CERN Program Library Long Writeup Q124

CSPACK

Client-Server

Routines and Utilities

Version 1.35 (June 1995)

Application Software Group

Computers and Network Division

CERN Geneva, Switzerland

```
CERN Program Library Office
CERN-CN Division
CH-1211 Geneva 23
Switzerland
Tel.  +41 22 767 4951
Fax.  +41 22 767 7155
Bitnet:  CERNLIB@CERNVM
DECnet:  VXCERN::CERNLIB (node 22.190)
Internet:  CERNLIB@CERN.CH
```


Contact Person:	Jamie Shiers /CN	`(Jamie.Shiers@CERN.CH)`
Technical Realization:	Michel Goossens /CN	`(Michel.Goossens@CERN.CH)`

Preliminary remarks

This **Complete Reference** of the CSPACK system, consists of four parts:

1 An **overview** of the system.
2 A **step by step tutorial introduction** to the system.
3 A **reference guide**, describing each command in detail.
4 An **installation and management guide**.

The CSPACK system is implemented on various mainframes and personal workstations. In particular, versions exist for IBM VM/CMS, VAX/VMS and various Unix-like platforms, such as APOLLO, Cray, DECstation 3100, IBM RS6000, Silicon Graphics, MIPs and SUN.

This document has been produced using LaTeX [1] with the `CERNMAN` style option. All pictures shown are produced with PAW and are included in PostScript [2] format in the manual.

Throughout this manual, commands to be **entered** are `underlined`

Acknowledgements

Many people have contributed to the CSPACK package. The main authors are: René Brun, Olivier Couet, Mike Gerard, Frédéric Hemmer, Burkhard Holl, Catherine Magnin, Ben Segal, Jamie Shiers, Jonathan Wood (Rutherford Appleton Laboratory, UK)

The author is grateful to the many people who contributed to the CSPACK project, through discussions or by providing code and assistance.

I am particularly grateful to Michel Goossens as document editor.

Related Documents

This document can be complemented by the following documents:

- FATMEN User Guide [3]
- CMZ User Guide [4]
- PATCHY User Guide [5]
- HBOOK User Guide [6]
- PAW User Guide [7]
- KUIP - Kit for a User Interface Package [8]
- ZEBRA - Data Structure Management System [9]
- The FATMEN Report [10]
- TMS - The CERN Tape Management System [11]
- The MUSCLE Report [12]
- Computing at CERN in the 1990s [13]

Table of Contents

List of Tables

Part I

CSPACK – Overview

Chapter 1: GLOSSARY

1.1 Software packages used in High Energy Physics

A short description of packages referred to in this document are given below.

1.1.1 ZEBRA - The data structure management system

The data structure management package ZEBRA was developed at CERN in order to overcome the lack of dynamic data structure facilities in FORTRAN, the favourite computer language in high energy physics. It implements the **dynamic creation and modification** of data structures at execution time and their transport to and from external media. ZEBRA input/output is either by a sequential or direct access method. Two data representations, **native** (no data conversion when transferred to/from the external medium) and **exchange** (a conversion to/from an interchange format is made if necessary), allow data to be transported between computers of the same and of different architectures.

Many of the packages described below are based on Zebra.

1.1.2 EPIO - A machine independant input/output package

EPIO is an input/output package still in use by some experiments at CERN. CSPACK provides remote file transfer and access for EPIO files.

1.1.3 KUIP - The user interface package

The purpose of KUIP (**K**it for a **U**ser **I**nterface **P**ackage) is to handle the dialogue between the user and the application program It parses the commands input into the system, verifies them for correctness and then hands over control to the relevant action routines.

1.1.4 HBOOK - The histogramming package

HBOOK provides a library of FORTRAN callable routines for the manipulation of histograms, scatter plots, tables and ntuples. These may be stored on disk files using the RZ direct access routines of the ZEBRA package.

1.1.5 PAW - The Physics Analysis Workstation

The PAW system is widely used by physicists to perform interactive data analysis and presentation. It uses the facilities provided by packages such as HBOOK, KUIP and of course ZEBRA.

1.1.6 FATMEN - A Distributed File and Tape Management System

The FATMEN system provides a fully distributed file catalogue and file access in a location, operating system and device independent manner. The ZEBRA RZ package is used to store the file catalogue information. The CSPACK facilities are also used by FATMEN for catalogue update distribution, remote file access and remote data file access.

1.1.7 PATCHY - The Source Code Management System

PATCHY is a source code management system which has been in use for many years. Files may be stored in a number of formats: CARD files, compact binary PAM files or in CETA format. All of the above formats may be transferred between different machines by tools in the CSPACK package.

1.1.8 CMZ - A Code Management system using ZEBRA

CMZ is an advanced Code Management system, backward compatible with PATCHY, that is based on ZEBRA. As with HBOOK, the ZEBRA RZ package is used to store data on disk.

1.2 Components of the CSPACK system

1.2.1 CZ - The ZEBRA Communications Package

The CZ package is a small set of FORTRAN callable routines used by FATMEN, PAW and other applications. It provides a simple means of starting a remote server and then exchanging character or binary data. The actual communication is performed by TCPAW, running over TCP/IP, or transparent DECnet task-to-task.

1.2.2 XZ - The remote I/O package

XZ is a small package built on top of CZ which permits remote I/O, such as OPEN, CLOSE, READ, WRITE etc. and remote file transfer.

1.2.3 TCPAW - The Networking Package

TCPAW provides the network layer for many of the tools in the current CSPACK package is built. It consists of FORTRAN callable C routines, and is implemented on a variety of platforms, including VM/CMS, VAX/VMS, and Unix systems.

TCPAW uses the internet daemon (INETD) to start servers, except on VM/CMS, where REXEC is used.

1.2.4 SYSREQ - The System Service Request Facility

SYSREQ is a facility developed at RAL for generalised inter-system communications. It allows commands to be sent to, and replies received from, services running in dedicated service machines under the VM/CMS. For example, all communication with the HEPVM Tape Management System (TMS), that was developed at the Rutherford Appleton Laboratory in the UK and is now running at several of the larger HEPVM sites, is via SYSREQ. At CERN, a facility has been developed to permit remote users use the facilities of SYSREQ, by forwarding the messages and replies over TCP/IP. This system is known as SYSREQ-TCP.

1.2.5 TELNETG - A extended TELNET program

TELNETG is a modified version of the standard TELNET program that allows the input/output of a HIGZ based graphics session on a remote system to be displayed in a graphics window on the local workstation. TELNETG is available for Unix and VAX/VMS systems.

1.2.6 TAGIBM - A 3270 terminal emulator

TAGIBM is a powerful 3270 terminal emulator similar to TELNETG but with full-screen emulation for IBM systems.

1.2.7 INETD - the internet daemon

On all systems except VM/CMS and IBM MVS, the server for ZFTP, distributed PAW and the CZ/XZ FORTRAN routines is started using the internet daemon (INETD), except between VAX/VMS systems when the DECnet option is activated.

The inetd daemon is normally started when your system is rebooted. Once started, the inetd daemon listens for connections on certain Internet sockets specified in the /etc/inetd.conf file. When the inetd daemon receives a request on one of these sockets, it determines what service corresponds to that socket and then either handles the service request itself or invokes the appropriate server, such as ZSERV or PAWSERV.

A separate process exists for each concurrent connection to a given host.

1.2.8 REXEC - the remote execution daemon

As INETD is not available for VM/CMS, another solution has to be used. TCPAW uses the REXEC command to start servers on VM/CMS systems. The REXEC daemon autologs the machine of the specified user, having verified the username and password. This means that the machine in question must not be in use, i.e. logged on or disconnected. Once the machine is autologged, the ZSERV or PAWSERV program is started.

If you have problems connecting to a remote VM system, first check that the account is not in use. If you still have problems, ensure that your PROFILE EXEC does not contain any statements which cause it to run a command, e.g. EXEC MAIL, either unconditionally or in DISCONNECTED mode.

Chapter 2: Alternatives and recommendations

Many of the tools developed in this package were first written in the framework of the PAW [7] project. They were extended and enhanced for the FATMEN [3] project. The tools are based on such de-facto standards as DECnet and TCP/IP sockets. However, new standards are now emerging which, together with enhancements to HEP packages, render parts of CSPACK redundant. Some of these are described below.

The main recommendations are:

- Use exchange data format whereever possible
- Only use files with fixed length records and no control words

2.1 ftp transfer

Unformatted files in exchange mode can be transferred using binary ftp between different machines without problems *except*:

1. On VMS systems, one cannot currently specify the format of the target file. The file can be converted on the VMS end using the **RESIZE** command, which invokes the following command file:

CERNLIB RESIZE command

```
$!*****************************************************************
$!*                                                               *
$!* RESIZE.COM v1.02                                              *
$!*                                                               *
$!* Resize ftp-files                                              *
$!* Author: J.Zoll 90/07/24                                       *
$!*                                                               *
$!* Mods       Date   Comments                                    *
$!* MARQUINA 92/12/07 Add cosmetics for public release            *
$!* M.Kelsey 93/10/01 Prevent clashes on simultaneous runs        *
$!*                                                               *
$!*****************************************************************
$ ver_proc=F$VERIFY(0)
$ SAY   :== WRITE/SYMBOL SYS$OUTPUT
$ If p1.eqs.""
$ Then Say "%DCL-I-SYNT syntax: resize [-s size] input_file [output_file]"
$      Exit
$ Endif
$ On ERROR     Then Goto EXIT
$ On CONTROL_Y Then Goto EXIT
$!
$      ifile=p1
$      ofile=p2
$      size =3600
$ If p1.eqs."-S"
$ Then ifile=p3
$      ofile=p4
$      size =p2
$ Endif
```

```
$ If ofile.eqs."".or.ofile.eqs."-" Then ofile=ifile
$
$ ffile="EXCH_"+F$GETJPI("","PID")+".DAT"
$ OPEN/WRITE OUTP 'ffile
$ WRITE OUTP    "RECORD"
$ WRITE OUTP    "BLOCK_SPAN             yes"
$ WRITE OUTP    "CARRIAGE_CONTROL       none"
$ WRITE OUTP    "FORMAT                 fixed"
$ WRITE OUTP    "SIZE                   ''size'"
$ CLOSE OUTP
$
$ Say "resize: setting record size of ",ofile," to ",size," bytes..."
$ EXCHANGE/NETWORK 'ifile 'ofile -
        /TRANSFER=BLOCK -
        /FDL='ffile
$!
$ purge/nolog 'ofile'
$ EXIT:
$ DELETE/NOCONF/NOLOG 'ffile'.*
$ dummy=F$VERIFY(ver_proc)
$ Exit
```

2. With version 3.3 (February, 1994) of TGV Multinet TCP/IP software for OpenVMS, it is possible to set the record size for binary transfers in FTP (previously the only allowed sizes were 512 and 2048 bytes). The commands are RECORD-SIZE nnnnn when the ftp session originates on the openVMS computer, and QUOTE SITE RMS RECSIZE nnnnn when ftp is started from the non-VMS end. Here are examples of the two cases.

For a transfer starting on an OpenVMS computer

```
$ FTP SHIFT.CERN.CH
SHIFT.CERN.CH> USER yourname
<Password required for yourname.
Password:
<User leeiv logged in.
SHIFT.CERN.CH> VERSION
DUKPHY.PHY.DUKE.EDU MultiNet FTP user process 3.3(109)
SHIFT.CERN.CH> BINARY
Type: Image, Structure: File, Mode: Stream
SHIFT.CERN.CH> RECORD-SIZE 32400
SHIFT.CERN.CH> RECORD-SIZE
Record size for IMAGE files: 32400
SHIFT.CERN.CH> GET myfile.rzhist myfile.rzhist
SHIFT.CERN.CH> QUIT
$
```

For a transfer starting from a non-Multinet computer

```
> ftp dukphy.phy.duke.edu
Connected to dukphy.phy.duke.edu.
220 DUKPHY.PHY.DUKE.EDU MultiNet FTP Server Process 3.3(14) at Mon 13-Jun-94
Name: yourname
331 User name (yourname) ok. Password, please.
Password:
ftp> binary
200 Type I ok.
ftp> quote site rms recsize 32400
200 IMAGE file record size now 32400 bytes
ftp> put myfile.rzhist myfile.rzhist
ftp> quit
```

3. On Unix systems, ZEBRA FZ files should be read and written with C I/O (option L in call to `FZFILE`.
4. On VM/CMS systems, one can specify the record format of the target file using an ftp subcommand such as the following:

   ```
   bin f 3600
   ```

   If connecting to a VM/CMS system, then use

   ```
   bin

   quote site fix 3600
   ```

2.2 NFS access

The rules for NFS access are as above, with the additional proviso that C I/O should also be used on VMS systems in all cases where the *record format* of the data file is **STREAM˙LF**.

2.3 ZEBRA RZ files

ZEBRA RZ files may now be written in both exchange and native data formats. For systems that use the IEEE floating point format, such as most Unix systems including Sun, Apollo, RS6000 etc., native and exchange formats are identical. It is recommended that exchange data format be used whereever possible. Such files may be transferred between different systems using the standard `ftp` utility and accessed at the record level using `nfs`. This obviates the need for the `GETRZ` and `PUTRZ` commands in `ZFTP`, for example.

2.4 ZEBRA FZ files

Exchange format has always existed for ZEBRA FZ files. However, due to limitations of certain FORTRAN implementations, such files have not been easily transferable to/from these systems. (FORTRAN typically writes control words at the beginning and end of each record in sequential files on most Unix systems. These control words render the file unreadable from other systems across NFS, or if the file is transferred using FTP without further conversion). ZEBRA FZ has now been enhanced to provide I/O using the C run time library (or FORTRAN direct-access I/O). Files written with either of these options maybe be shared across systems using NFS or transferred using FTP without further conversion.

2.5 PATCHY files

PATCHY files may be kept in binary (PAM) or formatted (CARD) files. Card files may be shared across systems without problems, unless certain special characters are used. The use of card files removes the need for the ZFTP GETP and PUTP commands.

Chapter 3: Introduction

Many High Energy physics experiments use some or all of the following packages:

- PATCHY or CMZ for code management
- Zebra FZ and RZ packages for I/O
- PAW, HBOOK for histogramming

The transfer of the files used by these packages is often difficult, and network access impossible.

For example, PATCHY PAM files have are normally transferred between different machines in a special interchange format, known as CETA. Network access to PAM files between different hardware platforms is not supported. The transfer of Zebra files also requires the use of an interchange format, Zebra binary (or even ASCII) exchange format. This requires a three step process to transfer a file:

- Convert to exchange format
- Transfer
- Convert back to native

Trasnfer of such files to and from Unix machines is further complicated by the fact that that data records, when written by FORTRAN, contain control information which renders the file unreadable on the remote system and so a further step is required to add or remote these control words.

3.1 CSPACK

CSPACK is designed to solve the above problems, by providing network transfer and access to the commonly used HEP formats with transparent, on the fly data conversion. This is performed through a file transfer program called ZFTP.

In addition, a FORTRAN callable interface allows users to code their own applications, or call directly routines that provide complete file transfer of ASCII, binary, FORTRAN direct-access, Zebra FZ or RZ and PAM files. Routines for record level access also exist.

CSPACK also includes other tools and routines for the distributed computing environment, such as the TELNETG program, which permits a graphics application, such as PAW or GEANT, to be run on a remote machine utilising a graphics window on the local workstation.

Part II

CSPACK – Tutorial

Chapter 4: A tutorial introduction to CSPACK

4.1 File transfer using the ZFTP program

ZFTP is a file transfer program which supports the transfer of formatted, unformatted and ZEBRA RZ files (CMZ, HBOOK, etc.). Formatted files are typically KUIP macros, command or EXEC files, source code or even FZ ALPHA exchange format files. Unformatted files may be FZ binary exchange format or EPIO files, or any other binary file with fixed length records. ZFTP also provides LS, CD, PWD and RSHELL commands. It provides a common interface on all systems and the possibility of macros (via KUIP). It avoids the problems of file format conversion that occur when transferring binary files to UNIX systems.

PAM files may be transferred in CARD, CETA, CMZ or even compact binary PAM format. (In the first release this last format is limited to 32 bit machines.) This is particularly convenient for software distribution amongst disparate hardware types.

Advantages of using ZFTP

The advantages of using ZFTP are best explained via examples. Suppose one creates an HBOOK file (which is stored in ZEBRA RZ format) on the IBM and that is required on a VAX. Without ZFTP, the file must first be converted to sequential format using the RTOX utility. The output file can then be transferred to the VAX via Interlink or standard FTP but must then be reconverted to RZ format using the RFRX utility. This requires extra disk space on both sides for the sequential file and a three-step process. On some UNIX systems the situation is even worse as the file transferred by FTP cannot be read by RFRX but must be converted for a second time.

The same operation using ZFTP is much simpler:

1 The user issues the command *ZFTP nodename* and provides the remote username and password much like with standard FTP.

2 The command *PUTRZ local-file remote-file* is issued.

Using this single stage operation the file is transferred with automatic conversion from IBM to VAX format. As the user interface of ZFTP is based on KUIP, the power of ZFTP may be extended using macros.

Data transfer rates are currently about 2/3 of those obtainable with standard FTP, but the effective transfer rate achieved by the elimination of the conversion to sequential format and back is much higher.

Example of a ZFTP session

```
VXST? zftp crnvmc
system 'crnvmc' service 'zserv'
Remote host/port = crnvmc/17303
Name (crnvmc:jamie):
fmsn201
Password (crnvmc:fmsn201):
crnvmc: loading zserv exec (30 sec timeout)...
Connected to crnvmc on TCP port 1305 at Wed Sep 19 12:43:49 1990
ZFTP>
ZFTP> pwd # Show current working directory
Current working directory is FMSN201 191
ZFTP>
ZFTP> cd fat3.192 # Change to FAT3 192
```

```
Remote directory changed to FAT3 192
ZFTP>
ZFTP> ls cspack.cards -l # Directory listing
CSPACK   CARDS    A1 F         80      10406        204  9/17/90 14:33:37 ZEBRA
ZFTP>
ZFTP> geta cspack.cards = s # Transfer file displaying statistics
 File transfer completed
 Transferred       261512 bytes, transfer rate =    4.636364     KB/S
 Elapsed time = 00:00:55 CP time =    6.590000     sec.
ZFTP>
ZFTP> cd fmsn201 # Back to home directory
Remote directory changed to FMSN201
ZFTP>
ZFTP> ls
FATSENDR EXEC     A1
FMCHARM2 KUMAC    A1
FXFILE   BIG      A1
FXFILE   DAT      A1
GETZS    EXEC     A1
LASTING  GLOBALV  A0
PROFILE  EXEC     A1
ZSERV    MODULE   A1
ZFTP>
ZFTP> getb fxfile.dat = 32400 s # Transfer a ZEBRA FZ exchange file
 File transfer completed
 Transferred           10 records, transfer rate =    31.60000     KB/S
 Elapsed time = 00:00:10 CP time =   0.9100000     sec.
ZFTP>
ZFTP> rm fmcharm2.kumac # Delete a remote file
ZFTP>
ZFTP>
ZFTP> cd fmcndiv
Remote directory changed to FMCNDIV
ZFTP>
ZFTP> getrz cern.fatrz = s
File transfer completed


     NREC    NWORDS    QUOTA(%)  FILE(%)   DIR. NAME
       2       1187      0.00     0.00   //RZ/CNDIV/CERNLIB/SUN
       3       2211      0.00     0.00   //RZ/CNDIV/CERNLIB
       2       1350      0.00     0.00   //RZ/CNDIV/JUDY
       2       1839      0.00     0.00   //RZ/CNDIV/JAMIE
       8       6424      0.01     0.01   //RZ/CNDIV
      12      10520      0.01     0.01   //RZ
 Transferred           41 KB, rate =    8.200000     KB/S
 Elapsed time = 00:00:05 CP time =   0.6700000     sec.
ZFTP>
ZFTP> q
```

Table 4.1: ZFTP commands

Command	**Function**	**Description**
`OPEN`	Open connection	Establish connection to specified host
`CLOSE`	Close connection	Close connection with current host
`GETA/PUTA`	Text file transfer	Text file transfer, e.g. scripts, EXECs, CARD pams etc.
`GETB/PUTB`	Binary file transfer	Binary file transfer (fixed length records only) e.g. ZEBRA FZ binary exchange format, EPIO, CETA files
`GETD/PUTD`	Direct-access file transfer	Direct-access file transfer e.g. ZEBRA RZ file between like machines.
`GETRZ/PUTRZ`	ZEBRA RZ file transfer	RZ file transfer with automatic conversion between different data representations, e.g. HBOOK histogram or ntuple files, CMZ files.
`GETFZ/PUTFZ`	ZEBRA FZ file transfer	FZ file transfer with automatic conversion between different data representations (currently in preparation)
`GETP/PUTP`	Compact binary PAM transfer	Transfer a compact binary PAM file (not yet to Cray)
`CD`	Change working directory	Set working directory on remote node
`LCD`	Change working directory	Set working directory on local node
`LS`	Remote LS command	Make remote directory listing
`LLS`	Local LS command	Make local directory listing
`MPUT`	Put multiple files	Send all files matching the specified pattern to the remote machine. The mode of transfer is determined by the file type: `.CET, .CETA = PUTB, .CMZ = PUTRZ, other = PUTA`
`MGET`	Get multiple files	Retrieve all files matching the specified pattern from the remote machine. The mode of transfer is determined by the file type: `.CET, .CETA = GETB, .CMZ = GETRZ, other = GETA`
`MV`	Move (rename) remote file	
`PWD`	Print remote directory	Display current remote directory
`LPWD`	Print local directory	Display current local directory
`RM`	Remove remote file	Remote file deletion
`LRM`	Remove local file	Local file deletion
`RSH`	Remote shell	Issue command to the remote shell
`VERSION`	Version of ZFTP	Print version of the ZFTP program
`SVERSION`	Version of server	Print version of the remote server (if connected)

4.2 Record transfer using the FORTRAN interface

The following example shows how individual records of a FORTRAN direct-access file may be accessed remotely.

Example of remote access of records in direct-access file

```
      common/pawc/paw(50000)
      parameter (lrecl=4096)
      dimension buff(lrecl)
*
*     Initialise ZEBRA the easy way (get HBOOK to do it for us...)
*
      call hlimit(50000)
*
*     Open the file /fatmen/fmopal/cern.fatrz on node fatcat
*     The record length is 4096 bytes
*
      call xzopen(80,'/fatmen/fmopal/cern.fatrz','fatcat',
     +            lrecl,'D',irc)
      open(81,file='opal.fatrz',access='direct',recl=lrecl)
      nrec = 0
*
*     Now read each record in turn. Error is assumed to be end of file
*
10    continue
      nrec = nrec + 1
      call xzread(80,buff,nrec,lrecl,ngot,' ',irc)
      if(irc.eq.0) then
         write(81,rec=nrec) buff
         goto 10
      endif
*
*     Terminate
*
      call xzclos(80,' ',irc)
      close (81)
      end
```

4.2.1 File transfer using the FORTRAN callable routines

The following program demonstrates file transfer using the FORTRAN callable routines. This program is used to transfer updates to the FATMEN catalogue, which are distributed as ZEBRA FZ files in ASCII exchange format, between CERNVM and the FATMEN server. It performs the following functions:

1 Initialise ZEBRA (via call to HLIMIT)
2 Initialise XZ (define logical units, log level)
3 Open connection to the FATMEN server
4 Call an EXEC that uses WAKEUP to wakeup upon arrival of new files in the RDR, or every hour.
5 If a new file has been received, this is then sent to the appropriate directory on the FATCAT machine.
6 If no new file has been received, or after successfully sending any new files, a search is made in the appropriate directories on the remote node for pending updates for CERNVM.

7 Any such files are transferred, and then the call to WAKEUP is reissued.

8 The program can only exit if a user hits enter on the console of the virtual machine, or if an appropriate SMSG is received from a suitably authorised used.

Example of file transfer using FORTRAN callable routines

```
      PROGRAM FATCAT
*CMZ :          21/02/91  16.24.17  by  Jamie Shiers
*-- Author :    Jamie Shiers   21/02/91
*     Program to move updates between CERNVM and FATCAT
*
      PARAMETER    (NMAX=100)
      CHARACTER*64 FILES(NMAX)
      CHARACTER*8  FATUSR,FATNOD,REMUSR,REMNOD
      CHARACTER*64 REMOTE
      CHARACTER*12 CHTIME
      CHARACTER*8  CHUSER,CHPASS
      CHARACTER*80 CHMAIL,LINE
      COMMON/PAWC/PAW(50000)
      PARAMETER    (IPRINT=6)
      PARAMETER    (IDEBUG=3)
      PARAMETER    (LUNI=1)
      PARAMETER    (LUNO=2)
      COMMON /QUEST/ IQUEST(100)
      COMMON/SLATE/IS(6),IDUMM(34)
*
*     Initialise ZEBRA
*
      CALL HLIMIT(50000)
*
*     Initialise XZ
*
      CALL XZINIT(IPRINT,IDEBUG,LUNI,LUNO)
*
*     Open connection to FATCAT...
*
      CALL CZOPEN('zserv','FATCAT',IRC)

    1 CALL VMCMS('EXEC FATSERV',IRC)
      IF(IRC.EQ.3) GOTO 2
      IF(IRC.NE.0) THEN
         PRINT *,'FATCAT. error ',IRC,' from FATSERV. Stopping...'
         GOTO 99
      ENDIF

*
*     Get the user and node name for this file...
*
      CALL VMCMS('GLOBALV SELECT *EXEC STACK FATADDR',IC)
      CALL VMRTRM(LINE,IEND)
      ISTART = ICFNBL(LINE,1,IEND)
      CALL FMWORD(FATUSR,0,' ',LINE(ISTART:IEND),IC)
      LFAT   = LENOCC(FATUSR)
      CALL FMWORD(FATNOD,1,' ',LINE(ISTART:IEND),IC)
      LNOD   = LENOCC(FATNOD)
```

```
      PRINT *,'FATCAT. Update received from ',FATUSR(1:LFAT), ' at ',
     +         FATNOD(1:LNOD)

      CALL DATIME(ID,IT)
      WRITE(CHTIME,'(I6.6,I4.4,I2.2)') ID,IT,IS(6)
*
*    Now put this file...
*    This assumes the FATCAT naming convention: /fatmen/fmgroup,
*                                         e.g. /fatmen/fml3
*
      REMOTE = '/fatmen/'//FATUSR(1:LFAT)//
     +         '/todo/'//FATUSR(1:LFAT)//'_'
     +         //FATNOD(1:LNOD)//'.'//CHTIME
      LREM   = LENOCC(REMOTE)

      CALL XZPUTA('FATMEN.RDRFILE.A',REMOTE(1:LREM),' ',IC)
      IF(IC.NE.0) THEN
         PRINT *,'FATCAT. error ',IC,' sending update from ',
     +            FATUSR,' at ',FATNOD,' to FATCAT'
         CALL VMCMS('#CP LOGOFF',IC)
      ENDIF
      CALL VMCMS('ERASE FATMEN RDRFILE A',IC)

*
*     Are there any files for us to get?
*
    2 CONTINUE
      ICONT  = 0
      NFILES = 0
      CALL XZLS('/fatmen/fm*/tovm/*',FILES,NMAX,NFILES,ICONT,' ',IC)
      IF(ICONT.NE.0) THEN
         PRINT *,'FATSRV. too many files - excess names ',
     +   'will be flushed'
*
   10    CONTINUE
         CALL CZGETA(CHMAIL,ISTAT)
         LCH = LENOCC(CHMAIL)
         IF(CHMAIL(1:1).EQ.'0') THEN
*
*        Nop
*
         ELSEIF(CHMAIL(1:1).EQ.'1') THEN
         ELSEIF(CHMAIL(1:1).EQ.'2') THEN
            GOTO 10
         ELSEIF(CHMAIL(1:1).EQ.'3') THEN
            IQUEST(1) = 1
            IRC = 1
         ELSEIF(CHMAIL(1:1).EQ.'E') THEN
            IQUEST(1) = -1
            IRC = -1
         ELSEIF(CHMAIL(1:1).EQ.'V') THEN
            GOTO 10
         ELSE
            IQUEST(1) = 1
            IRC = 1
         ENDIF
*
```

```
      ENDIF


      DO 3 I=1,NFILES
      LF = LENOCC(FILES(I))
      CALL CLTOU(FILES(I))
*
*     Fix for the case when there are no files...
*
      IF((NFILES.EQ.1).AND.
     +   (INDEX(FILES(I)(1:LF),'DOES NOT EXIST').NE.0)) GOTO 1
*
*     Remote file syntax is /fatmen/fm*/tovm
*
      ISLASH = INDEXB(FILES(I)(1:LF),'/')
      IF(INDEX(FILES(I)(ISLASH+1:LF),FATNOD(1:LNOD)).NE.0) THEN
         PRINT *,'FATCAT. skipping update for ',FATNOD(1:LNOD),
     +           '(',FILES(I)(1:LF),')'
         GOTO 3
      ENDIF
*
*     Get the name of the server for whom this update is intended...
*
      ISTART = INDEX(FILES(I)(1:LF),'/FM') + 1
      IEND   = INDEX(FILES(I)(ISTART:LF),'/')
      REMUSR = FILES(I)(ISTART:ISTART+IEND-2)
      LREM   = LENOCC(REMUSR)

      PRINT *,'FATCAT. update found for ',REMUSR(1:LREM),
     +           '(',FILES(I)(1:LF),')'

      CALL XZGETA('FATMEN.UPDATE.B',FILES(I)(1:LF),' ',IC)
      IF(IC.NE.0) THEN
         PRINT *,'FATCAT. error ',IC,' retrieving update'
         GOTO 99
      ENDIF

      CALL VMCMS('EXEC SENDFILE FATMEN UPDATE B TO '
     +           //REMUSR(1:LREM),IC)

      CALL XZRM(FILES(I)(1:LF),IC)
      IF(IC.NE.0) PRINT *,'FATCAT. error ',IC,' deleting file ',
     +           '(',FILES(I)(1:LF),')'

3     CONTINUE
*
*     Wait for some action...
*
      GOTO 1

   99 CALL CZCLOS(ISTAT)
      END
```

Part III

CSPACK – User Guide

Chapter 5: ZFTP

ZFTP is a file transfer program tuned to the needs of the HEP environment. Using the standard FTP program, files often reach the remote system in an unreadable format. This is due to differences such as file format (presence or absence of FORTRAN control words etc.) or, more importantly, differences used in the internal representation of data between machines. When files are transferred between systems using ZFTP, these problems are solved. Not only is data conversion performed automatically on the fly, but the file format is such that no further manipulation is required before processing with standard programs. Thus, an ntuple file produced on the Cray may be transferred to an Apollo workstation with a single command.

In addition, considerable advantages arise from the standard interface on all systems and the power of KUIP which provides macros and many other facilities. All of the functionality is also available through FORTRAN routines which are available as part of PACKLIB. This is used to advantage in the FATMEN package, to provide convenient remote file transfer.

The ZFTP program is started by typing the command `zftp`. As with the standard ftp program, if the nodename is given on the command-line, a connection will be established to that node, e.g. `zftp vxcrnb` Otherwise, use the OPEN command to establish a connection to a remote machine.

Valid options are described in the description of the OPEN command.

5.1 File conversion and commands

RFRF `FZFILE RZFILE [LRECL] [CHOPT]`

`FZFILE` File name of the input FZ file

`RZFILE` File name for the output RZ file

`LRECL` Record length for the output RZ file in bytes. If zero is specified, the record length of the original RZ file will be used.

`CHOPT` List of options

- `A` the input file is in FZ alpha format
- `S` display statistics on the RZ file
- `X` the RZ file will be created in eXchange mode
- `C` respect case of file names

This command converts an FZ exchange format file to an RZ file on the LOCAL machine. No network transfer is performed. The FZFILE must be the output of a previous RTOF command, or have been created using the RTOX or RTOA programs. On Unix systems, this file will be read with FORTRAN direct-access and will hence be transferable and readable on other systems.

By default, the output RZ file will have the same record length as the original RZ file. However, if LRECL is specified then this value will be used instead.

RTOF `RZFILE FZFILE [LRECL] [CHOPT]`

`RZFILE` File name of the input RZ file

`FZFILE` File name for the output FZ file

`LRECL` Record length for the output FZ file in bytes. If zero is specified, a record length of 3600 bytes will be used for binary files, or 80 bytes for ASCII files.

`CHOPT` List of options

- `A` the output file is to be in FZ alpha format
- `S` display statistics on the RZ file
- `C` respect case of file names

This command converts an RZ file into an FZ exchange mode format file on the LOCAL machine. No network transfer is performed. By default a binary exchange mode FZ file is created. On Unix systems, this file will be written with FORTRAN direct-access and will hence be transferable and readable on other systems.

FZCOPY `FZIN FZOUT [IFORM] [IRECL] [OFORM] [ORECL] [CHOPT]`

`FZIN` input FZ file name

`FZOUT` output FZ file name

`IFORM` Format of input FZ file

`IRECL` Input record length (in bytes)

`OFORM` Format of output FZ file

`ORECL` Output record length (in bytes)

`CHOPT` List of options

- `A` lpha exchange mode format - RECL not needed
- `N` native data but exchange file format - RECL not needed
- `X` exchange format file - RECL not needed
- `Z` native data and file format - RECL must be specified

This command copies an FZ file on the local machine. At the same time, file format and data format conversion is possible. Thus, FZCOPY can be used to convert a binary native format file into a alpha exchange format file etc.

RZCOPY `RZIN RZOUT [ORECL] [CHOPT]`

`RZIN` input RZ file name

`RZOUT` output RZ file name

`ORECL` Output record length (in bytes). If not specified, the record length of the input file will be used.

`CHOPT` List of options

- `N` convert exchange RZ file into native RZ file
- `X` convert native RZ file into exchange RZ file

This command copies an RZ file on the local machine. At the same time, the record length or data format may be changed. Thus, RZCOPY can be used to convert a native format RZ file with record length 512 into an exchange format file with record length 8192.
If not specified, the output record length will be set equal to that of the input file.

CTOF `CFILE FFILE [LRECL] [CHOPT]`

`CFILE` input file name

`FFILE` output file name

`LRECL` record length (in bytes)

`CHOPT` List of options

- `X` Zebra exchange format file - RECL not needed
  other files - RECL must be specified

This command copies a file written with C or FORTRAN direct-access I/O to one written with FORTRAN sequential I/O.

FTOC `FFILE CFILE [LRECL] [CHOPT]`

`FFILE` input file name

`CFILE` output file name

`LRECL` record length (in bytes)

`CHOPT` List of options

- `X` Zebra exchange format file - RECL not needed
  other files - RECL must be specified

This command copies a file written with FORTRAN sequential I/O to one written with FORTRAN direct-access I/O. The output file may be read with C I/O or FORTRAN direct access.

5.2 File transfer commands

GETA `REMOTE LOCAL [CHOPT]`

`REMOTE` Remote file name

`LOCAL` Local file name

`CHOPT` List of options

- `S` Print statistics on the file transfer
- `V` Create the remote file with variable length record format

Transfer a text file from the remote machine to a local file. If the local file name is not given, or a = sign specified, the local file will have the same name as on the remote system.

PUTA `LOCAL REMOTE [CHOPT]`

`LOCAL` Local file name

`REMOTE` Remote file name

`CHOPT` List of options

- `S` Print statistics on the file transfer
- `V` Create the remote file with variable length record format

Transfer a text file from the local machine to the remote system. If the remote file name is not given, or a = sign specified, the remote file will have the same name as on the local system.

GETB `REMOTE LOCAL [LRECL] [CHOPT]`

`REMOTE` Remote file name

`LOCAL` Local file name

`LRECL` Record length in bytes

`CHOPT` List of options

- `S` Print statistics on the file transfer

Transfer a binary file from the remote machine to a local file. If the local file name is not given, or a = sign specified, the local file will have the same name as on the remote system. The file must contain fixed length blocks (EPIO or FZ exchange format).

PUTB `LOCAL REMOTE [LRECL] [CHOPT]`

`LOCAL` Local file name

`REMOTE` Remote file name

`LRECL` Record length in bytes

`CHOPT` List of options

- `S` Print statistics on the file transfer

Transfer a binary file from the local machine to the remote system. If the remote file name is not given, or a = sign specified, the remote file will have the same name as on the local system. The file must contain fixed length blocks (EPIO or FZ exchange format).

GETD `REMOTE LOCAL LRECL [CHOPT]`

`REMOTE` Remote file name

`LOCAL` Local file name

`LRECL` Record length in bytes

`CHOPT` List of options

- `S` Print statistics on the file transfer

Transfer a binary direct access file from the remote machine to a local file. If the local file name is not given, or a = sign specified, the local file will have the same name as on the remote system. The file must contain fixed length blocks (EPIO or FZ exchange format).

PUTD `LOCAL REMOTE LRECL [CHOPT]`

`LOCAL` Local file name

`REMOTE` Remote file name

`LRECL` Record length in bytes

`CHOPT` List of options

- `S` Print statistics on the file transfer

Transfer a binary direct access file from the local machine to the remote system. If the remote file name is not given, or a = sign specified, the remote file will have the same name as on the local system. The file must contain fixed length blocks (EPIO or FZ exchange format).

GETP `REMOTE LOCAL [CHOPT]`

`REMOTE` Remote file name

`LOCAL` Local file name

`CHOPT` List of options

- `S` Print statistics on the file transfer

Transfer a compact binary PAM file from the remote machine to a local file. If the local file name is not given, or a = sign specified, the local file will have the same name as on the remote system.

PUTP `LOCAL REMOTE [CHOPT]`

`LOCAL` Local file name

`REMOTE` Remote file name

`CHOPT` List of options

- `S` Print statistics on the file transfer

Transfer a compact binary PAM file from the local machine to the remote system. If the remote file name is not given, or a = sign specified, the remote file will have the same name as on the local system.

GETFZ `REMOTE LOCAL RRECL RFORM LRECL LFORM [CHOPT]`

`REMOTE` Remote file name

`LOCAL` Local file name

`RRECL` Record length of the remote file in bytes

`RFORM` Format of the remote file

- Native file format - record length required
- `A` Alpha exchange format - record length forced to be 80 bytes
- `D` Direct access I/O - ignored if option X is not also specified

- `Z` Native file format - record length required
- `X` Binary exchange format - record length will be obtained from the file itself if not specified.

`LRECL` Record length of the local file in bytes

`LFORM`
- Native file format - record length required
- `A` Alpha exchange format - record length forced to be 80 bytes
- `D` Direct access I/O - ignored if option X is not also specified
- `Z` Native file format
- `X` Binary exchange format

`CHOPT` List of options
- `S` Print statistics on the file transfer

Transfer a ZEBRA FZ file from the remote machine to the local system. If the local file name is not given, or a = sign specified, the local file will have the same name as on the remote system. The FZ file is created on the local computer with the same parameters as on the remote machine. If the format of the local and remote files are not specified, this command file copies a remote native format file to a local native format file.

PUTFZ `LOCAL REMOTE LFORM LRECL RFORM RRECL [CHOPT]`

`LOCAL` Local file name

`REMOTE` Remote file name

`LRECL` Record length of the local file in bytes

`LFORM`
- Native file format - record length required
- `A` Alpha exchange format - record length forced to be 80 bytes
- `D` Direct access I/O - ignored if option X is not also specified
- `Z` Native file format - record length required
- `X` Binary exchange format - record length will be obtained from the file itself if not specified.

`RRECL` Record length of the remote file in bytes

`RFORM` Format of the remote file
- Native file format - record length required
- `A` Alpha exchange format - record length forced to be 80 bytes
- `D` Direct access I/O - ignored if option X is not also specified
- `Z` Native file format
- `X` Binary exchange format

`CHOPT` List of options
- `S` Print statistics on the file transfer

Transfer a ZEBRA FZ file to the remote machine from the local system. If the remote file name is not given, or a = sign specified, the remote file will have the same name as on the local system. The FZ file is created on the remote computer with the same parameters as on the local machine. If the format of the local and remote files are not specified, this command file copy a local native format file to a remote native format file.

GETRZ `REMOTE LOCAL [CHOPT]`

`LOCAL` Local file name

`REMOTE` Remote file name

`CHOPT` List of options

- `R` the local file will have RELATIVE organisation (VAX)
- `L` a list of the top level directories in the received file is displayed.
- `T` the entire directory tree is displayed.
- `S` Print statistics on the file transfer

PUTRZ `LOCAL REMOTE [CHOPT]`

`REMOTE` Remote file name

`LOCAL` Local file name

`CHOPT` List of options

- `R` the remote file will have RELATIVE organisation (VAX)
- `L` a list of the top level directories in the received file is displayed.
- `T` the entire directory tree is displayed.
- `S` Print statistics on the file transfer

Transfer a local RZ file to the remote machine. If the local file name is not given, or a = sign specified, the local file will have the same name as on the remote system. The RZ file is created on the remote computer with the same parameters as on the local machine.

GETX `REMOTE LOCAL LRECL [CHOPT]`

`REMOTE` Remote file name

`LOCAL` Local file name

`LRECL` Record length in bytes

`CHOPT` List of options

- `S` Print statistics on the file transfer

Transfer a binary direct access file from the remote machine to a local file. If the local file name is not given, or a = sign specified, the local file will have the same name as on the remote system. The file must contain fixed length blocks (EPIO or FZ exchange format). The `GETX` command uses Fortran sequential I/O on all systems except Unix, where files are processed with direct access I/O to avoid the control words that are written at the beginning and end of each record with binary sequential Fortran I/O.

PUTX `LOCAL REMOTE LRECL [CHOPT]`

`LOCAL` Local file name

`REMOTE` Remote file name

`LRECL` Record length in bytes

`CHOPT` List of options

- `S` Print statistics on the file transfer

Transfer a binary direct access file from the local machine to the remote system. If the remote file name is not given, or a = sign specified, the remote file will have the same name as on the local system. The file must contain fixed length blocks (EPIO or FZ exchange format). The `PUTX` command uses Fortran sequential I/O on all systems except Unix, where files are processed with direct access I/O to avoid the control words that are written at the beginning and end of each record with binary sequential Fortran I/O.

MGET `REMOTE LOCAL [CHOPT]`

`REMOTE` Remote file name

`LOCAL` Local file name

`CHOPT` List of options

- `S` Print statistics on the file transfer

Transfer all files matching the specified remote file name to the local system. The file name given may contain *, to match one or more characters, or character. By default the transfer is performed using GETA, unless the file name has a known extension.

```
e.g.
*.PAM         --> GETP
*.CETA, *.CET --> GETB, LRECL=3600
*.CMZ,  *.RZ  --> GETRZ
```

MPUT `LOCAL REMOTE [CHOPT]`

`LOCAL` Local file name

`REMOTE` Remote file name

`CHOPT` List of options

- `S` Print statistics on the file transfer

Transfer all files matching the specified local file name to the remote system. The file name given may contain *, to match one or more characters, or character. By default the transfer is performed using PUTA, unless the file name has a known extension.

```
e.g.
*.PAM         --> GETP
*.CETA, *.CET --> GETB, LRECL=3600
*.CMZ,  *.RZ  --> GETRZ
```

5.3 General commands

RSHELL `COMMAND`

`COMMAND` Command to be executed on the remote machine

the specified command is transmitted for execution to the remote machine.

OPEN `MACHINE [CHOPT]`

`MACHINE` Name of remote machine

`CHOPT` List of options

- Use TCP/IP to connect to remote systems.
- `D` Use DECnet to connect to remote systems. Only valid between VAX/VMS systems
- `V` The remote system is running VM/CMS. This option is required unless the remote node is known to the CSPACK software
- `M` The remote system is running MVS. This option is required unless the remote node is known to the CSPACK software

Opens a communication with the remote machine named MACHINE. MACHINE may be an alphanumeric host name or a TCP/IP address (e.g. CERNVM, 128.141.1.181) This command will prompt you for user authentification. Normally, a server is started by software known as the Internet Daemon, or inetd. This is not available with certain versions of TCP/IP, notably DEC/UCX, IBM VM/CMS and MVS. On VM/CMS systems only, the server is started using the REXEC remote execution client, supplied as part of IBM's TCP/IP software. More information on the inetd can be obtained by typing ”man inetd” on a Unix system.

CLOSE

Close communication with the current remote host.

CD `[PATHNAME] [PASSWORD] [CHOPT]`

`PATHNAME` Pathname

`PASSWORD` Password - for password protected VM/CMS minidisks only

`CHOPT` List of options

- `C` Case sensitive directory name. If not specified, pathnames are folded to lower case on Unix systems.

Change remote working directory. If a pathname is not specified, the current working directory is displayed.

```
ZFTP>cd JaMiE -c
```

On remote VM systems, one can change directory to a mini-disk that has a read or write password by specifying the password and access mode required, as in the examples below.

```
ZFTP> cd jamie.400 mypass -r | Read only link

ZFTP> cd fatmen.222 mypass -w | Write link
```

LCD `[PATHNAME] [PASSWORD] [CHOPT]`

`PATHNAME` Pathname

`PASSWORD` Password - for password protected VM/CMS minidisks only

`CHOPT` List of options

- `C` Case sensitive directory name. If not specified, pathnames are folded to lower case on Unix systems.

Change local working directory. If a pathname is not specified, the current working directory is displayed.

PWD

Print remote working directory.

LPWD

Print local working directory.

LS `[PATTERN] [CHOPT]`

`PATTERN` Filenames to list. If not specified, all files in the current working directory will be displayed

`CHOPT` List of options

- `L` Long listing

Issue remote LS command If option -l is given, a 'long listing' will be generated. This corresponds to the Unix ls -l option or the VM/CMS LISTFILE (L command.

LLS `[PATTERN] [CHOPT]`

`PATTERN` Filenames to list. If not specified, all files in the current working directory will be displayed

`CHOPT` List of options

- `L` Long listing

Issue local LS command If option -l is given, a 'long listing' will be generated. This corresponds to the Unix ls -l option or the VM/CMS LISTFILE (L command.

MV `SOURCE TARGET CHOPT`

`SOURCE`

`TARGET`

`CHOPT` List of options

- `C` Respect case of file names (Unix systems)

Move remote file from SOURCE to TARGET.

LMV `SOURCE TARGET CHOPT`

`SOURCE`

`TARGET`

`CHOPT` List of options

- `C` Respect case of file names (Unix systems)

Move local file from SOURCE to TARGET.

RM `FILENAME`

`FILENAME` Filename to be removed Filename 'Filename' C D=' '

Remove (delete) remote file

LRM `FILENAME`

`FILENAME` Filename to be removed Filename 'Filename' C D=' '

Remove (delete) local file

LOGLEVEL `LEVEL`

`LEVEL` Loglevel to set, default=0

Use the LOGLEVEL command to set the level of logging/debug of the ZFTP command.

SVERSION

Print version of server program

VERSION

Print version of client program

Chapter 6: Distributed PAW

Distributed PAW is currently limited to the ability to access remote histogram files, or histograms and ntuples existing in global sections on remote VMS systems. This will be extended over time to provide the equivalent of global sections on Unix systems, and to distribute CPU intensive parts of PAW on mainframes or powerful CPU servers.

Chapter 7: FORTRAN callable interface

The FORTRAN callable interface consists of the CZ and XZ packages. Normally, only the XZ package is of concern to the user: any calls to the CZ package being made in a completely transparent manner. The exception to this case is of course when a new application that requires a different server is to be built.

7.1 Basic client-server routines

7.1.1 Open communication with a remote node

```
CALL CZOPEN (SERVICE,HOST,IRC*)
```

SERVICE	Character variable specifying the name of the service required, e.g. ZSERV
HOST	Character variable specifying the name of the remote host.
IRC	Integer variable in which the return code is returned.

This routine opens a connection with a remote node. A new process is automatically created on the specified node using the username and password that are prompted for at the terminal.

When TCPAW is used as the network layer, usernames and passwords may also be given in a **.netrc** file in the user's home directory (Unix systems). In the case of VAX/VMS systems, the name of this file is .ftplogin;. For VM/CMS systems running the C version of TCPAW, this file is DOT NETRC A0. For a description of the format of these files, see page 66.

In the case of VM systems, the virtual machine of the specified user is autologged. This requires that the user in question is not currently logged on.

Example of using the CZOPEN routine

```
      CALL CZOPEN('ZSERV','CERNVM',IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from CZOPEN
```

To select DECnet instead of TCP/IP as the communications protocol, the variable IPROT in the sequence CZSOCK should be set to 1.

Example of using DECnet as the communications protocal

```
+CDE,CZSOCK. From CSPACK PAM
      IPROT = 1
      CALL CZOPEN('ZSERV','VXCRNA',IRC)
```

When using DECnet as the communications protocol, username and password prompting only occurs for interactive sessions. For other sessions, a server is started using the standard DECnet techniques, i.e. using a `PROXY` account if one exists, or else the default DECnet account.

To disable username and password prompting for interactive sessions, set the logical name `CZPROXY` to `TRUE`, e.g.

Turning off username prompting for DECnet connections

```
DEFINE CZPROXY TRUE
```

7.1.2 Close communication with the current remote node

CALL **CZCLOS** (IRC*)

IRC Integer variable in which the return code is returned.

This routine closes the connection with the current remote node. The process on the remote node is automatically terminated. The current remote node is the one specified in the last call to CZOPEN, or set by the routine CZSWAP.

Example of using the CZCLOS routine

```
      CALL CZCLOS(IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from CZCLOS
```

7.1.3 Switch communication to another node

CALL **CZSWAP** (NODE,LUN,IRC*)

NODE Character variable specifying the node name to which communication should be swapped.

LUN Integer variable specifying the logical unit associated with the remote node.

IRC Integer variable in which the return code is returned.

This routine changes the current node to that associated with the specified logical unit or nodename. If the nodename is non-blank, communication is swapped to the specified node. If the nodename is blank, communication is swapped to the node associated to LUN (e.g. from a call to XZOPEN, see on Page 50). This routine is called automatically by the routines of the XZ package and need normally not be called by a user.

Example of using the CZSWAP routine

```
      CALL CZSWAP(' ',77,IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from CZSWAP
```

7.1.4 Return real time elapsed since last call

CALL **CZRTIM** (ELAPSED*)

ELAPSED Character variable in which the elpased time is returned in the format HH:MM:SS.

The CZRTIM routine is used by the XZGET/PUT routines if the option S is specified in order to print statistics on data transfer rates. This routine must always be called twice: once to start the timer and a second time to return the elapsed time.

Example of using the CZRTIM routine

```
*
*     Start timer
*
      CALL CZRTIM(ELAPSD)
*     Work a little
*     ...
*
*     Get elapsed time since last call
      CALL CZRTIM(ELAPSD)
```

7.1.5 Send text string to current remote node

CALL **CZPUTA** (STRING,IRC*)

STRING Character variable containing the data to be sent to the remote node.

IRC Integer variable in which the return code is returned.

This routine sends a text string to the remote server.

Example of using the CZPUTA routine

```
*
*     Extract from the ZFTP routine ZFTPCD (action routine for
*     the CD command.
*
      CALL CZPUTA('XZIO :CD '//PATH(1:LPATH)',IRC)
      IF(IRC.NE.0) PRINT *,'Return code ',IRC,' from CZPUTA
```

7.1.6 Read text string from remote server

CALL **CZGETA** (STRING,IRC*)

STRING Character variable in which the data read from the remote server is returned.

IRC Integer variable in which the return code is returned.

This routine gets a text string from the remote server. An example of its use in the ZFTP program is shown on the following page.

Example of using the CZGETA routine

```
*
*     Sequence CZMESS from CSPACK - this sequence is used by the
*     various XZ routines to process server messages.
*
+KEEP,CZMESS.
*
*     Process server messages
*
10    CONTINUE
      CALL CZGETA(CHMAIL,ISTAT)
      LCH = LENOCC(CHMAIL)
      IF(CHMAIL(1:1).EQ.'0') THEN
*
*        Nop
*
      ELSEIF(CHMAIL(1:1).EQ.'1') THEN
         PRINT *,CHMAIL(2:LCH)
      ELSEIF(CHMAIL(1:1).EQ.'2') THEN
         PRINT *,CHMAIL(2:LCH)
         GOTO 10
```

```
      ELSEIF(CHMAIL(1:1).EQ.'3') THEN
         PRINT *,CHMAIL(2:LCH)
         IQUEST(1) = 1
         IRC       = 1
      ELSEIF(CHMAIL(1:1).EQ.'E') THEN
         IQUEST(1) = -1
         IRC       = -1
      ELSEIF(CHMAIL(1:1).EQ.'V') THEN
*
*        Number of bytes read from a variable length read
*
         READ(CHMAIL(2:11),'(I10)') NGOT
         GOTO 10
      ELSE
         PRINT *,'Unknown server message ',CHMAIL
         IQUEST(1) = 1
         IRC       = 1
      ENDIF
*
```

7.1.7 Send character array to remote server process

CALL **CZPUTC** (NCHAR,IRC*)

NCHAR — Integer variable giving the number of characters to be sent. The data is in the common block /CZBUFC/ in the character variable CHBUF.

IRC — Integer variable in which the return code is returned.

This routine sends a character string to the remote server.

Example of using the CZPUTC routine

```
      CALL CZPUTC(NTOT,ISTAT)
      IF(ISTAT.NE.0)GO TO 99
```

7.1.8 Get character array from remote server process

CALL **CZGETC** (NCHAR,IRC*)

NCHAR — Integer variable giving the number of characters to be sent. The data is in the common block /CZBUFC/ in the character variable CHBUF.

IRC — Integer variable in which the return code is returned.

This routine reads a character string from the remote server.

Example of using the CZGETC routine

```
      CALL CZGETC(NTOT,ISTAT)
      IF(ISTAT.NE.0)GO TO 99
```

7.1.9 Transfer data between client and server

CALL **CZTCP** (IBUFF,ICONTR)

IBUFF Array containing hollerith or binary data to be sent to the server or received from the server depending on the ICONTR vector.

ICONTR Integer vector of length 2 to determine mode of operation. ICONTR(1) = IMODE, ICONTR(2) = NBYTES

This routine sends or receives data to/from the remote server.

```
IMODE = 0: receive binary
IMODE = 1: send binary
IMODE = 2: receive character data
IMODE = 3: send character data
```

Example of using the CZTCP routine

```
*
*     Send the data
*
            ICONT(1) = 1
            LBUF     = NWORDS
            CALL CZTCP(IBUFF,ICONT)
            ENDIF
```

7.2 Routines to convert or copy files

7.2.1 Convert RZ file to FZ exchange format

CALL **XZRTOF** (CHRZ,CHFZ,LRECL,CHOPT,IRC)

CHRZ Character string giving the name of the RZ file to be converted.

CHFZ Character string giving the name of the output FZ file.

LRECL Integer variable specifying the record length for the output file in bytes. If not specified, a default of 3600 bytes will be used for binary exchange format files and 80 bytes for alpha exchange format files.

CHOPT Character variable specifying the options required

- A Output file should be in alpha exchange format (default is binary).
- C Respect case of input and output file names
- R Replace output file, if it exists

IRC Integer variable in which the completion status is returned.

This routine will convert a ZEBRA RZ file into FZ exchange format. The resultant file may then be transferred to another system and reconverted using XZRFRF.

Example of using the XZRTOF routine

```
*
*     Convert an RZ file to a FZ alpha file
*
      CALL XZRTOF('NTUPLE.DAT','NTUPLE.FA',0,'A',IRC)
```

7.2.2 Convert RZ file from FZ exchange format

CALL **XZRFRF** (CHFZ,CHRZ,LRECL,CHOPT,IRC)

CHFZ Character string giving the name of the FZ file to be converted.

CHRZ Character string giving the name of the output RZ file.

LRECL Integer variable specifying the record length for the output file in bytes. If not specified, the record length of the original RZ file is used.

CHOPT Character variable specifying the options required

- C Respect case of input and output file names
- R Replace output file, if it exists
- X Output file should be in exchange format (default is native).

IRC Integer variable in which the completion status is returned.

This routine will convert a ZEBRA FZ file created using the routine XZRTOF into FZ exchange format.

Example of using the XZRFRF routine

```
*
*     Convert an FZ exchange file back into an RZ file
*     Override the record length in the process
*
      CALL XZRTOF('NTUPLE.FX','NTUPLE.RZ',16384,'X',IRC)
```

CALL **XZCTOF** (CHIN,CHOUT,LRECL,CHOPT,IRC)

CHIN Character string giving the name of the file to be converted.

CHOUT Character string giving the name of the output file.

LRECL Integer variable giving the record length of the input file in bytes. In case of option X, the record length is automatically determined from the file itself.

CHOPT Character string specifying the options required.

- C Respect case of input and output file names
- R Replace output file, if it exists
- X Input file is in ZEBRA exchange format

IRC Integer variable in which the return code is returned.

This routine converts a binary file written with C or FORTRAN direct-access I/O into a file written with FORTRAN sequential I/O. This can be useful on Unix systems, when an FZ or EPIO file that has been transferred from another system is to be read using FORTRAN I/O.

Example of using the XZCTOF routine

```
*
*     Convert an FZ file for processing with FORTRAN
*
      CALL XZCTOF('FXFILE.DAT','FXFILE.OUT',0,'X',IRC)
```

CALL **XZFTOC** (CHIN,CHOUT,LRECL,CHOPT,IRC)

CHIN Character string giving the name of the file to be converted.

CHOUT Character string giving the name of the output file.

LRECL Integer variable giving the record length of the input file in bytes. In case of option X, the record length is automatically determined from the file itself.

CHOPT Character string specifying the options required.

- C Respect case of input and output file names
- R Replace output file, if it exists
- X Input file is in ZEBRA exchange format

IRC Integer variable in which the return code is returned.

This routine converts a binary file written with FORTRAN sequential I/O into a file written with FORTRAN direct access I/O. This can be useful on Unix systems, when an FZ or EPIO file written with FORTRAN sequential I/O is to be transferred to another system.

Example of using the XZFTOC routine

```
*
*     Convert an EPIO file for ftp-ing to another system
*
      CALL XZFTOC('EPIO.DAT','EPIO.OUT',3600,' ',IRC)
```

CALL **XZFZCP** (CHIN,CHOUT,IRECL,IFORM,ORECL,OFORM,CHOPT,IRC)

CHIN Character string giving the name of the file to be copied.

CHOUT Character string giving the name of the output file.

IRECL Integer variable giving the record length of the input file. The record length need only be specified in case of option Z below.

IFORM Character variable giving the format of the output file

- A Input file is in alpha exchange format
- N Input file is in exchange file format, but native data
- X Input file is in binary exchange format
- Z Input file is in native data and file format

`ORECL` Integer variable giving the record length of the output file. If not specified, the input record length will be taken, except for alpha exchange mode files, where a record length of 80 will be used.

`OFORM` Character variable giving the format of the input file

- `A` Output file is in alpha exchange format
- `N` Output file is in exchange file format, but native data
- `X` Output file is in binary exchange format
- `Z` Output file is in native data and file format

`CHOPT` Character string specifying the options required.

- `C` Respect case of input and output file names
- `R` Replace output file, if it exists

`IRC` Integer variable in which the return code is returned.

This routine copies an FZ file on the local machine, with optional format and/or data conversion.

Example of using the XZFZCP routine

```
*
*     Copy an alpha FZ file to a native FZ file
*
      CALL XZFZCP('fafile.dat','fzfile.dat',0,'A',32400,'Z',IRC)
```

CALL **XZRZCP** (CHIN,CHOUT,LRECL,CHOPT,IRC)

`CHIN` Character string giving the name of the file to be copied.

`CHOUT` Character string giving the name of the output file.

`LRECL` Integer variable giving the record length for the output file. The record length of the input file will be used if a value of 0 is given for LRECL.

`CHOPT` Character string specifying the options required.

- `C` Respect case of input and output file names
- `N` Output file should be in native format (default)
- `R` Replace output file, if it exists
- `X` Output file should be in exchange format

`IRC` Integer variable in which the return code is returned.

This routine copies an RZ file on the local machine, with optional data conversion and/or record length conversion.

Example of using the XZRZCP routine

```
*
*     Copy an ntuple, changing the record length and
*     data representation at the same time
*
      CALL XZRZCP('HRZTEST.DAT','hrztest.rz',8192,'CX',IRC)
```

7.3 Routines to transfer files

N.B. for all of the following routines, a connection must first be established using CZOPEN (see on Page 35). All of the following routines return:

```
IRC < 0 : error - explanatory message will be printed by routine
IRC = 0 : success : see statistics in IQUEST
IRC = 1 : cannot open remote file
IRC = 2 : cannot open local  file
IRC = 3 : problem in file transfer

For IRC = 0:

IQUEST(11) = Number of records transferred
IQUEST(12) = Number of kilobytes transferred
IQUEST(13) = Transfer rate in KB/second
IQUEST(14) = Number of hours elapsed (real time)
IQUEST(15) = Number of minutes elapsed (real time)
IQUEST(16) = Number of seconds elapsed (real time)
IQUEST(17) = Number of seconds elapsed (CPU time)
```

N.B. file names for VM/CMS systems should be specified in the form **filename.filetype**[**.filemode**], e.g. PROFILE.EXEC.A. VM mini-disks should be specified in the form [**username**[**.address**]],e.g. [JAMIE], [PUBWS.197]. File transfer to and from VM/CMS systems and access to files stored in VM/CMS systems is only possible to the current 'A-disk', which can be changed using the XZCD routine (see on Page 55).

7.3.1 Get text file

CALL **XZGETA** (LOCAL,REMOTE,CHOPT,IRC)

LOCAL Character variable specifying the local file name. If LOCAL = '=', then the file on the local system will have the same name as on the remote system.

REMOTE Character variable specifying the remote file name.

CHOPT Character variable to specify the options desired.

IRC Integer vriable in which the return code is returned.

This routine gets a text file from the remote system. If option 'S' is specified, statistics on the file transfer are printed. If option 'V' is specified, the local file will have variable length record format (IBM-VM systems only).

Example of using the XZGETA routine

```
      CALL XZGETA('=','CZPACK.CARDS','S',IRC)
      IF(IRC.NE.0) PRINT *,'File transfer failed'
```

7.3.2 Send text file

CALL **XZPUTA** (LOCAL,REMOTE,CHOPT,IRC)

LOCAL	Character variable specifying the local file name.
REMOTE	Character variable specifying the remote file name. If REMOTE = '=', then the file on the remote system will have the same name as on the local system.
CHOPT	Character variable to specify the options desired.
IRC	Integer variable in which the return code is returned.

This routine sends a text file to the remote system. If option 'S' is specified, statistics on the file transfer are printed. If option 'V' is specified, the remote file will have variable length record format (IBM-VM systems only).

Example of using the XZPUTA routine

```
      CALL XZPUTA('CZPACK.CARDS','=','S',IRC)
      IF(IRC.NE.0) PRINT *,'File transfer failed'
```

7.3.3 Get binary file: fixed length records

CALL **XZGETB** (LOCAL,REMOTE,LRECL,CHOPT,IRC)

LOCAL	Character variable specifying the local file name. If LOCAL = '=', then the file on the local system will have the same name as on the remote system.
REMOTE	Character variable specifying the remote file name.
LRECL	Integer variable specifying the record length of the file in bytes.
CHOPT	Character variable to specify the options desired.
IRC	Integer variable in which the return code is returned.

This routine gets a binary file from the remote system. The file must have fixed length records. ZEBRA FZ files in binary exchange format, PATCHY CETA files and EPIO files are examples of files that can be transferred with this routine. If option 'S' is specified, statistics on the file transfer are printed.

Example of using the XZGETB routine

```
      CALL XZGETB('FXFILE.DAT','FXFILE.VAX',32400,'S',IRC)
      IF(IRC.NE.0) PRINT *,'File transfer failed'
```

7.3.4 Send binary file: fixed length records

```
CALL XZPUTB (LOCAL,REMOTE,LRECL,CHOPT,IRC)
```

`LOCAL` Character variable specifying the local file name.

`REMOTE` Character variable specifying the remote file name. If REMOTE = '=', then the file on the remote system will have the same name as on the local system.

`LRECL` Integer variable specifying the record length of the file in bytes.

`CHOPT` Character variable to specify the options desired.

`IRC` Integer variable in which the return code is returned.

This routine sends a binary file to the remote system. The file must have fixed length records. ZEBRA FZ files in binary exchange format, PATCHY CETA files and EPIO files are examples of files that can be transferred with this routine. If option 'S' is specified, statistics on the file transfer are printed.

Example of using the XZPUTB routine

```
      CALL XZPUTB('CZPACK.CETA','=',3600,'S',IRC)
      IF(IRC.NE.0) PRINT *,'File transfer failed'
```

7.3.5 Get FORTRAN direct access file

```
CALL XZGETD (LOCAL,REMOTE,LRECL,CHOPT,IRC)
```

`LOCAL` Character variable specifying the local file name. If LOCAL = '=', then the file on the local system will have the same name as on the remote system.

`REMOTE` Character variable specifying the remote file name.

`LRECL` Integer variable specifying the record length of the file in bytes.

`CHOPT` Characer variable to specify the options desired.

`IRC` Integer variable in which the return code is returned.

This routine gets a direct access file from the remote system. ZEBRA FZ files in binary exchange format written with option D, ZEBRA RZ files (between like machines) are examples of files that can be transferred with this routine. If option 'S' is specified, statistics on the file transfer are printed.

Example of using the XZGETD routine

```
      CALL XZGETD('FXFILE.DAT','FXFILE.VAX',32400,'S',IRC)
      IF(IRC.NE.0) PRINT *,'File transfer failed'
```

7.3.6 Send FORTRAN direct access file

```
CALL XZPUTD (LOCAL,REMOTE,LRECL,CHOPT,IRC)
```

`LOCAL` Character variable specifying the local file name.

`REMOTE` Character variable specifying the remote file name. If REMOTE = '=', then the file on the remote system will have the same name as on the local system.

`LRECL` Integer variable specifying the record length of the file in bytes.

`CHOPT` Character variable to specify the options desired.

`IRC` Integer variable in which the return code is returned.

This routine sends a direct access file to the remote system. ZEBRA FZ files in binary exchange format written with option D, ZEBRA RZ files (between like machines) are examples of files that can be transferred with this routine. If option 'S' is specified, statistics on the file transfer are printed.

Example of using the XZPUTD routine

```
      CALL XZPUTD('FXFILE.DATA','=',32400,'S',IRC)
      IF(IRC.NE.0) PRINT *,'File transfer failed'
```

7.3.7 Get binary PAM file

```
CALL XZGETP (LOCAL,REMOTE,CHOPT,IRC)
```

`LOCAL` Character variable specifying the local file name. If LOCAL = '=', then the file on the local system will have the same name as on the remote system.

`REMOTE` Character variable specifying the remote file name.

`CHOPT` Character variable to specify the options desired.

`IRC` Integer variable in which the return code is returned.

This routine gets a binary PAM file from the remote system. If option 'S' is specified, statistics on the file transfer are printed.

Example of using the XZGETP routine

```
      CALL XZGETP('=','ZEBRA.PAM','S',IRC)
      IF(IRC.NE.0) PRINT *,'File transfer failed'
```

7.3.8 Send binary PAM file

CALL **XZPUTP** (LOCAL,REMOTE,CHOPT,IRC)

LOCAL Character variable specifying the local file name.

REMOTE Character variable specifying the remote file name. If REMOTE = '=', then the file on the remote system will have the same name as on the local system.

CHOPT Character variable to specify the options desired.

IRC Integer variable in which the return code is returned.

This routine sends a binary PAM file to the remote system. If option 'S' is specified, statistics on the file transfer are printed.

Example of using the XZPUTP routine

```
      CALL XZPUTP('KERNAPO.PAM','/cern/new/pam/kernapo/pam','S',IRC)
      IF(IRC.NE.0) PRINT *,'File transfer failed'
```

7.3.9 Get ZEBRA FZ file

CALL **XZGETF** (LOCAL,REMOTE,LRECL,LFORM,RRECL,RFORM,CHOPT,IRC)

LOCAL Character variable specifying the local file name. If LOCAL = '=', then the file on the local system will have the same name as on the remote system.

REMOTE Character variable specifying the remote file name.

LRECL Integer variable specifying the record length of the local file in bytes.

LFORM Character variable specifying the format of the local file.

RRECL Integer variable specifying the record length of the remote file in bytes.

RFORM Character variable specifying the format of the remote file.

CHOPT Character variable to specify the options desired.

IRC Integer variable in which the return code is returned.

This routine gets a binary file from the remote system. If the local file format or record length are not given, they default to the same values as on the remote system. The format may be 'A', for FZ exchange, ASCII mapping, 'X', for FZ exchange, binary, or ' ' for FZ native. For ASCII files the record length defaults to 80 bytes. For binary exchange format files, the record length is taken from the file itself. For native format files the record length must be specified. For binary exchange format files, a 'D' may also be specified, indicating that the file should be processed using direct-access I/O. If option 'S' is specified, statistics on the file transfer are printed.

Example of using the XZGETB routine

```
*
*     Transfer a remote ASCII exhange format file to a local
*     binary exchange format file
*
      CALL XZGETF('FXFILE.DAT','FXFILE.VAX',80,'A',32400,'X','S',IRC)
      IF(IRC.NE.0) PRINT *,'File transfer failed'
```

7.3.10 Send ZEBRA FZ file

```
CALL XZPUTF (LOCAL,REMOTE,LRECL,LFORM,RRECL,RFORM,CHOPT,IRC)
```

LOCAL	Character variable specifying the local file name.
REMOTE	Character variable specifying the remote file name. If REMOTE = '=', then the file on the remote system will have the same name as on the local system.
LRECL	Integer variable specifying the record length of the local file in bytes.
LFORM	Character variable specifying the format of the local file.
RRECL	Integer variable specifying the record length of the remote file in bytes.
RFORM	Character variable specifying the format of the remote file.
CHOPT	Character variable to specify the options desired.
IRC	Integer variable in which the return code is returned.

This routine sends a ZEBRA FZ file to the remote system. If the remote file format or record length are not given, they default to the same values as on the local system. The format may be 'A', for FZ exchange, ASCII mapping, 'X', for FZ exchange, binary, or ' ' for FZ native. For ASCII files the record length defaults to 80 bytes. For binary exchange format files, the record length is taken from the file itself. For native format files the record length must be specified. For binary exchange format files, a 'D' may also be specified, indicating that the file should be processed using direct-access I/O. If option 'S' is specified, statistics on the file transfer are printed.

Example of using the XZPUTF routine

```
*
*     Transfer the local exchange format file to a remote native
*     format file
*
      CALL XZPUTF('FZFILE.DATA','=',32400,'X',32400,' ','S',IRC)
      IF(IRC.NE.0) PRINT *,'File transfer failed'
```

7.3.11 Get RZ file

```
CALL XZGETR (LOCAL,REMOTE,CHOPT,IRC)
```

LOCAL	Character variable specifying the local file name. If LOCAL = '=', then the file on the local system will have the same name as on the remote system.
REMOTE	Character variable specifying the remote file name.
CHOPT	Character variable to specify the options desired.
IRC	Integer variable in which the return code is returned.

This routine gets a ZEBRA RZ file from the remote system. If option 'S' is specified, statistics on the file transfer are printed.

Example of using the XZGETR routine

```
      CALL XZGETR('=','HBOOK.CMZ','S',IRC)
      IF(IRC.NE.0) PRINT *,'File transfer failed'
```

7.3.12 Send ZEBRA RZ file

```
CALL XZPUTR (LOCAL,REMOTE,CHOPT,IRC)
```

`LOCAL` Character variable specifying the local file name.

`REMOTE` Character variable specifying the remote file name. If REMOTE = '=', then the file on the remote system will have the same name as on the local system.

`CHOPT` Character variable to specify the options desired.

`IRC` Integer variable in which the return code is returned.

This routine sends a ZEBRA RZ file to the remote system. ZEBRA RZ files include HBOOK histogram files, ntuples, CMZ files etc. If option 'S' is specified, statistics on the file transfer are printed.

Example of using the XZPUTR routine

```
      CALL XZPUTR('FPACK.CMZ','=,'S',IRC)
      IF(IRC.NE.0) PRINT *,'File transfer failed'
```

7.3.13 Get exchange format file

```
CALL XZGETX (LOCAL,REMOTE,LRECL,CHOPT,IRC)
```

`LOCAL` Character variable specifying the local file name. If LOCAL = '=', then the file on the local system will have the same name as on the remote system.

`REMOTE` Character variable specifying the remote file name.

`LRECL` Integer variable specifying the record length of the file in bytes.

`CHOPT` Characer variable to specify the options desired.

`IRC` Integer variable in which the return code is returned.

This routine retrieves an exchange format file from the remote system. An exchange format file is one with fixed length records and no control words. The `XZGETX` routine uses Fortran sequential I/O on all systems except Unix, where files are processed with direct access I/O to avoid the control words that are written at the beginning and end of each record with binary sequential Fortran I/O. If option 'S' is specified, statistics on the file transfer are printed.

Example of using the XZGETX routine

```
      CALL XZGETX('FXFILE.DAT','FXFILE.VAX',32400,'S',IRC)
      IF(IRC.NE.0) PRINT *,'File transfer failed'
```

7.3.14 Send exchange format file

```
CALL XZPUTX (LOCAL,REMOTE,LRECL,CHOPT,IRC)
```

`LOCAL` Character variable specifying the local file name.

`REMOTE` Character variable specifying the remote file name. If REMOTE = '=', then the file on the remote system will have the same name as on the local system.

`LRECL` Integer variable specifying the record length of the file in bytes.

`CHOPT` Character variable to specify the options desired.

`IRC` Integer variable in which the return code is returned.

This routine sends an exchange format file to the remote system. An exchange format file is one with fixed length records and no control words. The `XZPUTX` routine uses Fortran sequential I/O on all systems except Unix, where files are processed with direct access I/O to avoid the control words that are written at the beginning and end of each record with binary sequential Fortran I/O. If option 'S' is specified, statistics on the file transfer are printed.

Example of using the XZPUTX routine

```
      CALL XZPUTX('FXFILE.DATA','=',32400,'S',IRC)
      IF(IRC.NE.0) PRINT *,'File transfer failed'
```

7.4 Routines to perform remote I/O

7.4.1 Open remote file

```
CALL XZOPEN (LUN,FILE,NODE,LRECL,CHOPT,IRC)
```

`LUN` Integer variable specifying logical unit to be used.

`FILE` Character variable specifying the remote file name.

`NODE` Characer variable specifying the remote node name.

`LRECL` Integer ariable specifying the record length in bytes.

`CHOPT` Character variable to specify the options desired.

`IRC` Integer variable in which the return code is returned.

This routine opens a file on the specified node. If a connection to the remote system is not yet established, a call to CZOPEN is made automatically. The record length is currently only required for direct access files.

```
CHOPT: 'D' - Open the file for direct access (default=sequential)
CHOPT: 'F' - Open the file 'FORMATTED' (default=unformatted)
CHOPT: 'N' - Open the file with STATUS='NEW' (default=unknown)
```

Example of using the XZOPEN routine

```
      CALL XZOPEN(11,'/user/jamie/cspack/cspack.ceta',3600,' ',IRC)
      IF(IRC.NE.0) PRINT *,'Cannot open remote file'
```

7.4.2 Close remote file

```
CALL XZCLOS (LUN,CHOPT,IRC)
```

LUN	Integer variable specifying logical unit to be used.
CHOPT	Character variable to specify the options desired.
IRC	Integer variable in which the return code is returned.

This routine closes a remote file previously opened by XZOPEN.

CHOPT: 'D' - Delete remote file

Example of using the XZCLOS routine

```
      CALL XZCLOS(11,' ',IRC)
      IF(IRC.NE.0) PRINT *,'Error closing remote file'
```

7.4.3 Open a remote RZ file

```
CALL XZRZOP (LUN,NODE,CHFILE,CHOPT,LRECL,IRC)
```

LUN	Integer variable specifying logical unit to be used.
NODE	Characer variable specifying the remote node name.
CHFILE	Character variable specifying the remote file name.
CHOPT	Character variable to specify the options desired.
LRECL	Integer ariable specifying the record length in bytes.
IRC	Integer variable in which the return code is returned.

Use the XZROPN to open a remote RZ file. See the description of the RZOPEN routine in the Zebra manual for more details.

7.4.4 Read record from remote file

```
CALL XZREAD (LUN,IBUFF,NREC,NWANT,NGOT,CHOPT,IRC)
```

LUN	Integer variable specifying logical unit to be used.
IBUFF	Array to receive the data.
NREC	Integer variable specifying the record number to read (for direct access files only).
NWANT	Integer variable specifying the number of bytes to read (for files with variable length records NWANT specifies the maximum number of bytes that can be accepted).
NGOT	Integer variable specifying the number of bytes read for files with variable length records.
CHOPT	Character variable to specify the options desired.
IRC	Integer variable in which the return code is returned.

This routine reads a record from a remote file previously opened by XZOPEN.

Example of using the XZREAD routine

```
CALL XZREAD(11,IBUFF,0,32400,NGOT,' ',IRC)
IF(IRC.NE.0) PRINT *,'Error reading remote file'
```

7.4.5 Write record to remote file

`CALL XZRITE (LUN,IBUFF,NREC,NWRITE,CHOPT,IRC)`

LUN	Integer variable specifying logical unit to be used.
IBUFF	Array to containing the data to be written.
NREC	Integer variable specifying the record number to write (for direct access files only).
NWRITE	Integer variable specifying the number of bytes to write. for files with variable length records.
CHOPT	Character variable to specify the options desired.
IRC	Integer variable in which the return code is returned.

This routine writes a record from a remote file previously opened by XZOPEN.

Example of using the XZRITE routine

```
NREC    = 30
LENBUFF = 8192
CALL XZRITE(11,IBUFF,NREC,32400,LENBUFF,' ',IRC)
IF(IRC.NE.0) PRINT *,'Error writing to remote file'
```

7.4.6 Read a line from a remote file

`CALL XZGETL (LUN,CHLINE,CHFORM,CHOPT,IRC)`

LUN	Integer variable specifying logical unit to be used.
CHLINE	Character variable to receive the line
CHFORM	Character variable specifying the format to be used for reading the line
CHOPT	Character variable specifying the options required
IRC	Integer variable in which the return code is returned.

This routine reads a record from a remote formatted file previously opened with the XZOPEN routine.

Example of using the XZGETL routine

```
CALL XZGETL(LUFZFA,CHLINE,'(A)',' ',IRC)
IF(IRC.NE.0) GOTO 20
```

7.4.7 Write a line to a remote file

```
CALL XZPUTL (LUN,CHLINE,CHFORM,CHOPT,IRC)
```

LUN	Integer variable specifying logical unit to be used.
CHLINE	Character variable containing the data to be written
CHFORM	Character variable specifying the format to be used for writing the line
CHOPT	Character variable specifying the options required
IRC	Integer variable in which the return code is returned.

This routine writes a record to a remote formatted file previously opened with the `XZOPEN` routine.

Example of using the XZPUTL routine

```
      CALL XZPUTL(LUFZFA,CHLINE,'(A)',' ',IRC)
      IF(IRC.NE.0) GOTO 20
```

7.4.8 Rewind remote file

```
CALL XZREWD (LUN,CHOPT,IRC)
```

LUN	Integer variable specifying logical unit to be used.
CHOPT	Character variable to specify the options desired.
IRC	Integer variable in which the return code is returned.

This routine rewinds a remote file previously opened by XZOPEN.

Example of using the XZREWD routine

```
      CALL XZREWD(11,' ',IRC)
      IF(IRC.NE.0) PRINT *,'Error rewinding to remote file'
```

7.4.9 Inquire if remote file exists

```
CALL XZINQR (LUN,FILE,NODE,IEXIST,LRECL,IRC)
```

LUN	Integer variable specifying logical unit to be used.
FILE	Character variable specifying the remote file name.
NODE	Character variable specifying the node on which the file resides
IEXIST	Integer variable in which the remote file status is returned.
LRECL	Integer variable in which the record length of the remote file status is returned.
IRC	Integer variable in which the return code is returned.

This routine checks whether a remote file exists or is OPENed.

Example of using the XZINQR routine

```
      CALL XZINQR(11,'DISK$CERN:<JAMIE>ZEBRA.PAM',
     +   'VXCRNA',IEXIST,LRECL,IRC)
      IF(IRC.NE.0) PRINT *,'Error issuing remote inquire'
```

7.5 General utility routines

7.5.1 Initialise XZ package

```
CALL XZINIT (LPRINT,LDEBUG,LUNI,LUNO,IRC)
```

`LPRINT` Integer variable specifying logical unit to be used to print diagnostic messages.

`LDEBUG` Integer variable specifying the level of debug messages to be printed. See the description of the XZLOGL routine for details of the various log levels.

`LUNI` Integer variable specifying the logical unit used for file input by the XZGETx/XZPUTx routines.

`LUNO` Integer variable specifying the logical unit used for file output by the XZGETx/XZPUTx routines.

`IRC` Integer variable in which the return code is returned.

This routine sets the logical units to be used by the XZ package and the log level. The log level may be reset at any time by a call to XZLOGL or by a further call to XZINIT.

Example of using the XZINIT routine

```
      CALL XZINIT(6,0,11,21,IRC)
      IF(IRC.NE.0) PRINT *,'Error from XZINIT'
```

7.5.2 Set log level of XZ package

```
CALL XZLOGL (LDEBUG)
```

`LDEBUG` Integer variable specifying the level of debug messages to be printed.

This routine sets the log level of the XZ package The log level may be reset at any time by a further call to XZLOGL or by XZINIT.
The various levels are described below.

– -3 Suppress all log messages
– -2 Error messages
– -1 Terse logging
– 0 Normal
– 1 Log calls to XZ routines
– 2 Log to monitor XZ internals
– 3 Debug messages

Example of using the XZLOGL routine

```
      CALL XZLOGL(-3)
```

7.5.3 Print date of generation of package

CALL **XZVERS**

This routine prints the PAM file title from the CSPACK PAM file and the date and time of the PATCHY run that generated the code.

Example of using the XZVERS routine

```
      CALL XZVERS
```

7.6 Directory utilities

7.6.1 Change remote directory

CALL **XZCD** (PATH,IRC)

PATH Character variable specifying the name of the remote directory to be set.

IRC Integer variable in which the return code is returned.

This routine changes the remote directory to that specified by the character variable PATH. On VM systems, the remote directory should be given in the form user.address or ¡user.address¿ If the address is omitted, 191 is assumed.

Example of using the XZCD routine

```
      CALL XZCD('FAT3.192',IRC)
      IF(IRC.NE.0) PRINT *,'Error setting remote directory'
```

On remote VM systems, one can change directory to a mini-disk that has a read or write password by specifying the password and access mode required, as in the examples below.

```
*
*     Read link to FAT3.192
*
      CALL XZCD('FAT3.192 MYPASS R',IRC)
*
*     Write link to FAT3.192
*
      CALL XZCD('FAT3.192 MYPASS W',IRC)
```

7.6.2 Change local directory

CALL **XZLCD** (PATH,IRC)

PATH Character variable specifying the name of the local directory to be set.

IRC Integer variable in which the return code is returned.

This routine changes the local directory to that specified by the character variable PATH. On VM systems, the local directory should be given in the form user.address or ¡user.address¿ If the address is omitted, 191 is assumed.

Example of using the XZLCD routine

```
      CALL XZLCD('FAT3.192',IRC)
      IF(IRC.NE.0) PRINT *,'Error setting local directory'
```

7.6.3 Get current remote directory

```
CALL XZPWD (PATH,IRC)
```

PATH Character variable in which the current remote directory is returned.

IRC Integer variable in which the the return code is returned.

This routine returns the current remote directory.

Example of using the XZPWD routine

```
CALL XZPWD(PATH,IRC)
IF(IRC.NE.0) THEN
   PRINT *,'Error setting remote directory'
ELSE
   PRINT *,'Current working directory is ',PATH(1:LENOCC(PATH))
ENDIF
```

7.6.4 Get current local directory

```
CALL XZLPWD (PATH,IRC)
```

PATH Character variable in which the current local directory is returned.

IRC Integer variable in which the return code is returned.

This routine returns the current local directory.

Example of using the XZLPWD routine

```
CALL XZLPWD(PATH,IRC)
IF(IRC.NE.0) THEN
   PRINT *,'Error obtaining local directory'
ELSE
   PRINT *,'Current working directory is ',PATH(1:LENOCC(PATH))
ENDIF
```

7.6.5 Issue remote LS command

```
CALL XZLS (PATH,FILES,MAXFIL,NFILES,ICONT,CHOPT,IRC)
```

PATH Character variable specifying the path name for the remote ls command. If the intention is to list the current working directory, PATH should be set to a single blank.

FILES Character array of size MAXFIL in which the remote file names are returned. If more than MAXFIL files are found, IRC will be set to -1. XZLS may be called again with ICONT.NE.0 to receive the the next batch of file names. **N.B. no further communication with the remote node is possible until all pending file names have been read. Use the PATCHY sequence CZFLUSH to flush pending file names if required.**

MAXFIL	Integer constant specifying the dimension of the character array FILES.
NFILES	Integer variable in specifying the number of files returned in FILES.
ICONT	Integer variable specifying the number of files returned in FILES.
CHOPT	Character variable specifying the required options: If CHOPT = 'L' a 'long listing' will be returned. This corresponds to the Unix ls option -l and the VM LISTFILE option L.
IRC	Integer variable in which the return code is returned.

This routine issues a remote LS command and returns the output in the character array FILES.

Example of using the XZLS routine

```
      CALL XZLS('*.CARDS',FILES,100,NFILES,0,'L',IRC)
      IF(IRC.NE.0) THEN
         PRINT *,'Error issuing remote LS command'
      ELSE
         DO 10 I=1,NFILES
            PRINT *,FILES(I)(1:LENOCC(FILES(I)))
10       CONTINUE
      ENDIF
```

7.6.6 Issue local LS command

CALL **XZLLS** (PATH,FILES,MAXFIL,NFILES,ICONT,CHOPT,IRC)

PATH	Character variable specifying the path name for the local ls command. If the intention is to list the current working directory, PATH should be set to a single blank.
FILES	Character array of size MAXFIL in which the remote file names are returned. If more than MAXFIL files are found, IRC will be set to -1. XZLS may be called again with ICONT.NE.0 to receive the
MAXFIL	Integer constant specifying the dimension of the character array FILES.
NFILES	Integer variable in specifying the number of files returned in FILES.
ICONT	Integer variable specifying the number of files returned in FILES.
CHOPT	Character variable specifying the required options: If CHOPT = 'L' a 'long listing' will be returned. This corresponds to the Unix ls option -l and the VM LISTFILE option L.
IRC	Integer variable in which the return code is returned.

This routine issues a remote LS command and returns the output in the character array FILES.

Example of using the XZLLS routine

```
      CALL XZLLS('*.CARDS',FILES,100,NFILES,0,'L',IRC)
      IF(IRC.NE.0) THEN
         PRINT *,'Error issuing LS command'
      ELSE
         DO 10 I=1,NFILES
            PRINT *,FILES(I)(1:LENOCC(FILES(I)))
10       CONTINUE
      ENDIF
```

7.6.7 Issue remote MV command

```
CALL XZMV  (SOURCE,TARGET,CHOPT,IRC)
```

SOURCE Character variable specifying the source file name

TARGET Character variable specifying the target file name

CHOPT Options

- C Respect case of file names (Unix systems)

IRC Integer variable in which the return code is returned.

This routine moves the remote file from SOURCE to TARGET.

7.6.8 Issue local MV command

```
CALL XZLMV  (SOURCE,TARGET,CHOPT,IRC)
```

SOURCE Character variable specifying the source file name

TARGET Character variable specifying the target file name

CHOPT Options

- C Respect case of file names (Unix systems)

IRC Integer variable in which the return code is returned.

This routine moves the local file from SOURCE to TARGET.

7.6.9 Issue remote RM command

```
CALL XZRM  (FILE,IRC)
```

FILE Character variable specifying the name of the file to be removed.

IRC Integer variable in which the return code is returned.

This routine issues deletes the specified file on the remote system.

Example of using the XZRM routine

```
      CALL XZRM('CSPACK.CARDS',IRC)
      IF(IRC.NE.0)  PRINT *,'Error issuing RM command'
```

7.6.10 Issue local RM command

```
CALL XZLRM  (FILE,IRC)
```

FILE Character variable specifying the name of the file to be removed.

IRC Integer variable in which the return code is returned.

This routine issues deletes the specified file on the local system.

Example of using the XZLRM routine

```
      CALL XZLRM('CSPACK.CARDS',IRC)
      IF(IRC.NE.0)  PRINT *,'Error issuing RM command'
```

Chapter 8: TELNETG and TAG++

When using the standard TELNET program to login to a remote host, such as an IBM mainframe, from a local workstation, the graphics capabilities of the workstation are normally lost. TELNETG is a modified version of TELNET which overcomes this deficiency for HIGZ applications such as PAW or GEANT in a rather elegant manner. Not only is the user able to display graphical output from the remote session in a window on the local station, the mouse may also be used to provide input. More importantly, the HIGZ macro primitives are very compact, resulting in a significant reduction in network traffic (and corresponding increase in performance). Factors of 10 improvement are typical for one dimensional histograms, rising to 100 or more for two dimensional histograms, surfaces, LEGO plots etc.) The only change that the user must make (apart from typing TELNETG instead of TELNET, is to specify the negative value of the workstation type in the remote application. Thus, when using TELNETG from an Apollo DN3000 to run PAW on CERNVM, the workstation type -10002 should be used.

TAG++ is a terminal emulator that provides full-screen access to IBM VM systems. The version included in CSPACK has been enhanced to provide the same kind of graphics support as in TELNETG. As with TELNETG, HIGZ applications, such as PAW, may display graphical output in a local window and receive graphical input, e.g. using the mouse.

Chapter 9: SYSREQ and SYSREQ-TCP

On VM/CMS systems, two versions of SYSREQ exist. The first requires a CP modification to add a new command plus a diagnose (Diagnose 140). The second version uses `IUCV` and is enabled by selecting `IUCVREQ` when installing the package via the `PATCHY` [5] command `+USE,IUCVREQ`.

SYSREQ-TCP provides a remote interface to a central SYSREQ server over TCP/IP connections. SYSREQ-TCP is currently only used to provide remote access to the HEPVM Tape Management System (TMS) from nodes at CERN other than CERNVM, where the TMS currently resides. However, the mechanism of passing commands and messages to a server that is already running is of general use and so it is planned to release this code as a separate component that avoids all use of SYSREQ on the IBM system.

Both command line and FORTRAN callable interfaces to SYSREQ exist. The command line interface is shown below.

Using the SYSREQ command line interface

```
SYSREQ service command

e.g.

SYSREQ TMS QVOL I29021
```

9.1 The SYSREQ FORTRAN interface

`CALL` **SYSREQ** `(SERVICE,COMMAND,IRC*,REPLY*,*LENREP*)`

`SERVICE`	Character variable specifying the service required
`COMMAND`	Character variable specifying the command to pass to that service
`IRC`	Integer variable in which the return code is returned
`REPLY`	Character array of length LENREP in which the reply is returned
`LENREP`	Integer variable containing the number of elements of REPLY on input and the number of elements of REPLY containing returned data on output

This routine sends the specified command to the named service via the SYSREQ mechanism. One may also use the routine FMSREQ, which is part of the FATMEN [3] and resides in PACKLIB. This routine as the same calling sequence as SYSREQ, but provides automatic protection against network problems (timeouts etc.) with retry where required.

`IRC` Return status

- `0` Normal completion
- `2` Reply longer then LENREP. The COMMAND(LENREP) contains the command to issue to get the remaining part of the reply.

Example of using the SYSREQ routine

```
      CHARACTER*240 COMMAND
      CHARACTER*8   SERVICE
      INTEGER       IRC
      INTEGER       REPLEN
      PARAMETER     (REPLEN=100)
      CHARACTER*132 TMSREP(REPLEN)

      IRC = 0

      SERVICE = 'TMS'
      COMMAND = 'Q VID I29001 - I29010'
      LCOMM   = LENOCC(COMMAND)

 500  CONTINUE
      I = REPLEN
      CALL SYSREQ(SERVICE,COMMAND(1:LCOMM),IRC,TMSREP,I)

      DO 20 J=1,I-1
      WRITE (6,200) TMSREP(J)
 200  FORMAT(1X,A80)
 20   CONTINUE

      IF (IRC .EQ. 2) THEN
*
*     Reply exceeded buffer length. Print command that we
*     should issue to get remainder of reply
*
         COMMAND = TMSREP(I)
         LCOMM   = LENOCC(COMMAND)
         PRINT *,'Issuing ',COMMAND(1:LCOMM)
         GOTO 500
      ENDIF
C       Print the Last Line
      WRITE (6,200) TMSREP(I)

 9999 CONTINUE
      PRINT *,'SYSREQ(Fortran): RC(',IRC,')'

      END
```

Chapter 10: The ZEBRA and PAW servers

The ZEBRA and PAW servers (ZSERV, PAWSERV) are all built as part of the standard program library installation. More details can be found in the Installation and Management section of this manual.

The following server routines are all controlled by a single server steering routine. This routine receives messages from the client, unpacks the messages and calls the appropriate server routine with a standard FORTRAN call.

The remote file transfer routines behave similarly. However, rather than just issue remote reads or writes record by record, the individual records of the files to be transferred are blocked to reduce the number of network operations. This has a significant effect on the file transfer rate.

Error and informational messages from the server are sent back to the client using the CZPUTA routine. These are processed in a standard manner using the PATCHY sequence CZMESS.

10.1 Server Routines to perform remote I/O

10.1.1 Open remote file

`CALL` **SZOPEN** `(LUN,FILE,LRECL,CHOPT,IRC)`

`LUN`	Integer variable specifying logical unit to be used.
`FILE`	Character variable specifying the file name.
`LRECL`	Integer variable specifying the record length in bytes.
`CHOPT`	Character variable to specify the options desired.
`IRC`	Integer variable in which the return code is returned.

This routine opens a file on the server node.

```
CHOPT: 'D' - Open the file for direct access
CHOPT: 'F' - Open the file 'FORMATTED' (default=unformatted)
CHOPT: 'N' - Open the file with STATUS='NEW'
```

10.1.2 Close remote file

`CALL` **SZCLOS** `(LUN,CHOPT,IRC)`

`LUN`	Integer variable specifying logical unit to be used.
`CHOPT`	Character variable to specify the options desired.
`IRC`	Integer variable in which the return code is returned.

This routine closes a remote file previously opened by SZOPEN.

```
CHOPT: 'D' - Delete remote file
```

10.1.3 Read record from remote file

```
CALL SZREAD (LUN,IBUFF,NREC,NWANT,NGOT,CHOPT,IRC)
```

LUN	Integer variable specifying logical unit to be used.
IBUFF	Array to receive the data.
NREC	Integer variable specifying the record number to read (for direct access files only).
NWANT	Integer variable specifying the number of bytes to read (for files with variable length records NWANT specifies the maximum number of bytes that can be accepted).
NGOT	Integer variable specifying the number of bytes read for files with variable length records.
CHOPT	Character variable to specify the options desired.
IRC	Integer variable in which the return code is returned.

This routine reads a record from a remote file previously opened by SZOPEN.

10.1.4 Write record to remote file

```
CALL SZRITE (LUN,IBUFF,NREC,NWRITE,CHOPT,IRC)
```

LUN	Integer variable specifying logical unit to be used.
IBUFF	Array to containing the data to be written.
NREC	Integer variable specifying the record number to write (for direct access files only).
NWRITE	Integer variable specifying the number of bytes to write. for files with variable length records.
CHOPT	Character variable to specify the options desired.
IRC	Integer variable in which the return code is returned.

This routine writes a record from a remote file previously opened by SZOPEN.

10.1.5 Rewind remote file

```
CALL SZREWD (LUN,CHOPT,IRC)
```

LUN	Integer variable specifying logical unit to be used.
CHOPT	Character variable to specify the options desired.
IRC	Integer variable in which the return code is returned.

This routine rewinds a remote file previously opened by SZOPEN.

10.1.6 Inquire if remote file exists

```
CALL SZINQR (LUN,CHOPT,IRC)
```

LUN	Integer variable specifying logical unit to be used.
FILE	Character variable specifying the remote file name.
NODE	Character variable specifying the remote node name.
IEXIST	Integer variable in which the remote file status is returned.
LRECL	Integer variable in which the record length of the remote file status is returned.
IRC	Integer variable in which the return code is returned.

This routine checks whether a remote file exists or is OPENed.

10.2 General utility routines

10.2.1 Print date of generation of package

```
CALL SZVERS
```

This routine prints the PAM file title from the CSPACK PAM file and the date and time of the PATCHY run that generated the code.

10.3 Remote directory utilities

10.3.1 Change remote directory

```
CALL SZCD (PATH,IRC)
```

PATH	Character variable specifying the name of the remote directory to be set.
IRC	Integer variable in which the return code is returned.

This routine changes the remote directory to that specified by the character variable PATH. On VM systems, the remote directory should be given in the form user.address or ¡user.address¿ If the address is omitted, 191 is assumed.

10.3.2 Get current remote directory

```
CALL SZPWD (PATH,IRC)
```

PATH	Character variable in which the current remote directory is returned.
IRC	Integer variable in which the return code is returned.

This routine returns the current remote directory.

10.3.3 Issue remote LS command

```
CALL SZLS (PATH,CHOPT,IRC)
```

`PATH` Character variable specifying the path name for the remote ls command. If the intention is to list the current working directory, PATH should be set to a single blank.

`CHOPT` Character variable specifying the required options: If CHOPT = 'L' a 'long listing' will be returned (Unix and VM systems). This corresponds to the Unix ls option -l and the VM LISTFILE option L.

`IRC` Integer variable in which the return code is returned.

This routine issues a remote LS command and returns the output to the client.

Chapter 11: Format of the netrc and ftplogin files

On Unix and VM/CMS systems, the `.netrc` `DOT NETRC A0` on VM/CMS systems) have the following format. On Unix systems these files must reside in the home directory of the relevant user and be correctly protected using the command

Protecting a .netrc file

```
chmod 0600 .netrc
```

Format of the .netrc files

```
machine <host-name> login <user-name> password <password>

e.g.

machine cernvm login zftptest password kwerdal
```

On VAX/VMS systems, the file is named `ftplogin.;` and should again reside in the home directory. It should be protected as follows:

Protecting an ftplogin file

```
SET FILE/PROTECTION=(S,W,G,O:R) FTPLOGIN.;
```

The format of this file is somewhat simpler, containing no keywords, as shown in the following example.

An example of an ftplogin file

```
cernvnm zftptest kwerdal

vxcrna  zftptest -
```

Note that the minus sign will cause a prompt for the password.

Part IV

CSPACK – Installation and Management Guide

Chapter 12: Availability of CSPACK at CERN

The CSPACK PAM file is available on all central CERN systems in the normal place, e.g. on the CERN-PAMS disk on CERNVM, in /cern/pro/pam on Unix systems and in CERN:¡PRO.PAM¿ on VAX/VMS.

The CSPACK FORTRAN callable interface routines are installed in PACKLIB on all systems which have PACKLIB corresponding to CERN Computer News Letter 200 or above. The tools (ZFTP, ZSERV, PAWSERV, SYSREQ, TELNETG) are installed on all central systems with the following exceptions:

1 TELNETG is available on Unix and VAX/VMS systems only

2 SYSREQ requires Wollongong or Multinet TCP/IP (VAX/VMS systems). Note that SYSREQ is currently only used (in the context of CSPACK) for remote access to the CERN TMS system. On VM/CMS systems, IUCREQ (SRQSRV) should be used.

Chapter 13: Installing and using the CSPACK package

All of the routines and programs that compose CSPACK are installed used the standard CERNLIB installation tools. The general procedure is:

1. Generate PACKLIB (e.g. MAKEPACK -l PACKLIB)
2. Generate the tools (e.g. MAKEPACK ZFTP, MAKEPACK ZSERV etc.)
3. Configure the system files

More information on the CERNLIB installation procedures can be found on the INSTALL PAM in the deck UGUIDE for the relevant machine. For example, on Unix systems, the deck is in PATCH=DUNIX,D=UGUIDE, for VAX systems, the deck is in PATCH=DVAX,D=UGUIDE etc.

The link procedure for the generation of user-developed client-server programs should be based on that of ZFTP and ZSERV for the relevant machines. In all cases, the CERNLIB installation procedure should be followed.

13.1 Configuration for use with TCPAW

To use the CSPACK tools with TCPAW as the network layer, files on both the client and server side must be correctly configured. Firstly, the TCP port numbers and associated services must be defined. Secondly, in the case that incoming connections are allowed (and possible), the programs to be run for each known service must be defined. The following table lists the current values and names used at CERN. Note that these definitions must be made on both client and server and must match. On Unix these definitions are made by adding a line to the file `/etc/services` as follows:

```
pawserv  345/tcp   # Comment string, e.g. 'For distributed PAW'
```

For VAX/VMS and other systems, see the relevant system specific details.

Table 13.1: Service names and TCP ports used by CSPACK

Service	TCP port	Description
pawserv	`345`	Distributed PAW
zserv	`346`	Server for ZFTP, remote Zebra
fatserv	`347`	Server for FATMEN

These ports have been registered with the Internet Assigned Number Authority at ISI.

To permit incoming connections, a definition in the relevant services file must be made. This is typically in */etc/inetd.conf* for Unix systems, although in the case of the Silicon Graphics the file is in */usr/etc/inetd.conf*. The information to be added to this file consists of one line per service, specifying the service name and program to be run, e.g.

```
zserv stream tcp nowait root /cern/pro/bin/zserv zserv
```

13.1.1 Unix specific details

After modifying the `/etc/inetd.conf` file, the inetd must be told to re-read the file. This can be done by rebooting the system, or by sending a hangup signal.

Table 13.2: Signalling inetd to reread the /etc/inetd.conf file

System	Command
AIX	**refresh -s inetd**
HP/UX	**inetd -l**
Alpha/OSF	**/sbin/init.d/inetd stop—start**
Others	**kill -1 pid**

AIX

On systems running AIX, it is recommended that the following line be added to the file `/etc/environment`. This sets the shell variable `xrf_messages` to `no` and thus prevents FORTRAN run-time messages being printed.

```
xrf_messages=no
```

These messages disturb the protocol used between the ZSERV or PAWSERV servers and the ZFTP or PAW clients.

13.1.2 Systems running ”yellow pages” (NIS)

On systems running **yellow pages**, use the ypmake command to update the NIS database. For more information, consult the *man* pages on your host, or see your system administrator.

Updating NIS database

```
ypmake services
```

13.1.3 VAX/VMS specific details

N.B. ZSERV and PAWSERV are linked against ’the system’ (e.g. LINK/SYSEXE on AXP, sys$system:sys.stb on VAX) and are thus likely to be INCOMPATIBLE across VMS releases. If you experience problems, check that there is not a version mismatch, as shown below. If there is, you MUST rebuild the appropriate server on your system.

Checking for mismatches in the system level

```
r cern:[pro.exe]zserv
%DCL-W-ACTIMAGE, error activating image CERN:[PRO.EXE]ZSERV
-CLI-E-IMGNAME, image file $1$DUA3:[CERN.][PRO.EXE]ZSERV.EXE;35
-SYSTEM-W-SYSVERDIF, system version mismatch - please relink
```

On VAX/VMS systems, three versions of TCP/IP are currently supported. The CERNLIB installation procedure automatically performs the correct link procedure but the configuration of the system files must be performed manually.

The VAX/VMS version of ZFTP and ZSERV also support connections via DECnet from other VAX/VMS systems. This is activated by the -d option, e.g.

```
$ZFTP VXCRNA -D
```

DECnet

No configuration is required for ZFTP. However, ZSERV must be defined as a known DECnet object, e.g.

```
MCR NCP
NCP>SET OBJECT ZSERV NUMBER 0 FILE CERN:[PRO.EXE]ZSERV
NCP>DEF OBJECT ZSERV NUMBER 0 FILE CERN:[PRO.EXE]ZSERV
NCP>EXIT
```

The ZSERV program automatically detects whether the incoming request is via DECnet or TCP/IP and acts accordingly.

DEC UCX version 3.0

To define the service PAWSERV for UCX version 3.0, issue the following command:

Defining PAWSERV for UCX 3.0

```
$ UCX SET SERVICE PAWSERV /PORT=345
                          /FILE=CERN:[PRO.EXE]UCX$PAWSERV_STARTUP
                          /PROCESS_NAME=PAWSERV
                          /USER=SYSTEM

$ UCX ENABLE SERVICE PAWSERV

The default protocol is /PROTOCOL=TCP.
```

ZSERV is defined similarly (port 346).

The command procedure UCX$PAWSERV˙STARTUP looks like:

UCX$PAWSERV˙STARTUP command file

```
$ RUN CERN:[PRO.EXE]PAWSERV
```

DEC UCX prior to version 3.0

Versions of the DEC UCX product prior to 3.0 do not provide an Internet Daemon (inetd), hence incoming connections are not possible. Thus there is no equivalent to the `/etc/inetd.conf` file.

Furthermore, the library routine GETSERVBYNAME returns -1, indicating 'function not implemented'. Until this function is included in the UCX library, the routine GETSERVBYNAME from PATCH TCPAW in the CSPACK pam file may be used. This code is activated by selecting +USE,UCX in the PATCHY step of the installation procedure. An example configuration file for use with this routine is available in P=CONFIG,D=SERVICES.

Wollongong

In the case of Wollong the equivalent of `/etc/services` is `TWG$TCP:<NETDIST.ETC>SERVICES`. The file format is the same for Unix systems, thus an entry such as

```
zserv 346/tcp
```

should be made.

The equivalent to the */etc/inetd.conf* is the file `TWG$TCP:<NETDIST.ETC>SERVERS.DAT`. An example entry is shown below.

```
service-name    Pawserv
program         CERN:<PRO.EXE>PAWSERV.EXE
socket-type SOCK_STREAM
socket-options SO_ACCEPTCONN | SO_KEEPALIVE
socket-address  AF_INET , 345
working-set     300
INIT TCP_Init
LISTEN TCP_Listen
CONNECTED TCP_Connected
SERVICE         Run_Program
```

Multinet

On systems running MULTINET TCP/IP, the equivalent file to `/etc/services` is (somewhat confusingly) MULTINET:HOSTS.LOCAL. Entries should be added to this file in the format

```
SERVICE : TCP : 345 : PAWSERV :
SERVICE : TCP : 346 : ZSERV :
```

After making changes to this file, it should be compiled using the command

```
MULTINET HOST_TABLE COMPILE
```

To activate these changes without restarting the system, type

```
@MULTINET:INSTALL_DATABASES

@MULTINET:START_SERVER
```

To define servers to Multinet, use the command

```
MULTINET CONFIGURE /SERVERS
```

An example dialogue is given below:

```
SERVER-CONFIG>add zserv
Protocol: [TCP] tcp
TCP Port number: 346
Program to run: CERN:[PRO.EXE]ZSERV.EXE
SERVER-CONFIG>RESTART
```

More details on configuration MULTINET may be found in the Multinet System Administrator's Guide, including how to restrict access to certain services etc.

13.1.4 VM/CMS specific details

On VM/CMS systems, two versions of TCPAW exist. The recommended version is the same as used on other systems, but requires the IBM SAA C compiler and IBM's TCP/IP version 2 or higher. When using this version, which is activated by selecting the PATCHY flag TCPSOCK (performed by default in the CERN Program Library installation cradles), the file ETC SERVICES, which resides on the TCP/IP installation disk, must be modified to include definitions of the required services (ZSERV, PAWSERV) as for Unix systems.

If TCPSOCK is de-selected, e.g. +USE,TCPSOCK,T=INHIBIT, then the older PASCAL version of TCPAW is activated. This version has the definitions of zserv and pawserv hard-coded (to ports 346 and 345 respectively).

There are some limitations with the PASCAL version, the most significant of which is the fact that connections between two VM systems is not currently possible.

Other limitations that currently exist for VM/CMS systems include:

1 Servers are started using a different technique to other systems. It is for this reason that the remote system must know that it is talking to a VM system. Use the option -V on the open command, e.g. ZFTP vmnode -V in such cases.

2 Due to limitations of VM/CMS, the username specified when starting a server on VM systems must not be currently in use ('logged on').

Appendix A: CSPACK overview

Table A.1: ZFTP commands

Command	Function	Description
OPEN	Open connection	Establish connection to specified host
CLOSE	Close connection	Close connection with current host
GETA/PUTA	Text file transfer	Text file transfer, e.g. scripts, EXECs, CARD pams etc.
GETB/PUTB	Binary file transfer	Binary file transfer (fixed length records only) e.g. ZEBRA FZ binary exchange format, EPIO, CETA files
GETD/PUTD	Direct-access file transfer	Direct-access file transfer e.g. ZEBRA RZ file between like machines.
GETRZ/PUTRZ	ZEBRA RZ file transfer	RZ file transfer with automatic conversion between different data representations, e.g. HBOOK histogram or ntuple files, CMZ files.
GETFZ/PUTFZ	ZEBRA FZ file transfer	FZ file transfer with automatic conversion between different data representations (currently in preparation)
GETP/PUTP	Compact binary PAM transfer	Transfer a compact binary PAM file (not yet to Cray)
CD	Change working directory	Set working directory on remote node
LCD	Change working directory	Set working directory on local node
LS	Remote LS command	Make remote directory listing
LLS	Local LS command	Make local directory listing
MPUT	Put multiple files	Send all files matching the specified pattern to the remote machine. The mode of transfer is determined by the file type: `.CET, .CETA = PUTB, .CMZ = PUTRZ, other = PUTA`
MGET	Get multiple files	Retrieve all files matching the specified pattern from the remote machine. The mode of transfer is determined by the file type: `.CET, .CETA = GETB, .CMZ = GETRZ, other = GETA`
MV	Move (rename) remote file	
PWD	Print remote directory	Display current remote directory
LPWD	Print local directory	Display current local directory
RM	Remove remote file	Remote file deletion
LRM	Remove local file	Local file deletion
RSH	Remote shell	Issue command to the remote shell
VERSION	Version of ZFTP	Print version of the ZFTP program

Table A.1: ZFTP commands (continued)

Command	Function	Description
`SVERSION`	Version of server	Print version of the remote server (if connected)

Table A.2: CSPACK routine calling sequences

CZ routines	
Description `CALLING SEQUENCE`	Page
Open communication with a remote node CZOPEN	35
Close communication with the current remote node CZCLOS	36
Swith communication to another node CZSWAP	36
Return real time elapsed since last call CZRTIM	36
Send text string to current remote node CZPUTA	37
Read text string from remote server CZGETA	37
Send character array to remote server process CZPUTC	38
Get character array from remote server process CZGETC	38
Transfer data between client and server CZTCP	39
XZ routines	
Description **CALLING SEQUENCE**	**Page**
Send text file XZPUTA	44
Get text file XZGETA	43
Send binary file XZPUTB	45
Get binary file: fixed length records XZGETB	44

Table A.2: CSPACK routines (continued)

Description CALLING SEQUENCE	Page
Send binary file XZPUTB	45
Get D/A file XZGETD	44
Send D/A file XZPUTD	47
Get binary PAM file XZGETP	46
Send Zebra FZ file XZPUTF	48
Get FZ file XZGETF	47
Send Zebra RZ file XZPUTR	49
Get RZ file XZGETR	48
Send PAM file XZPUTP	47
Get PAM file XZGETP	46
Open remote file XZOPEN	50
Close remote file XZCLOS	51
Read record from remote file XZREAD	51
Write record to remote file XZRITE	52
Rewind remote file XZREWD	53
Inquire if remote file exists XZINQR	53
Initialise XZ package XZINIT	54
Set log levelXZ package XZLOGL	54

Table A.2: CSPACK routines (continued)

Description CALLING SEQUENCE	Page
Change remote directory	
XZCD	55
Get current remote directory	
XZPWD	56
Issue remote LS command	
XZLS	56
Remove remote file	
XZRM	58
Change local directory	
XZLCD	55
Get current local directory	
XZLPWD	56
Issue local LS command	
XZLLS	57
Remove local file	
XZRM	58

Bibliography

[1] L. Lamport. *LaTeX A Document Preparation System (2nd Edition)*. Addison-Wesley, 1994.

[2] Adobe. *PostScript Language Manual (Second Edition)*. Addison Wesley, 1990.

[3] J. D. Shiers. *FATMEN - Distributed File and Tape Management, nProgram Library Q123*. CERN, 1992.

[4] M. Brun, R. Brun, and F. Rademakers. *CMZ - A Source Code Management System*. CodeME S.A.R.L., 1991.

[5] H. J. Klein and J. Zoll. *PATCHY Reference Manual, nProgram Library L400*. CERN, 1988.

[6] CN/ASD Group. *HBOOK Users Guide (Version 4.21), nProgram Library Y250*. CERN, January 1994.

[7] CN/ASD Group. *PAW users guide, nProgram Library Q121*. CERN, October 1993.

[8] CN/ASD Group. *KUIP – Kit for a User Interface Package, nProgram library I202*. CERN, January 1994.

[9] CN/ASD Group and J. Zoll/ECP. *ZEBRA Users Guide, nProgram Library Q100*. CERN, 1993.

[10] C. Curran et al. The FATMEN Report. Technical Report DD/89/15, CERN, 1989.

[11] A. Cass. *CERN Tape Management System Users Guide nTo be published*. CERN, 1992.

[12] D. O. Williams et al. The Muscle Report. Technical Report DD/88/1, CERN, 1989.

[13] J. J. Thresher et al. *Computing at CERN in the 1990s*. CERN, 1989.

Index