221
CYLINDERS
  parameter for LOCSITE and SITE 84

D

data conversion
  CMI 127
  DPI 143, 156, 161, 162, 171
data translation for CLFTP 87
data translation for clplink 106
data types
  CMI 127
  DPI 143
DCBDSN
  parameter for LOCSITE and SITE 84
debug flags
  setting 210
debugging
  CLFTP 97
  CLIO/S 263
  hardware problems 266

debugging *(continued)*
installation problems 263
intermittent problems 266
network problems 266
the AIX environment 267
decr_pw 35
defaults
changing for PARSRV 34
establishing for PARSRV 34
DELETE subcommand of CLFTP 89
deleting files
from a remote host using CLFTP 89
device allocation 251
directory
APPEND parameter 88
CD parameter 70
changing on remote MVS host 71
changing the directory on MVS 76
changing the directory on remote host 70
establishing for CLFTP 62
GET parameter 90
LCD parameter 75
local MVS host 76
LPWD parameter 79
MKDIR parameter 73
MVS CLFTP server 76
parameter for LOCSITE and SITE 84
PUT parameter 92
remote working directory 74
the local working directory 75
displaying
current working-level qualifier 79
file transfer status messages in CLFTP 66
local host status information for CLFTP 67
remote status information for CLFTP 68
the current remote working directory for CLFTP 74
displaying server information 210
Distributed Processing Interface
See DPI
documentation 7
downloading files from anonymous FTP server 9
DPI
See also DPI C interface
See also DPI FORTRAN interface
about 146
sample master and servant code 148
using 143
DPI C interface
ALERT 150
ALERTGRP 151
cdpi_async 151
cdpi_check 152
cdpi_s2c 152
CLFLUSH 153
CLIOBLK 156
CLIOENV 157
DPI C interface *(continued)*
CLIOSOCK 158
CLIOTBS 159
CLSFAM 160
CLSPAR 160
CLSSRV 161
FNDMACH 161
MS_flush 163
MS_xxxx (macros) 162
MSANY 162
MSxxxx 162
NMACHS 163
OPNFAM 164
OPNPAR 164
OPNSRV 165
QMACH 166
QSLAVE 167
RP_xxxx (macros) 167
RPANY 167
rw_dpi 168
SCHEDUL 169
SETFAM 170
SETSRV 170
SM_flush 172
SM_xxxx 171
SMANY 171
SP_xxxx 173
SPANY 172
SYNCHRO 174
TSKEXIT 174
WAITFOR 175
WAITREQ 176
DPI FORTRAN interface
ALERT 150
ALRTGRP 151
CDPIASY 151
CDPICHK 152
CLFLUSH 153
CLIOBLK 156
CLIOBUF 156
CLIODEB 157
CLIOENV 157
CLIOMST 158
CLIOTMO 159
CLSFAM 160
CLSPAR 160
CLSSRV 161
FNDMACH 161
LNxxxx 154
MSFLSH 163
MSxxxx 162
NLxxxx 155
NMACHS 163
OPNFAM 164
OPNPAR 164
OPNSRV 165

DPI FORTRAN interface *(continued)*
RPxxxx 167
SCHEDUL 169
SETFAM 170
SETSRV 170
SM_xxxx (macros) 171
SMFLSH 172
SMxxxx 171
SP_xxxx (macros) 172
SPxxxx 172
SYNCHRO 174
WAITFOR 175
WAITREQ 176
DPI low level interface
rw_dpi 179
dynamic allocation
for CLPLINK and CLFTP 84
for tape server 183
list of parameters 251

E

ECHO subcommand of CLFTP 66
encr_pw 35
ending a CLIO/S connection 132
ENDONERROR subcommand of CLFTP 80
environment variables
changing default values 34
CLIOADEB 30
CLIOBIN 30
CLIOBLK 30
CLIOBUF 30
CLIODEB 30
CLIONODE 32
CLIOPATH 32
CLIOPSID 32
CLIOPSPT 32
CLIOSTRIPES 32
CLIOSYS 32
CLIOTBS 33
CLIOTMO 33
list of reserved names 33
MVS characteristics for *user_ID*.CLIOENV 30
setting 29
error codes
internal, CLFTP 97
error reports, generating 267
errpt command
using to debug CLIO/S problems 267
ESCON
DPI support of 146
esoteric names 17
establishing a TCP/IP connection 141
ESTAE support 177

F

FCFCAN 206
FCFCANC 206
FCFCANC REXX EXEC 207
fcfcmi.h header file 127, 143
fcfconf.c 34
FCFDEFS
sample program 243
fcfdpi.h header file 143
FCFFCLPR
about 209
JCL for starting 209
FCFFCLPW
about 209
JCL for starting 210
FCFFPIPR
about 215
JCL for starting 216
FCFFPIPW
about 215
JCL for starting 216
FCFFSERV 59
FCFIDUM 207
FCFPARDE 210
FCFPARIN 210
FCFPARPI 211
FCFPARRE 212
FCFPARSR
See PARSRV
FCFPARST 214
FCFREXX
See also JCL generation, REXX
sample program 226
FCFTPX00 tape exit 196
FCFTTEST sample program 199
FCFVARY
about 216
modifying 216
using 218
file transfer types
and CLFTP 80
and CLPLINK 106, 107
find_pw 35
finding related publications xv
FNDMACH DPI routine 161
forcing data transfer to the servant ID 170
fs_accpt() socket routine 123
fs_close() socket routine 123
fs_conn() socket routine 123
fs_read() socket routine 123
fs_selct() socket routine 123
fs_shutd() socket routine 123
fs_write() socket routine 123

G

GDG data sets
  specifying tape allocation parameters 184
generating error reports 267
generating JCL
  See JCL generation
GET subcommand of CLFTP 90
getsockopt() 123
getting
  a list of tape volumes 191
  an Internet address 136
  HELP for CLFTP 65
  machine information 166
  servant machine information 167
  the environment 132
  the environment file 157
  the machine information ID 137

H

hardware, list of supported see CLIO/S *General Information Manual*
HASH subcommand of CLFTP 66
header file
  fcfcmi.h 127
help
  online (AIX man pages) 8
HELP subcommand of CLFTP 65
host
  disconnect (CLFTP) 93
  local status information for CLFTP 67
  remote 70
  remote status information 68
how to use this book xiii

I

independent masters, starting 158
indicating a servant is waiting for work 176
informing a master the servant process is finished 174
initializing
  CLIO/S 134
  the DPI environment 164
installation problems 263
INSTDEFS file
  See FCFDEFS
intermediate nodes
  CLPLINK example 115, 116
  configuring tape mount daemon for 41
  PARCONF entries for 24
  using CLFTP 60, 94
  using CLPLINK 107
  using tape API 184
IOCP generation
  esoteric names required for 17
  IOCDS statements 17
IODEVICE IOCDS statement 17

J

JCL generation
  for started tasks 222
  migrating to CLIO/S Version 2 Release 1 222
  pseudo-JCL
    assignment statements 237
    overview 235
    refreshing PARSRV server 34
    sample FCFDEFS program 243
    sections 237
    selecting 235
    special variables 238
    variables 237
  REXX
    debugging with TRACE and SAY 225
    generating JCL 223
    implementation of 225
    input to 223
    sample FCFREXX program 225
    SAY 225
    services available 223
    setting return code 225
    TRACE 225
  specifying language for PARSVR 33

L

large-volume transmission 121
LCD subcommand of CLFTP 75
limitations of sockets 124
linking the tape interface on MVS 199
linking the tape interface on RISC System/6000 199
listing files in local directory with CLFTP
listing files in remote directory with CLFTP
LLS subcommand of CLFTP 76
LMKDIR subcommand of CLFTP 77
LNxxxx DPI routine 154
local host
  creating directory using CLFTP 77
  listing files in directory using CLFTP 76
  specifying with CLIONODE 32
  status information 67
locating
  a machine by symbolic name 137, 161
  machine information using a machine number 136
  machine information using an Internet address 134
LOCSITE
  list of AIX parameters for CLFTP 86
  list of MVS parameters for CLFTP 83
  subcommand of CLFTP 82
LOCSTAT subcommand of CLFTP 67
logging off CLFTP 93

LPWD subcommand of CLFTP 79
LRECL
  parameter for LOCSITE and SITE 85
LS subcommand of CLFTP 72
lsattr command
  showing the state of an adapter 268

M

man pages 8
masking printed status messages 157
master/servant model 145
MAXBLOCKSIZE
  parameter for LOCSITE and SITE 86
messages
  CLFTP 97
migrating
  FCFDEFS 222
  JCL generation programs 222
missing interrupt handler entry 18
MKDIR subcommand of CLFTP 73
monitoring
  tape mount daemon activity 42
MS_flush DPI routine 163
MS_xxxx DPI macros 162
MSANY DPI routine 162
MSFLSH DPI routine 163
MSxxxx DPI routine 162
multivolume tape support 198
MVS tape management 197
MVS UNITNAME 17
MVS user abend 768 97

N

netrc
  and CLFTP 61
  and CLPLINK 110
  bypassing for all intermediate CLFTP hosts 96
  bypassing for some intermediate CLFTP hosts 96
  identifying yourself to CLFTP 61
  specifying the downstream host for CLFTP 95
netrc file
  creating 27
netstat command
  using to debug CLIO/S problems 268
network statistics 268
networking CLIO/S 15
NLxxxx DPI routine 155
NMACHS DPI routine 163
NOBLKWRITE
  parameter for LOCSITE and SITE 86
non-CLIO/S applications, sharing adapters with 37
NORAWWRITE
  parameter for LOCSITE and SITE 86
notation, syntax xvi

O

objectives for this manual xiii
online 20, 218
online help (AIX man pages) 8
opening
  a family of processes 164
  the tape device for use 187
OPNFAM DPI routine 164
OPNPAR DPI routine 164
OPNSRV DPI routine 165
organization of this book xiii
overview of CLIO/S 1

P

parallel channel adapter (PCA)
  See BMCA
PARCONF file 24
PARCTC file 26
pardeb 210
parinfo 210
parping 211
parrefr 212
PARSRV
  about 28
  customizing JCL for jobs started by 221
  generating JCL for jobs started by 221
  JCL for starting 213, 214
  multiple instances 32, 35, 40, 44, 59, 101, 108, 112, 221
    CLPLINK 112
  port number 29
  refreshing 212
  specifying JCL generation language 33
  starting 28, 33, 213
  stopping 214
  testing whether running 211
  used by CLIO/S FTP 59
  used by DPI 146
  where it looks for servant programs 33
parstop 214
password
  supplying for multiple hosts for CLFTP 95
password security 35
PCA
  See BMCA
performance hints, socket 125
piping data between remote hosts
port number
  See PARSRV
prerequisites see CLIO/S *General Information Manual*
PRIMARY
  parameter for LOCSITE and SITE 85

problem solving 263
problems, installation 263
Program Directory for CLIO/S 15
pseudo-JCL
  See JCL generation, pseudo-JCL
PUT subcommand of CLFTP 92
PWD subcommand of CLFTP 74

Q

QMACH DPI routine 166
QSAM
  used by CLFTP 61
  used by clplink 113
QSLAVE DPI routine 167
QUIT subcommand of CLFTP 93

R

RAWWRITE
  parameter for LOCSITE and SITE 86
read() socket routine 123
readers of this book xiii
reading
  data 168
  data over a CLIO/S connection 139
  multiple tape vols 198
  the tape 188
receive buffer sizes 159
receiving data from a process 167
RECFM
  parameter for LOCSITE and SITE 85
reconfiguring
  the CLIO/S tape pseudo device driver 44
recv() socket routine 123
refreshing PARSRV defaults 212
related publications xv
remote host
  creating directory using CLFTP 73
  listing files in directory using CLFTP 72
  sending information 82
  status information 68
removing
  the CLIO/S tape pseudo device driver 46
requesting a CLIO/S connection to another process 131
restarting
  the CLIO/S tape mount daemon 44
  the CLIO/S tape pseudo device driver after a reboot 44
return codes 96
return values, tape 195
returning
  a CLIO/S pointer 152
  a process ID 138
  the socket descriptor 158
rewinding a tape 189
REXX
  See JCL generation, REXX
routed nickname
  See intermediate nodes
routing
  See intermediate nodes
RP_xxxx DPI macros 167
RPANY DPI routine 167
RPxxxx DPI routine 167
rw_dpi 179
rw_dpi DPI routine 168

S

samples
  about sample JCL 205
  CLFTP as a batch job 94
  DPI master and servant programs 148
  FCFCANC 206
  FCFDEFS 243
  FCFFCLPR 209
  FCFFCLPW 210
  FCFFPIPR 215
  FCFFPIPW 215
  FCFIDUM 207
  FCFPARDE 210
  FCFPARIN 210
  FCFPARPI 211
  FCFPARRE 212
  FCFPARSR 213
  FCFPARST 214
  FCFREXX 226
  FCFTTEST 199
  FCFVARY 218
  list of JCL samples 205
  tape interface 181
SCHEDUL DPI routine 169
scheduling a task to start 169
SECONDARY
  parameter for LOCSITE and SITE 85
security, password 35
select() socket routine 123
send buffer sizes 159
send() socket routine 123
sending data to a process 172
SETFAM DPI routine 170
setsockopt() 123
SETSRV DPI routine 170
setting
  an alert group 150
  an asynchronous flag by group 151
  CLFTP transfer type to ASCII 80
  CLFTP transfer type to BINARY 80
  CLIO/S environment variables 29
  debug flags for PARSRV 210

setting *(continued)*
the asynchronous data flag 129
the buffer bypass limit 156
the environment 133
the family ID 170
the size of the internal buffer 156
the time-out period 159
sharing a CTC adapter with other applications 37
shutdown() socket routine 123
SITE
list of AIX parameters for CLFTP 86
list of MVS parameters for CLFTP 83
subcommand for CLFTP 82
skipping
tape files 189
tape records 190
SM_flush DPI routine 172
SM_xxxx DPI macros 171
SMANY DPI routine 171
SMFLSH DPI routine 172
SMxxxx DPI routine 171
SOCC device
allowing access 23
DPI support of 146
socket calls 121
socket inheritance 124
socket performance hints 125
sockets programming with CLIO/S 121
sockets, limitations 124
sockets, portability 123
software, list of supported see CLIO/S *General Information Manual*
SP_xxxx DPI macros 172
SP_xxxx DPI routine 173
SPANY DPI routine 172
specifying a tape drive 54
specifying local host information for CLFTP 82
specifying the tape device you want 53
SPxxxx DPI routine 172
starting
a master process 149
a process on a node 131
a series of reads 179
a series of writes 179
a servant process 149, 165
independent masters 158
the CLIO/S server 28, 213
the CLIO/S tape mount daemon 47
the remote tape server 183
statistics, network 268
status information using CLFTP 66
STATUS subcommand of CLFTP 68
stopping
a process on a node 134
all processes in a family 160
tape mount daemon 48
stopping *(continued)*
the CLIO/S server 214
the CLIO/S tape mount daemon 43
the remote tape server 57, 191
the servant process 161
striping function 35
striping, configuring CLIO/S for 36
subchannels
configuring BMCA/PCA 19
configuring ESCON 21
varying on and off 216
Sun RPC 145
supported hardware and software see CLIO/S *General Information Manual*
SYNCHRO DPI routine 174
syntax notation xvi
SYS1.PARMLIB(IECIOS00) entries 18
system information using CLFTP 66

T

tape device 53
tape device note 181
tape driver 53
tape exit routine 196
tape I/O 202
tape interface
about 181
AIX tape exit routine 197
cltalloc 54
cltclntd command 47
cltterm 48
cltunall command 57
ctpalloc C function 183
ctpclose C function 186
ctpexit 197
ctpopen C function 187
ctpread C function 188
ctprew C function 189
ctpskpf C function 189
ctpskpr C function 190
ctpunall C function 191
ctpvols C function 191
ctpweof C function 192
ctpwrite C function 193
ctpwrto C function 194
FCFTPX00 196
MVS tape exit routine 196
return values 195
tp utility 200
tpalloc FORTRAN function 183
tpclose FORTRAN function 186
tpopen FORTRAN function 187
tpread FORTRAN function 188
tprew FORTRAN function 189
tpskpf FORTRAN function 189

tape interface *(continued)*
tpskpr FORTRAN function 190
tpunall FORTRAN function 191
tpvols FORTRAN function 191
tpweof FORTRAN function 192
tpwrite FORTRAN function 193
tpwrto FORTRAN function 194
tpwrto program 200
tape mount daemon
about 39
monitoring 42
restarting 44
signaled by tape pseudo device driver 47
stopping 43
tape pseudo device driver
changing 43
making available 39
reconfiguring 44
removing 46
restarting after reboot 44
signals tape mount daemon 47
stopping 48
unconfiguring 45
tape return values 195
TAPESRV 39
tcp_accept CMI routine 140
tcp_connect CMI routine 141
TCP/IP
ensuring complete installation 17
selecting as CLIO/S protocol 25
testing the CLIO/S Cross System Pipe Link 215
testing whether PARSRV is running 211
toggles
ECHO—echoing subcommands 66
ENDONERROR—controlling termination on error 80
HASH—controlling file transfer status messages 66
LOCSITE 82
TRANSLATE—controlling data translation site 87
tp tape message utility 200
tpalloc tape interface routine 183
tpclose tape interface routine 186
tpopen tape interface routine 187
tpread tape interface routine 188
tprew tape interface routine 189
tpskpf tape interface routine 189
tpskpr tape interface routine 190
tpunall tape interface routine 191
tpvols tape interface routine 191
tpweof tape interface routine 192
tpwrite tape interface routine 193
tpwrto support 200
tpwrto tape interface routine 194
trace command
using to debug CLIO/S problems 267
tracing an adapter 267
TRACKS
parameter for LOCSITE and SITE 85
transferring data
APPEND subcommand of CLFTP 88
DELETE subcommand of CLFTP 89
from master to servant 162
from servant to master 171
GET subcommand of CLFTP 90
PUT subcommand of CLFTP 92
transferring files
using CLFTP 59
using CLIO/S Cross System Pipe Link 102
TRANSLATE subcommand of CLFTP 87
TSKEXIT DPI routine 174

U

unallocating a CLIO/S tape device 57
unconfiguring
the CLIO/S tape pseudo device driver 45
UNIT
parameter for LOCSITE and SITE 86
UNITNAME IOCDS statement 17
user abend 768 97
user ID
for CLFTP 61
for CLPLINK 110
supplying for multiple hosts for CLFTP 95
using CLFTP
appending files on a remote host 88
copying files from remote host 90
copying files to remote host 92
creating directories 73, 77
deleting files from remote host 89
deleting remote data 89
disconnect from the host 93
displaying local host status information 67
displaying status information 66
establishing an AIX working directory 62
establishing an MVS working directory 62
listing files in a local directory 76
listing files in a remote directory 72
specifying site parameters 82
transferring data 88, 90, 92
transferring files 59
with multiple PARSRVs 59
using the CLIO/S striping function 35
using this book xiii

V

validating
CLIO/S communication 207
varying on and off subchannels 216

verifying
CLIO/S Cross System Pipe Link 209
verifying CLIO/S communication 207
VOLUME
parameter for LOCSITE and SITE 86

W

WAITFOR DPI routine 175
waiting
until all or selected tasks are completed 174
until an alert is available 175
WAITREQ DPI routine 176
working directory
displaying name of 74
on AIX for CLFTP 62
on MVS for CLFTP 62, 75, 76
working-level qualifier 75, 79
write() socket routine 123
writing
data 168
data over a CLIO/S connection 140
message to operator 194
multiple tape vols 198
tape marks 192
to tape 193

Communicating Your Comments to IBM

IBM Client Input Output/Sockets
User's Guide and Reference
Version 2 Release 1.0

Publication No. CLIO-user-01

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

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

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:

  FAX: (International Access Code)+1+914+432-9405
- If you prefer to send comments electronically, use this network ID:
  - IBMLink: KGNRCF at KGNVMC
  - IBM Mail Exchange: USIB27BQ at IBMMAIL
  - Internet: kgnrcf@vnet.ibm.com

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies.

Readers' Comments — We'd Like to Hear from You

IBM Client Input Output/Sockets
User's Guide and Reference
Version 2 Release 1.0

Publication No. CLIO-user-01

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Today's date: _____

What is your occupation?

Newsletter number of latest Technical Newsletter (if any) concerning this publication:

How did you use this publication?

[ ]	As an introduction	[ ]	As a text (student)
[ ]	As a reference manual	[ ]	As a text (instructor)
[ ]	For another purpose (explain)		

_____

_____

Is there anything you especially like or dislike about the organization, presentation, or writing in this manual? Helpful comments include general usefulness of the book; possible additions, deletions, and clarifications; specific errors and omissions.

Page Number: Comment:

Name	Address
Company or Organization	
Phone No.	

Readers' Comments — We'd Like to Hear from You
CLIO-user-01

IBM®

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

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

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

IBM®

Program Number: 5648-129

Printed in U.S.A.

CLIO-USER-01