NASA Technical Memorandum 107564 (Revised)

IN-60-
160508
R50

CONVEX

Mini Manual
February 1993

Central Scientific Computing Complex
Document CX-1f

NASA

National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23681-0001

NASA Langley Research Center

Central Scientific
Computing Complex
Document CX-1f

CONVEX
Mini Manual

February 1993

(Replaces CX-1e Dated February 1992)

Geoffrey M. Tennille and Lona M. Howser

PREFACE

This record of revision is a permanent part of the *CONVEX Mini Manual*. In this release, some section headings have the suffix [f] added to indicate that the section was added or substantially changed at this revision. Section headings without a suffix or with suffixes [a] through [e] have not changed. This method of annotation was chosen since sections in the Mini Manual are generally brief, thus a revision record can be kept with the text of the document. Grammatical, spelling and other changes that don't effect the context of the section are not marked.

DATE	REVISION	MAJOR FEATURES
November 1988	Original	Typographical corrections to Preliminary version
March 1989	Revision a	CX batch; NOS TCP/IP Communications; Mass Storage Utilities; Use of laser printer
July 1989	Revision b	Changes to MSS commands and CXbatch; CONVEX OS 7.1; Implementation of all mathematical and graphics libraries described in previous revision
April 1990	Revision c	CONVEX OS 8.0; CXbatch queues; Use of psjet+ laser postscript printer; Ada on Eagle; Changes to MSS utilities and their full implementation on the CONVEX computers; List of acronyms used; Section guides to hidden files and commands used in this manual; Appendix on sponsoring unsupported software; fcl utility to generate FORTRAN source listings
February 1991	Revision d	Reorganization of Mini Manual; CONVEX OS 7.1
February 1992	Revision e	CONVEX OS 9.1; Hardware upgrades to Mustang & Eagle; CONVEX Performance Analyzer (CXpa); OSF/MOTIF; Release 6.1 of the FORTRAN compiler; Reconfiguration of and changes in access to Mustang & Eagle; Mathematica; Removal of support for Ada
February 1993	Revision f	CXdb debugger; Application compiler

ACRONYMS [f]

ACD	Analysis and Computation Division
ANSI	American National Standards Institute
ARC	Ames Research Center
BLAS	Basic Linear Algebra Subroutines
CAB	Computer Applications Branch
CGL	Common Graphics Library
CMB	Computer Management Branch
CPU	Central Processing Unit
DCM	Division Computing Manager
EARS	Explicit Archival and Retrieval System
FTP	File Transfer Protocol
IBM	International Business Machines
IMSL	International Mathematical Statistical Library
I/O	Input/Output
ISO	International Organization for Standardization
LaRC	Langley Research Center
LaRCGOS	LaRC Graphics Output System
LaTS	LaRC Telecommunications System
LCUC	LaRC Computer Users Committee
MFLOPS	Million FLoating Operations Per Second
MOTD	Message-of-the-Day
MSS	Mass Storage Subsystem
NAS	Numerical Aerodynamic Simulator
NCAR	National Center for Atmospheric Research
NFS	Network File System
NCS	NOS Computing Subsystem
NOS	Network Operating System
NQS	Network Queuing System
OCO	Operations Control Office
PVI	Precision Visuals, Inc.
RGL	Remote Graphics Library
RM/RMT	Raster Metafile/RM Translator
SAG	Supercomputing Applications Group
SNS	Supercomputing Network Subsystem
SSIG	Supercomputing Special Interest Group
TCP/IP	Transmission Control Protocol/Internet Protocol
VPS	Vector Processing Subsystem
X, X11	X Window System

Table of Contents

1. INTRODUCTION [f]

The **Supercomputing Network Subsystem** (SNS) consists of two computers from CONVEX, a CRAY-2S/4128 and a CRAY Y-MP8E/8256 (see CR-1). Much documentation is on-line, so only a limited supply of printed documentation is available from OCO. If you are an experienced UNIX user, it is recommended that you peruse this manual to check for topics that are unique to CONVEX before using the machines. The novice UNIX user should read the *SNS Programming Environment User's Guide* (A-8) prior to beginning.

1.1. The CONVEX Mini Manual [f]

This revision of the CONVEX Mini Manual reflects the installation of the CONVEX Application Compiler and the X Window System based debugger CXdb. Additionally, some minor typographical changes have been made.

As with other mini manuals, this one does not attempt to provide a tutorial on any given topic. It is intended as a broad overview of the system, with references to more detailed information. Commands described in this manual usually have several other options for which space does not allow a discussion. Examples used are for the C-shell unless specifically designated as Bourne shell examples. Use of the *man* pages, a standard UNIX feature, is recommended to learn more about any particular command. The *notes* utility is installed on the CONVEX computers (see section 8.3 of A-8). It is used to disseminate information rapidly to SNS users.

1.2. Characteristics of the CONVEX Supercomputers [e]

SNS includes two computers from CONVEX Computer Corporation. There is a CONVEX C210 named **Eagle** and a CONVEX C220 named **Mustang.** The C220 differs from the C210 in that it has 2 CPU's rather than 1. Hereafter, both the C210 and C220 will be referred to as C2's in the text. **Eagle** and **Mustang** are user application computers envisioned to be used to support applications on the CRAY supercomputers, **Voyager** and **Sabre** through program development and checkout, debugging, and pre- and post-processing. The following subsections give a brief description of the C2's.

1.2.1. Hardware [e]

The C2 is a register to register vector processing computer with a virtual memory architecture. **Eagle** has 256 Mbytes (32 million 64-bit or 64 million 32-bit words) of central memory, while **Mustang** has 512 Mbytes (64 million 64-bit or 128 million 32-bit words) of central memory. The central memory of both machines is interleaved in 16 memory banks with a virtual address space of 4 Gbytes. Pages are 4096 bytes each.

The CPU is similar in many ways to those of the CRAY supercomputers, **Voyager** and **Sabre,** which are also parts of **SNS.** The C2 includes 8 address, 8 scalar, and 8 vector registers. Each vector register holds up to 128 64-bit words. Vectors are transferred to and from the memory through a single port between the registers and the memory. There are two Vector Processing Unit Boards which process alternate elements from vector operands to provide a result per minor cycle when doing either a vector add/subtract (from the add / logical functional unit) or a vector multiply (from the multiply / divide / square root functional unit). It is possible for both functional units to operate concurrently on independent operands or for chaining to occur between them. There is also a scalar arithmetic unit to do non-vector arithmetic and logical operations. That unit can run concurrently with the vector unit when no conflict exists.

Eagle is configured with 9 DKD-308 disk drives (about 8.5 Gbytes of storage) and **Mustang** is configured with 2 DKD-308 disk drives (about 2 Gbytes of storage) plus 3 DKD-314 removable disk drives.

1.2.1.1. Tape Drives [e]

There are two nine-track 6250 bpi tape drives associated with each of the C2's. There is no automatic mounting of tapes in the batch environment. Users need to physically bring their tapes to the Operations Control Office (room 1047 in building 1268). The UNIX utilities *tar* and *cpio* are used to write tapes. Once a tape is mounted by an operator, the *tpmount* and *tpunmount* utilities must be used to access the tape drive from your interactive session and to free the drive for use by others.

1.2.2. Software [e]

CONVEX's UNIX-based operating system is an enhanced version of the University of California Berkeley UNIX 4.2 operating system that supports a demand paging virtual memory. The current version (as of September 1991) on the C2's is CONVEX UNIX Version 9.1. It supports all of the traditional features of UNIX, some of which are described in sections 2.2.5, 2.2.7 and 3 of A-8. Batch use of the C2's is implemented with the CXbatch utility (see section 4.3.1). The FORTRAN compiler, described in section 2.4.1, is standard FORTRAN-77 with some extensions. The compiler performs automatic vectorization on user source codes. There is no explicit vector syntax. Also available on the C2's are an ANSI C compiler (see section 3.1), several mathematics libraries (see chapter 7 of A-8), and debugging utilities (see chapter 5).

1.2.3. Performance [a]

The 40 nanosecond minor cycle CPU can provide one result per cycle from each of the two functional units in vector mode. This translates into a peak speed of 25 MFLOPS (Million of FLoating point OPerations per Second) per unit. This rate can be doubled when both units are simultaneously involved in doing either independent vector operations or chaining. The C2 also supports 32 bit arithmetic operations but there is no significant difference in speed between the two modes.

Benchmark results indicate that a single CPU of a C2 has about 35% of the computing power of the now-removed **VPS-32** when comparing kernel calculations which contain a mix of long and short vectors, nonunit stride vectors and some scalar operations. Comparisons with highly vectorized **VPS-32** codes are probably less favorable. It is a good short vector computer achieving nearly the same performance on short vector calculations (22 MFLOPS with average vector length of 80) as on the long vector calculations(23 MFLOPS with average vector length of 5000). Performance using vectors of non-unit stride is not degraded, provided that the stride does not contain a factor of 16, 8, 4 or 2. The worst case, a stride of 16, causes memory load/store time to increase by a factor of 9. The effect such degradation has on the performance of a vectorized DO loop depends on the amount of memory activity as compared to the floating point operations in the loop. The scalar speed of the C2 has been measured at 3 MFLOPS.

The vector length at which it becomes favorable to use vector operations, called the crossover point, is at n= 6. The vector length which achieves half of the maximum performance is at n= 10. However, both of these figures are problem dependent.

1.2.4. Programming [e]

Two higher-level programming languages, FORTRAN and C, are provided on **Mustang** and **Eagle.** The CONVEX FORTRAN compiler is called *fc* (see section 2.1). It is a FORTRAN-77 standard compiler, with extensions for vectorization, some VAX-11 functions and a few extensions unique to CONVEX. Documents CX-4 and CX-5, CONVEX FORTRAN Reference Manual and User's Guide contain detailed information.

The CONVEX C compiler is called *cc* (see section 3.1). The *cc* compiler is an ANSI standard C compiler and has a reference manual (CX-46) and an optimization guide (CX-48). **The C Programming Language, Version 2** by Brian Kernighan and Dennis Ritchie is considered the standard C programming language reference manual by most C programmers.

2. FORTRAN ON THE CONVEX COMPUTERS [d]

CONVEX FORTRAN, the *fc* compiler, is a vectorizing FORTRAN compiler. It contains standard FORTRAN as defined by the American Standard FORTRAN-77 (ANSI 3.9-1978) with extensions. More detailed information may be obtained by using the *man* command and Documents CX-4, *CONVEX FORTRAN Language Reference Manual* and CX-5, *CONVEX FORTRAN User's Guide.*

2.1. Compiling and Loading with fc [e]

To compile and load a FORTRAN program use the *fc* command:

fc [options] files [loader options]

A partial list of *options* follows:

-On Optimization level
n=0 local scalar optimization
n=1 local scalar optimization plus global scalar optimization
n=2 local and global optimization plus vectorization

-c Suppresses the loading. Output from file x.f is x.o.

-cfc Causes the compiler to use 64 bit reals and integers and to accept certain features of the Cray FORTRAN language. Read Appendix G of the CONVEX FORTRAN *User's Guide* (CX-5) for more details.

-cs Produces code that checks array bounds at run time.

-db Produces information for the symbolic debugger, csd.

-in Controls lengths of INTEGER and LOGICAL type with undeclared lengths. n may be 2, 4 or 8 bytes. **The default is 4, which means INTEGER*4 and LOGICAL*4.**

-LST Listing of source file is written to *stdout*

-no No optimization. **(Default if *-On* is not selected)**

-o The executable produced by the loader is written to file specified after the -o. **The default is file a.out.**

-p8 Lengths of INTEGER, LOGICAL and REAL variables with undeclared lengths occupy 8 bytes of storage.

-rn Controls lengths of REAL type with undeclared lengths, where n may be 4 or 8 bytes. **The default is REAL*4.**

-vn Identifies compiler version. Output goes to stdout.

-xr Calls the fxref cross-reference generator.

-72 Truncates source at column 72. **The default is 80 columns.**

NOTE: If your program requires 64 bit accuracy and requires an external library, you must be careful and choose the library with the correct precision. See section 2.2 and CX-16 for more details. All the graphics libraries use the defaults of REAL*4 and INTEGER*4.

This next section describes the *files* argument.

The *files* specified generally have names ending with *.f, .o, .a* or *.s*. The meaning of each of these extensions is:

.f FORTRAN source code to be compiled.
.o Object (binary) files already generated by the compiler.
.a Library file (already compiled).
.s Symbolic assembly language files to be assembled.

The *loader options* options listed are passed on to the loader. Document CX-8, *CONVEX Loader User's Guide*, has more detailed information. The default *fc* command option is to load the FORTRAN program after compilation. This is the preferred method of invoking the loader for FORTRAN programs. It ensures that the required FORTRAN libraries are loaded in the proper order. The most common loader option that a user might use is the *-l* option. This option allows the user to specify additional libraries to be searched to satisfy external references. For example if you wish to use routines in the *veclib* library, you need to specify *-lveclib* as an option on your *fc* command. Note that there is **no** space between the *l* and *veclib* and it must come after the .f files.

Some examples of the *fc* command follow:

fc prog.f

FORTRAN source file *prog.f* is compiled and loaded. The executable file *a.out* is generated.

fc -O2 -LST -r8 -o prog prog.f >& prog.l

FORTRAN source file *prog.f* is compiled with local and global optimization plus vectorization. All real variables are declared REAL*8 (64 bit). The program is loaded and the executable file is *prog*. A source listing with compiler messages is written to file *prog.l*. Be sure to redirect *stdout* and *stderr* when using the -O2 or -LST options, otherwise the listings and compiler messages will be written to the screen.

fc -o prog prog.f exam.o

FORTRAN source file *prog.f* is compiled . The executable file *prog* is generated by linking files *prog.o* from the compilation and the object file *exam.o*.

2.1.1. Compiler Messages [e]

Messages produced by the compiler are directed to standard error (stderr) or the screen for an interactive job. The message consists of:

The line number where the error occurred.

The character position within the line.

The source file name.

A brief description of the error.

An example of a typical compiler error message follows:

fc:Error on line 372.8 of prog.f:Label referenced but not defined.

By default, no source listing is generated by the *fc* compiler. Use the -LST parameter to generate a source listing or use a text editor to locate line numbers.

2.1.2. Execution of FORTRAN Programs [e]

To execute your program after it has been loaded by the compiler, type the name of the executable file. The default name of the executable file is *a.out*. If the option *-o name* was included in the *fc* command, the name of the executable file would be *name*.

FORTRAN unit numbers (except unit 5 and 6) are usually associated with a file by a FORTRAN OPEN statement. If a FORTRAN unit number is not associated with a file name by a FORTRAN OPEN statement, the default file name is *fort.n* where *n* is the unit number in the FORTRAN I/O statement. FORTRAN units 5 and 6 are standard input and output respectively. See section 2.4 of A-8 for redirecting I/O to and from units 5 and 6. Printed output can be routed to the central site laser printer (see section 4.2 of A-8).

A simple script which compiles, loads and executes a typical program follows.

```
#
fc -O2 -LST -r8 -o prog prog.f > & prog.l
prog < datax > & prog.outx
```

The source is on file *prog.f*. File *prog.l* contains a source listing with line numbers, vectorization information, errors and other messages from the compiler. File *prog* is the executable generated by the compiler. File *datax* contains data read from unit 5. File *prog.outx* contains output from the program and run time errors (see section 4.2 of A-8).

A generic script to perform the same tasks could be written as follows:

```
#
fc -O2 -LST -r8 -o $1 $1.f > & $1.l
$1 < $2 > & $1.outx
```

If the above script is on a file *runit*, then the command

```
runit prog datax
```

would execute the same commands as the first example.

2.1.3. FORTRAN Programming Hints [e]

CONVEX extensions to FORTRAN can be found by reviewing document CX-4, *CONVEX FORTRAN Language Reference Manual,* Appendix A. Some items that have presented problems during testing or that are useful extensions are listed below.

2.1.3.1. PROGRAM Statement

A PROGRAM statement is not required, but if it is used it can contain only PROGRAM *pgm* where *pgm* is the name of your main program; no file declarations may be placed on the PROGRAM statement.

2.1.3.2. Columns 73-80

Statement text begins in column 7 and continues to the end of the line (default is 80 columns) or an exclamation mark (which marks the rest of the line as a comment). If columns 73-80 contain sequencing information, the *fc* option *-72* should be used.

2.1.3.3. DOUBLE PRECISION [c]

The type statement DOUBLE PRECISION means 64-bit precision for real variables.

2.1.3.4. OPTIONS

The OPTIONS statement can be used to set options not set by the *fc* command line or to override options in the command line. The options remain in effect only within the program unit in which they are defined, so it is useful for changing options for a single subroutine. It must be the first statement in a program unit, i.e. before the SUBROUTINE statement.

2.1.3.5. SAVE Statement [c]

The SAVE statement retains the values of designated variables in a subroutine or function when that module is exited. At the next CALL to the module, SAVEd variables have the same values as before the previous RETURN. The SAVE statement may be required if the program has come from a computer that always retains the values of variables in a subroutine or function, such as the *-a static* (static allocation) option on the CRAY *cft77* command.

2.1.3.6. SECOND Function [e]

The SECOND function supplied with the *fc* libraries can only be used with the *-cfc* compiler option.

The object code of a C function called SECOND that can be called by FORTRAN programs compiled with the *-p8* compiler option or any compiler option that specifies 64-bit real variables is found in *~howser/sec.o* . You may use the *ln* command (See section 2.2.7.10 of A-8) to insure the file is in your directory, or you may copy it to your directory. This file must be included in the *fc* command line to be loaded and used.

Example:

fc -O2 -p8 prog.f ~howser/sec.o

SECOND requires a dummy argument that is not used (i.e. TIM= SECOND(DUM)). SECOND returns the elapsed time since the last call to SECOND.

2.2. Using VECLIB and VECLIB8 [e]

VECLIB and VECLIB8 are math and scientific libraries supplied by CONVEX which contain many of the subroutines available in the CRAY LIBSCI library. They include the public domain software packages, LINPACK, EISPACK, and MINPACK subprograms. CONVEX provides two libraries, VECLIB and VECLIB8 for compatibility with the FORTRAN allowed data types. VECLIB is designed for the default compiler data types, while VECLIB8 is designed for programs compiled with the *-cfc* and *-p8* compiler options. However, VECLIB8 contains only a subset of VECLIB standard library routines. Consult the *CONVEX VECLIB User's Guide,* CX-16, for more details. The section entitled VECLIB8, under each subroutine, will specify if that routine is available in VECLIB8.

Both VECLIB and VECLIB8 must be explicitly loaded with the *fc* command.

Example:

```
fc -O2 prog.f -lveclib
```

The default data precision types are used by the compiler, so *veclib* is the library loaded.

Example:

```
fc -cfc -O2 prog.f -lveclib8
```

The *-cfc* option requires loading *veclib8,* but be sure the subroutine is included in VECLIB8.

If the required subroutine is not available in VECLIB8, chapter 6 of the LARCLIB directory in CX-3 contains two subroutines (I4TOI8 and I8TOI4) for converting integer arrays from 32-bit to 64-bit representation and vise versa when the *-cfc* or *-p8* option is used on the *fc* command.

3. STANDARD C ON THE CONVEX COMPUTERS [e]

In addition to FORTRAN, CONVEX supports ANSI standard C. The following section describes how to compile, load and execute a program using the C compiler.

3.1. Standard C [f]

CONVEX supports the standard C compiler, called *cc*, which conforms to the ANSI X3J11/90-013 and POSIX standards. The *cc* compiler is the standard C compiler described in **The C Programming Language, Version 2** by Brian Kernighan and Dennis Ritchie. CONVEX supplies *CONVEX* C (CX-46) a user's guide and language reference manual for the *cc* compiler. The *CONVEX C Optimization Guide* (CX-48) and the *CONVEX C Programmer's Reference* (CX-37) are also available from OCO (see chapter 7).

The implementation of C on CONVEX includes the construct "long long integer", which is a 64-bit integer. Additional extensions include an enumeration data type and structure to structure assignment. This version of the C compiler has compatibility mode options, including a backward-compatible mode option for applications that were compiled by non-ANSI C compilers. The default mode generates 32-bit precision. The syntax for the use of the C compiler is:

cc [options] files

The *files* can have the extensions *.c*, *.o* or *.s*, which are C source code, previously compiled code and assembly language code respectively. A partial list of *options* follows:

-g Build a symbol table for use by the csd debugger.

-p Build a monitor file for use by the prof profiler.

-O Optimize.

-o Specify a name for the executable, i.e. *-o prog*. **The default name is a.out.**

4. COMPUTING ENVIRONMENT [e]

The UNIX operating system is designed primarily as an interactive system. The user has the ability to customize his environment and to create new commands (or scripts) to perform frequently executed tasks. UNIX is case sensitive and expects system commands to be entered in lower case letters. The environment can be created automatically at login by using the *.login* and *.cshrc* files (see sections 4.1.4 and 4.1.5) or as needed by changing the environment or C-shell variables (see section 4.1.2). Usually, the *.login* file is used to initialize batch vs interactive environment, your search path for commands and environment variables. The *.cshrc* file is used to insure that all your aliases are passed on to subsequent C-shells and to initialize C-shell variables.

4.1. Logging Onto CONVEX [e]

When you get connected to **Eagle** or **Mustang,** the system responds

CONVEX UNIX, RELEASE V9.1 (machine_name)

and then prompts for your login name with

login:

Type your login name followed by a carriage return. The carriage return is followed by a prompt for your password:

password:

If you type either your login name or password incorrectly, the system prompts you again. If you hit the backspace in an attempt to correct an error, your login attempt fails. Just try again and be more careful. If you can't login at all, call Password Validation at 864-8282. If your login is successful, you are told the name of your default account (group) (see section 1.3.1 of A-8).

4.1.1. Changing Your Password [e]

You must change your initial password when you first login to both **Eagle** and **Mustang.** You must also change your password on both C2's at least once a year. Once changed, a password may not be changed during the next two weeks. Your new password must meet the following security requirements:

1. It must be at least six characters long.
2. It must have at least two alphabetic and one numeric or special character.
3. It must not be any permutation of your login name.
4. It must differ from your old password by at least three characters.

Also, words found in a dictionary with only a single digit appended to the end or added at the beginning to form the password are highly susceptible to being compromised and should not be used. The command to change your password is *passwd.* It is an interactive command, so all that you must do is enter the command and *passwd* prompts you for a response. As you enter your old and new passwords at the appropriate prompts, notice that the system does not echo your password to the screen. This is a security feature and the reason that the system prompts you twice for your new password. If the two entries for your new password don't match, the system does not change your password and prompts you again to enter your new password. If you forget your password, contact Password Validation at 864-8282 for assistance.

4.1.2. Environment and C-Shell Variables

Data about your environment such as your home directory and terminal type is maintained in two sets of variables called environment and C-shell variables. Environment variable values are inherited by all programs executed by the shell, including new shells that you spawn or fork. C-shell variables are inherited by execution of your *.cshrc* file whenever you spawn a new C-shell. Many of your environment and C-shell variables are defined by the system administrator in your initial *.cshrc* and *.login* files, but you may change these variables by using the *setenv* or *set* commands respectively. These variables generally define information that programs need to execute correctly. Your current environment variables can be displayed by typing:

printenv

The *setenv* and *set* commands have slightly different syntax, as illustrated below

setenv variable string

and

set variable=string

where *variable* is the name of the environment variable and *string* is the new value. Some of the C-shell variables are Boolean and are either *set* (i.e. true) or *unset* (i.e. false), such as *noclobber* and *ignoreeof*. These two C-shell variables are used in the sample *.cshrc* file in section 4.1.5.

Some common environment variables include HOME, PATH, TERM, USER, SHELL, PRINTER and EDITOR. These variables are discussed in detail in the CONVEX UNIX Primer (CX-2) in chapter 10. Some are illustrated in the examples of *.login* and *.cshrc* files that are discussed in sections 4.1.4 and 4.1.5.

4.1.3. Personalizing Your Environment [c]

A nice feature of UNIX is the ease with which you can change your environment. It can be done automatically with the *.cshrc* and *.login* files, or it can be custom tailored for a single interactive session. When you are assigned a login name and password, the system administrator provides a default *.cshrc* and *.login* file in your home directory. You may change these to suit your needs. A frequent change that users make to the *.cshrc* file is to add aliases. Aliases allow you to create another name for frequently used commands. For example if you type (or add to your *.cshrc* file) the next line

alias rm "rm -i"

then each time you type *rm* to delete a file the system automatically prompts you to insure that you really intended to remove the specified file. Otherwise you would have to type

rm -i

each time you wanted the system to prompt you on a file removal.

The directory */usr/local/admin/skel* has several sample hidden files that you may use to customize your environment, including *.login* and *.cshrc* files. These files are similar to but not exactly like the sample files in sections 4.1.4 and 4.1.5. Additionally there are sample skeleton files for the *.exrc, .mailrc, .forward, .netrc, .rhosts* and *.logout* files that are discussed in A-8 or elsewhere in this manual (See the Guide to Hidden Files at the end of the Mini Manual). The next two sections have sample *.login* and *.cshrc* files that are similar but not identical to the skeleton files that the system administrator gives you initially. If you modify the default files or bring your *.login* and .cshrc files from another machine and discover that something isn't working, then you can copy the default *.login* and *.cshrc* files from the directory */usr/local/admin/skel* to your home directory.

4.1.4. Your .login File [e]

The *.login* file is automatically executed every time that you log into the C-shell. (An analogous *.logout* file is automatically executed every time that you log out of the C-shell, if such a file exists.) An example of a typical *.login* file is given below. Everything to the right of the pound sign (#) is a comment to explain the various entries. Your *.login* file may not look exactly like this, but you can tailor it to suit your needs. This *.login* file has been tailored to allow automatic notification of new notes (see section 8.3 of A-8) and to test for batch or interactive environment.

```
setenv MAIL= /usr/spool/mail/john     # define mbox address
umask 022                             # deny write permission to group and world
setenv DELIVER bin999                 # your delivery information
setenv NFSEQ "*"                      #
set machine='hostname'                #
if ( "$machine" == "eagle" ) then     # Read notes only on eagle
  set nres='checknotes -v'            # Check for new notes
  if ( "$nres" != "" ) then           # New notes exist
    echo "New notes -- $nres"         # List new notes categories
    echo -n "Read Notes? "            #
    if ("$<" =~ [yY]*) then           $
      clear                           # Clear screen
      autoseq                         # Read new notes
    endif                             #
  endif                               #
endif                                 #
biff y                                # automatic notification when mail arrives
mesg n                                # don't allow other users to talk to you
date                                  # give date and time at login
setenv TERM vt100                     # set default terminal type to vt100
stty erase "^H"                       # define the backspace character
setenv VISUAL /usr/ucb/vi             # use ~v for vi in mail
setenv EDITOR /usr/ucb/vi             # use ~e for vi in mail
setenv LPP 48                         # set default page length to 48 lines
endif
```

More information on these commands can be found using *man* for the specific command or for *csh,* which is the command to spawn a new C-shell. The *.login* file only shows up in a list of files generated by the *ls* command if the *-a* option is used.

4.1.5. Your .cshrc File [a]

Every time you log onto the CONVEX or spawn a new C-shell the *.cshrc* file is automatically executed. It may also be executed by typing:

source .cshrc

An example of a typical *.cshrc* file is given below. Everything to the right of the pound signs (#) is a comment to explain the various entries. Your *.cshrc* file may not look exactly like this one, but remember that you can tailor it to suit your needs.

```
set mail=(/usr/spool/mail/john)                    # set up for mail notification
set path=(~/bin /usr/local /bin%# command search order
        /usr/bin /usr/local/bin%#
        /usr/ucb /usr/convex .)%#
if (! $ ? ENVIRONMENT) then%# check environment
 %# for batch or interactive
set history=20                                     # save last 20 commands on history file
set prompt="mustang.john% "                        # set prompt to show machine & user
set ignoreeof                                      # disable ^D for logout
set noclobber                                      # avoid accidental file overwrite
alias h history                                    # shorthand notation for history
alias c clear                                      # clear the screen
alias rm "rm -i"                                   # avoid accidental removal of files
```

The *.cshrc* file appears in a list of files only if the *-a* option is used with the *ls* command. More information on the commands listed above can be found using the *man csh* command.

4.2. Interactive Computing [e]

UNIX is basically an interactive operating system, but unlike some other operating systems the user can be executing multiple tasks, called processes under UNIX, at the same time. This feature tends to make your terminal sessions more productive, since you don't have to wait for one process to finish to start another. However, there is a limit of 40 processes per user.

4.2.1. Processes

A process is a program that is running. The most frequently running program is the command interpreter called *csh* and referred to as the C-shell. CONVEX UNIX also supports the Bourne shell, which is called *sh*. Every time the user issues a command the *csh* spawns (or forks) a new process. The spawned process is the child of the process that created it. Processes may be run in the foreground or background. Usually they are run in the foreground; however by running processes in the background, you are able to do a lengthy task like a compile, while continuing to do other work in the foreground. For example, to compile a FORTRAN program, prog.f, in the background on CONVEX, you type:

fc prog.f &

The ampersand (&) causes the program to be compiled in the background. If you are in the C-shell and logout, the process continues executing if you have redirected standard input and output (see section 2.4 of A-8). In the Bourne shell you must precede the *fc* with a *nohup*, otherwise the process dies when you logout. For example,

nohup fc prog.f &

would continue to execute after you logout.

You can check on the status of processes with the *ps* command, which stands for process status. Each process is assigned a number called a Process IDentifier (PID). This number is important if you have a hung process and want to kill it (see section 4.2.2). Processes can also be moved between the foreground and background with the *fg* and *bg* commands. A job in the foreground may also be stopped with the ^Z (Control Z). Using ^Z does not kill a process, it merely suspends it. If you attempt to logout while a job is stopped the system gives you a warning message:

There are stopped jobs.

Another attempt to logout will succeed in logging out, but stopped jobs are killed. This gives you the option of restarting the stopped process before you logout.

4.2.2. Killing a Runaway Process

If you have a runaway process running in the background, the *ps* command gives you the PID which you can then use to kill the process. For example to kill the process with PID 1234, type:

kill -9 1234

Sometimes you may have a hung terminal or a runaway process running in the foreground. There is no need to panic. First try using the ^C (control C) signal. If that doesn't work, just login to the machine again from another terminal and enter the *ps* command. Then use the *kill* command as just illustrated to kill the process(es) associated with the other terminal. If you kill the wrong C-shell by mistake, you are logged off the system, so just try again.

4.2.3. Logging Out

Usually, you log off the system with the *logout* command. If you are no longer in your login shell then you must use the *exit* command. The ^D (Control D) also logs you off the computer if you do not have the variable *ignoreeof* set in your *.cshrc* file (see section 4.1.5). You may also create a special *.logout* file that is executed every time that you logout. A sample *.logout* file might contain the following:

```
clear          #clear your screen
/bin/rm a.out  #remove generic executables
/bin/rm core   #remove core file
date           #display date and time at logout
```

4.3. Batch Processing on CONVEX [c]

Batch processing capability on **Eagle** and **Mustang** is provided with the CXbatch utility. CXbatch is similar to CRAY's Network Queuing System (NQS). Many CXbatch commands are similar to NQS commands, however, CXbatch does not recognize commands embedded in scripts that begin with the character string '# QSUB' like NQS. CXbatch scripts use the string '# @$' rather than '# QSUB'. The next section provides an introduction to CXbatch. It may be necessary to tune the various queue limits to the mix of jobs at LaRC to obtain the optimal throughput on the C2's. A note under the topic *CXadmin* (see section 8.3 of A-8) gives the current queue definition. Changes to the queues are announced in the message of the day, referring you to the new entry under *notes CXadmin*. Table 4.1 summarizes the CXbatch queues. Document CX-23 is the *CONVEX CXbatch User's Guide*.

4.3.1. CXbatch [e]

You can use CXbatch to submit jobs to either **Eagle** or **Mustang,** and to query the system for status or kill a running process. Before attempting to use CXbatch, your *.cshrc* file must be modified to check for a batch or interactive environment (see section 4.1.5), and your path must be set prior to this check. CXbatch does not execute your *.login* file. CXbatch commands include:

qsub	Submits a request to a batch queue.
qstat	Displays the status of CXbatch queue and requests.
qdel	Deletes or signals CXbatch request.
qlimit	Displays batch limits and shell strategy

To use CXbatch, you must first create a shell script (see section 2.1.3) of commands to be executed in batch mode. In general, scripts are used to change into a scratch directory for the compilation, loading and execution of a large application program. CXbatch scripts execute in the shell specified by the shell strategy, which is set to your login shell (the C-shell for most users). Some of the more frequently used *qsub* options are:

-a 5pm	Submit job to CXbatch queue at 5 PM.
-lw 2Mw	Establish a per-process working set size limit of 2 Mw.
-lt 1000	Establish a per-process CPU time limit of 1000 seconds.
-lT 1000	Establish a per-request CPU time limit of 1000 seconds.
-mb	Send mail when the request begins execution.
-me	Send mail when the request ends execution.
-q large	Send job to the large CXbatch queue

Note that Cxbatch has no mechanism to support a per-process or per-request memory limits. Only the per-process working set size may be specified. Check the *man* page for details on other *qsub* options.

An example of a sample CXbatch script for user *john* follows:

```
#
# @$-lw 2mw
# @$-lt 1000
# @$-me
#
cd /scr/john
/bin/time fc -LST -o prog prog.f > & prog.l
/bin/time prog < prog.in > & prog.out
```

This job when submitted to CXbatch, runs in the scratch directory of *john,* with a working set size limit of two million words and a CPU time limit of 1000 seconds. The job times both the compilation and execution and the request sends *john* a mail message when it completes.

The syntax of a batch job submission using CXbatch is:

qsub [options] script_name

The *options* may include any options that can be embedded in the CXbatch script. Any options specified with the *qsub* command override the same options that may be specified in the script. Once you submit the job, the system responds:

request *request_id* **submitted to queue:** *queue_name*

Refer to *notes CXadmin* for the current description of the CXbatch queues. The *request_id* is used by the *qdel* command, if it becomes necessary to kill the request. To kill a request, use:

qdel -k request_id

The *-k* option is required only if the process has begun execution.

The *qlimit* command displays the resources available on **Eagle** and **Mustang** and the corresponding parameter names to be used by *qsub*. The shell strategy displayed is the name of the shell which interprets script commands (your login shell, usually the C-shell), if you do not specify a shell.

The *qstat* command is used to query the system about the status of your request(s) as illustrated below:

qstat -a	Show all requests.
qstat -m	Show medium summary of queue requests.
qstat -l	Show long summary of queue requests.
qstat -x	Show queue header summary.

The *qstat* command has other options, check the *man* page.

Other changes to CXbatch at this release include two new options, *-e filename* and *-o filename,* which are used to rename the .e and .o files associated with the termination of a CXbatch job. These files can be a valuable source of debugging information if your job terminates abnormally. Additionally, if nothing is written to *stderr* or *stdout,* these files are created with zero length if they have not been renamed with the *-e* and *-o* options. If these options have been used and nothing is written to the standard error or output files, a zero length file is not created.

CXbatch Queue	Working Set (Mw)	CPU Time (seconds)
interactive *		1200
short	4	1200
medium	4	3600
long	4	7200
largedebug	8	300
large	8	3600
large-long **	8	10800

* *Interactive is not a true CXbatch queue, but is included for completeness.*

** *Active only during non-prime shifts.*

Table 4.1 CXbatch Queues

These CXbatch queues were modified with the hardware upgrades to **Eagle** and **Mustang.** Changes to the queues are announced in the Message-of-the-Day (found in */etc/motd*), *notes CXadmin* and by computer bulletin. Space has been left in **Table 4.1** for you to make changes to the queue definitions. Jobs are automatically routed to the correct CXbatch queue as specified by the qsub options. The *notes CXadmin* category on **Eagle** and **Mustang** contains the latest queue information.

WARNING: Any job submitted to CXbatch that exceeds the established queue limits is deleted by the CXbatch daemon and will never appear with a *qstat* command. However, the daemon does send you electronic mail to inform you that your request could not be queued.

5. DEBUGGING AND PERFORMANCE ANALYSIS [f]

CONVEX supports four debugging utilities, including a new X-based implementation of the UNIX debugger, *dbx*, called CXdb. CONVEX OS also supports the CXpa performance analyzer and three profiling utilities, *prof*, *bprof* and *gprof*. A new utility, *build*, supports interprocedural analysis of FORTRAN and C applications, see Section 5.7.

The *pmd* Post-Mortem Dump and *csd* Source Code (Interactive) debuggers both require that a symbol table be built during compilation. The *adb* debugger can be used when your program terminates abnormally and a symbol table was not built.

The CONVEX profiling utilities, *prof*, *bprof* and *gprof*, all require that the program be compiled with a special option. Then the program is executed normally. An analysis of the profiling statistics is generated using the profiling utilities. The *man* pages for these profilers are in CX-28.

The *nm* utility can be used to find the routine in which your program aborted without having to recompile by typing

nm -gn [executable_file]

which produces a list of all external symbols (subroutine names) in numerical order by memory location. Additionally, CONVEX places all code on **READ_ONLY** pages in memory, so that it should be impossible to overwrite your code with data.

5.1. CXdb Symbolic Debugger [f]

The CXdb *(cxdb)* debugger is an interactive symbolic debugger that has the capability to perform the following functions:

1. Setting breakpoints, tracepoints and watchpoints.
2. Controlling program execution.
3. Displaying and changing data.
4. Managing image (executable, core and running process) files.
5. Defining debug variables, aliases and macros.

The manual for *cxdb*, *CONVEX CXdb User's Guide* (CX-44), is organized such that the various sample debug sessions throughout the manual refer to source code found in Appendix A. The manuals *CONVEX CXdb Concepts* (CX-43) and *CONVEX CXdb Reference Manual* (CX-53) are also available. Multithreaded applications can also be debugged interactively using *cxdb*. Compilation at level -O3 is required for these applications. Almost all the *dbx* commands are implemented with this release of *cxdb*, either directly or through the use of supplied aliases and macros. Additionally, several CONVEX-specific commands have been included. The *cxdb* utility is very similar to the CRAY utility *cdbx*. Appendix C of CX-44 contains a cross-reference between the commands for the CONVEX line-oriented debugger *csd* and *cxdb* commands.

To use *cxdb*, it is recommended that each source code module be on a separate file. Using optimization inhibits the effectiveness of the debug tables that are generated. The best results from using *cxdb* are with unoptimized code (compiler level -no). However, *cxdb* is unique in the respect that it allows debugging at all optimization levels (-no, -O0, -O1, -O2, and -O3). To use the full capabilities of *cxdb*, you must compile your program using the -cxdb option with the latest version of the CONVEX FORTRAN or CONVEX C compiler. The -cxdb option tells the compiler to generate the debugging information *cxdb* needs in subdirectory .CXdb.

This release of *cxdb* supports the CXwindows interface (CONVEX version of the X Window System) in addition to the "batch mode" which features file input/output (-b option). As there are no default executable, core or running process file names, in order to debug your program, you must specify the executable file and a process generated from that executable file. *cxdb* uses the image of the process to map between the current state of the process and the original source code. A simplified syntax of the *cxdb* command follows:

cxdb [-e efile] [-c cfile] [-a process-id]

-e specifies an executable file, efile, to debug.

-c specifies a core file, cfile, to debug.

-a attaches cxdb to the running process with the specified process-id.

Other *cxdb* options are available and includes batch mode selection, immediate execution of specified commands or command file and directory additions to the default search path. Check the *man* page for all available options. See Chapter 2 of CX-44 for more detailed information on invoking *cxdb*.

5.2. adb Debugger

The *adb* debugger is an object-level debugger for use at the assembly language level. It is described in *CONVEX* adb (CX-10). It does not require recompilation or special compiler options and works with all levels of optimization. To access the *adb* debugger type

adb [executable_file] [core]

where *executable_file* is the execution file that aborted and *core* is the core image generated. This briefest of forms of *adb* shows which routine the program was executing when it aborted. The address given by *adb* can be used with the output of the *nm -gn* command to see whether the error was near the start, middle or end of the routine.

5.3. csd Source Code Debugger [c]

The *csd* debugger is an interactive symbolic debugger that has the capability to perform the following functions:

1. Setting breakpoints and traces.
2. Controlling program execution.
3. Displaying and changing data.
4. Managing image (core or executable) files.
5. Defining debug variables and aliases.

All the *dbx* commands are implemented with this release of *csd*. Additionally, several CONVEX-specific commands have been included. The *csd* utility is described in Chapter 2 of the *CONVEX Consultant User's Guide* (CX-11). The *csd* utility is similar to the CXdb utility *cxdb*, but without the X interface. To use *csd*, symbol tables must have been generated when the program was compiled by selecting the *-db* option for the *fc* compiler. In general, better results from *csd* are obtained when the subroutines being debugged have been compiled without any optimization. A simplified syntax of the *csd* command follows:

csd [-r] [symfile] [corefile]

-r — execute *symfile* immediately, without waiting for *csd* commands.

symfile — select the file that contains the symbolic information. **The default is a.out.**

corefile — select the file that contains a dump of the user memory image. **The default is core.**

Other *csd* options are available, check the *man* page.

5.4. pmd Post-Mortem Dump

The *pmd* debugger generates a Post-Mortem Dump to the file *stderr* if execution aborts and a core file is created. It can be used with all levels of optimization, but does require that the *-db* option be selected for the compilation. The syntax of *pmd* is

pmd [-alsv] [-d n:m ...] a.out

where *a.out* is the name of the default executable (you may have called it by another name). The *-d* option allows you to specify limits on the printing of arrays. There are other options not indicated above; check the *man* pages or *CONVEX Consultant User's Guide* (CX-11) The remaining options have the following meanings:

- a Print address registers in several formats.
- l Generate a long format dump.
- s Generate a short format dump.
- v Include the contents of the vector registers in the dump.

5.5. Timing Your Program [e]

The CONVEX computers have an architecture that is not too dissimilar from the CRAY-2 and CRAY Y-MP. The FORTRAN compiler is FORTRAN-77, but has some extensions and restrictions that are unique to CONVEX. There is no explicit vector syntax, so programs written in standard FORTRAN-77 should need little work to execute correctly on a C2.

Once your program is executing correctly, one of the first things that you may want to do is to time it. You can use the */bin/time* command to time your compilation and execution in the shell script (See section 2.1.3 of A-8) that runs your program, in the following manner.

```
/bin/time fc -LST -o prog prog.f > & prog.l
/bin/time prog < prog.in > & prog.out
```

The */bin/time* command returns elapsed time, CPU time and system time labeled as *real, user* and *sys* respectively. The *time* command is built into the C-shell and returns a different output than */bin/time,* which is not as convenient to interpret. The FORTRAN compiler *fc* is discussed in section 2.4.1 and file redirection (i.e. using the < and > symbols) is discussed in section 2.4 of A-8. **A FORTRAN-callable SECOND function in the *fc* compiler's libraries is available only when the *-cfc* option is used.** A C function called SECOND is available and callable by FORTRAN programs when the *-r8* or *-p8* options are used, but it must be explicitly loaded (see section 2.1.3.6).

Additionally CONVEX has several timing utilities that are described in Document CX-11, *CONVEX Consultant User's Guide.* These profilers are called *prof, bprof* and *gprof* and are discussed in the next section.

5.5.1. prof, bprof and gprof [c]

These CONVEX profiling utilities all require that the program be compiled with a special option. Then the program is executed normally. Finally an analysis of the profiling statistics is generated using the *prof, bprof* or *gprof* utilities. The *man* pages for these utilities are in *CONVEX Consultant Programmer's Reference (CX-28).*

5.5.1.1. prof

The *prof* utility is the general profiler to generate basic profile data such as: total cpu time per routine and percentage of cpu time in relationship to the other profiled routines; number of calls; and average cpu time per call. To use *prof* the *-p* option must be selected during compilation and loading. After executing your program, the profiler would interpret the data gathered with the following command:

prof a.out mon.out

where the file *a.out* is your default executable name and *mon.out* is the default name for the file containing profiling information. For more detail on the use of *prof,* check CX-11 or the *man* pages.

5.5.1.2. gprof

The *gprof* utility builds a call graph so you can see where a specific routine is called most frequently. The utility is used just like *prof,* except that the compiler option to specify is *-pg* and the default name for the file containing profiling information is *gmon.out.* For more detail on the use of *gprof,* check CX-11 or the *man* pages.

5.5.1.3. bprof

The *bprof* utility counts the number of times that each line of source code is executed and forces compilation to be done with no optimization, which could skew statistics. The option to specify on compilation is *-pb* and the default file containing profiling information is *bmon.out.* For more detail on the use of *bprof,* check CX-11 or the *man* pages.

5.6. CXpa Performance Analyzer [e]

The interactive code profiler, CXpa, supports selective, interactive profiling of routine, loop, and basic block code sections. CXpa works with programs compiled by the Convex FORTRAN and C compilers.

The FORTRAN command line to compile the executable (see section 2.1) required for routine-level and loop-level profiling (the *-pa* option) is:

fc [options] -pa [CXpa options] files [loader options]

To invoke CXpa on an executable file, enter the following command line:

cpa [-f] [-nx] [-udir...] objfile

See the *man* page for *cpa* for a description of the various options.

When CXpa is invoked, you see the following message:

Convex Performance Analyzer

Type 'help' for help
Reading executable 'objfile'
(cpa)

The following is a list of the command available in CXpa:

Execution control commands:

run	- create a cpa.pdf file
rerun	- reruns the last run command entered
quit	- end a CXpa session

Profiling control commands:

deselect	- cancels selected monitor points
monitor	- selects all parts of a program for monitoring
profile	- changes the name of the output PDF

Analysis control commands:

analyze	- analyze a loop-level profile

Information Commands:

file	- change current source file, used (a) to add one or more directory (b) to list current directory list
help	- get more information about a command
list	- display text files
source	- execute a command file from within CXpa
status	- check status of PDF during a session

For more information, see *man cpa*. Or refer to *CONVEX Performance Analyzer (CXpa) User's Guide* (CX-40) and *CONVEX CXpa Programmer's Reference* (CX-41).

5.7 The Application Compiler [f]

The Application Compiler is invoked with the *build* command, which is modeled after the *make* command. The Application Compiler is described in the manual *CONVEX Application Compiler User's Guide* (CX-42). The Application Compiler may be used with both FORTRAN and C programs. Its purpose is to go beyond global optimization of a code to the level of interprocedural analysis. Interprocedural analysis is critical to the effective parallelization of any application.

The Application Compiler builds a database of program information that describes the flow of data and control between procedures. This removes many restrictions on optimization forced on a compiler that is only processing one routine at a time. It also automatically performs certain optimizations, such as procedure inlining that previously were manual. Various checks such as for over or under subscripted arrays, uninitialized variables and type checking can be done automatically. The Application Compiler can also be run in parallel with the *-j* option, which can reduce compile time, especially on **Mustang.**

The *build* command looks for a file called *buildfile* or *Buildfile* by default. This text file contains all the directives for *build*. You may specify another name with the *-f* option. Chapter 2 of CX-42 describes the various options available for *build*. Chapter 3 describes how to create the *buildfile*. The script *makebuildfile* creates a *buildfile* from an existing *makefile*. The script is described in Appendix B. Appendix A describes the supported compiler options and Appendix C describes errors messages.

6. CONVEX UTILITIES [e]

Mathematica, the X Window System (X) and OSF/Motif are supported by CONVEX. Mathematica is a symbolic manipulation language that is similar to Macsyma. Release 4 of X11 is the supported version of X, which is called CXwindows on the C2's.

6.1 Mathematica [e]

Mathematica is an interactive system for doing mathematical computation. It handles numerical, symbolic and graphical computations, and incorporates a high-level programming language. Mathematica is available only on **Eagle** and can run under X-Windows. The reference for the Mathematica system is **Mathematica: A System for Doing Mathematics by Computer** (Second Edition, 1991) by S. Wolfram, published by Addison-Wesley. You must add the directory */usr/local/unsupported/bin* to your *PATH* environment variable (see section 4.1.2) in your *.cshrc* or *.login* file in order to use Mathematica. Mathematica may be executed by entering:

math [-noprompt] [-remote] [-run command]

Options

-noprompt	Produce no input or output prompts.
-remote	Use MathTalk packet-based communication protocol (see mathremote)
-run command	Run a Mathematica command during initialization.

To use Mathematica with X-Windows, set the environment variable *DISPLAY* to the network name of the workstation or X-Terminal to which you are connected, appended the string with *:0.0*. For example, if your workstation has the network name **works,** use the following command on **Eagle:**

setenv DISPLAY works.larc.nasa.gov:0.0

On the workstation **works,** either before logging into **Eagle** or in a separate window, enter:

xhost +eagle.larc.nasa.gov

When you execute *math,* you see this line after the start-up message:

-- X11 windows graphics initialized --

Graphics will now be directed to the workstation **works.**

Mathematica reads the file *init.m* when it begins and the file *end.m* when it exits. Since the CONVEX math library is listed when you execute *man* math, you must do the following to read the *math man pages:*

nroff -man /usr/local/unsupported/src/math/Install/man/math.1

6.1.1 Remote Execution of Mathematica [e]

The *mathremote* command runs the remote-access version of Mathematica. It may be executed by entering:

mathremote [-crc] [-run command]

Options

-crc	Enable CRC error checking.
-run command	Run a Mathematica command during initialization.

The *man* page for *mathremote* may be display by entering:

nroff -man /usr/local/unsupported/src/math/Install/man/psfix.1

6.1.2 PostScript Processing and Mathematica [e]

The *psfix* command takes one or more files containing PostScript graphics descriptions produced by Mathematica and produces a file suitable for output to a standard PostScript rendering device, such as a PostScript printer or PostScript display driver. There are too many options to *psfix* to describe here. See the *man* page by entering:

nroff -man /usr/local/unsupported/src/math/Install/man/psfix.1

6.2 OSF/Motif and CXwindows [e]

Motif is a graphical user-interface from the Open Software Foundation (OSF). It consists of a window manager, a widget set, and a user-interface language. Debate about a standard user-interface is on-going, however many workstation vendors have chosen to support the Motif look and feel (i.e. IBM, HP, SGI, DEC). Motif is also available on **Eagle** and **Mustang.**

On the Convex computers, Motif is provided as part of the CXWindows software package. The Motif window manager, *mwm* is available in the */usr/bin/X11* directory. This window manager provides the capability of opening, closing, moving and resizing windows with a 3-D look. Refer to the **OSF/Motif User's Guide,** published by the OSF further information on using *mwm.* A copy of this document, as well of the full set of documentation available from OSF, may be obtained in the LCUC directory on the Mass Storage Subsystem:

@lcuc/text/motif/doc

The directory contains the following documents in both PostScript and ASCII formats:

OSF/Motif User's Guide
OSF/Motif Programmer's Guide
OSF/Motif Programmer's Reference
OSF/Motif Porting Guide
OSF/Motif Style Guide
OSF/Motif Release Notes
AES User Environment Volume

The Motif widget set is available in the */usr/lib* directory as *libXm.a.* Refer to the *OSF/Motif Programmer's Guide* and *OSF/Motif Programmer's Reference* For information on developing programs using the Motif widget set.

The OSF/Motif User-Interface Language, *uil,* is also available in the */usr/bin/X11* directory. The *uil* program provides an alternate language (as opposed to C) for creating Motif applications. Refer to the *OSF/Motif Programmer's Guide uil.*

The X Window System (X) is a network-transparent graphical window-based software system that was developed at MIT in 1984. The architecture of X, which is called CXwindows in the CONVEX implementation, is based on a client-server model. CXwindows is based on Version 11, Release 4 of X. Display servers are programs residing on workstations, X terminals, and PC's, which provide display capabilities and manage user input devices (keyboard, mouse, etc.). Clients are application programs that perform specific tasks and may be run on workstations, PC's or **Eagle** and **Mustang.** Additional information on CXwindows, including more detailed references may be found in section 3.4 of *SNS Programming Environment User's Guide* (A-8), or in the *notes Xinfo* topic.

7. CONVEX DOCUMENTATION [f]

Users of the CONVEX computers have several sources of information and assistance (see chapter 8 of A-8). Since the C2's run under a version of UNIX much information is available on-line. However, ACD still provides substantial printed documentation that is described below.

ACD distributes a subset (highlighted in **boldface** in **Table 7.1** on the next page) of the user manuals that describe **Eagle** and **Mustang** to all **SNS** Document Librarians. The documentation listed in **Table 7.1** is available from the Operations Control Office (OCO). You may order a personal copy of any of these manuals by calling OCO at 864-6562, or by sending electronic mail to *oco@eagle,* for overnight service. This list is accurate by revision level for CONVEX OS Version 9.1. Three of these CONVEX manuals have been compiled by ACD personnel: CX-1, *CONVEX Mini Manual;* CX-3, *CONVEX Mathematical Libraries;* and CX-26, *CONVEX LINPACK and EISPACK Subroutines.* Each newly validated **SNS** user receives copies of CX-1; CR-1, the *CRAY Mini Manual;* and A-8, the *SNS Programming Environment User's Guide.* There are sufficient quantities of CX-2, CX-4 and CX-5 in OCO for any user to get a personal copy. Since UNIX has on-line documentation, only a limited number of most manuals are available from OCO for individual distribution. Demand determines the number of copies that are kept in stock.

7.1. The CONVEX UNIX Primer

Document CX-2, the *CONVEX UNIX Primer,* is a useful manual for the novice UNIX programmer. It is designed to assist you to

1. Login to a CONVEX system.
2. Use basic UNIX commands.
3. Communicate with other users.
4. Create files.
5. Compile, load and execute programs.
6. Understand shells and the hierarchical file system.
7. Learn the basics of the visual editor, *vi.*
8. Learn how to use the *make* utility.

If you use the exercises at the end of each chapter, CX-2 may be used as a self-paced tutorial on CONVEX OS.

CSCC Doc No	Title	CONVEX Doc No
CX-1f	**CONVEX Mini Manual (February 1993)**	
CX-2a	**CONVEX UNIX Primer**	**710-000221-202**
CX-3a	**CONVEX Mathematical Libraries (January 1990)**	
CX-4c	**CONVEX FORTRAN Language Reference Manual**	**720-002230-003**
CX-5c	**CONVEX FORTRAN User's Guide**	**720-000030-208**
CX-6d	**CONVEX OS Man Pages for Users**	**710-015830-000**
CX-7	**CONVEX Text Editor's User's Guide**	**740-000430-000**
CX-8d	CONVEX Loader User's Guide	710-008630-000
CX-9d	CONVEX OS Man Pages for Programmers	710-004030-002
CX-10a	CONVEX adb Debugger User's Guide	740-002630-203
CX-11a	CONVEX Consultant User's Guide	740-002530-203
CX-12d	CONVEX Internet Services User's Guide	710-002530-204
CX-14a	CONVEX Network File System User's Guide	710-001530-203
CX-16b	**CONVEX VECLIB User's Guide**	**710-011030-000**
CX-17	CONVEX LSQPACK User's Guide	740-002230-202
CX-18	CONVEX Guide to Software Development	710-001930-200
CX-19a	CONVEX Notesfile Reference Manual	
CX-20b	CONVEX FORTRAN Master Index	720-003230-001
CX-21	CONVEX GNU Emacs Manual	
CX-22	CONVEX Make Utility	
CX-23a	CONVEX CXbatch User's Guide	710-002730-205
CX-24	CONVEX nroff/troff Manual	
CX-25a	CONVEX CXbatch Concepts	710-006630-003
CX-26	**CONVEX LINPACK and EISPACK Subroutines (January 1990)**	
CX-27a	**CONVEX FORTRAN Optimization Guide**	
CX-28	**CONVEX Consultant Programmer's Reference**	**710-004130-000**
CX-29a	CONVEX CXbatch Programmer's Reference	710-004430-003
CX-30	CONVEX Network File System Programmer's Reference	710-004230-000
CX-36a	CONVEX FORTRAN Programmer's Reference	720-001630-002
CX-37a	CONVEX C Programmer's Reference	720-001530-001
CX-38a	CONVEX CXwindows Programmer's Reference	710-004530-002
CX-39	CONVEX OSF/Motif and CXwindows User's Guide	
CX-40	CONVEX Performance Analyzer (CXpa) User's Guide	710-007230-002
CX-41	CONVEX CXpa Programmer's Reference	710-004730-002
CX-42	CONVEX Application Compiler User's Guide	720-004030-001
CX-43	CONVEX CXdb Concepts	710-015330-001
CX-44	CONVEX CXdb User's Guide	710-015530-001
CX-45	CONVEX X Widgets User's Guide	
CX-46	CONVEX C Guide	720-000630-205
CX-47	CONVEX Toolbox Man Pages	710-004930-001
CX-48	CONVEX C Optimization Guide	720-001130-202
CX-49	CONVEX CX/Motif Programmer's Reference	710-008930-001
CX-53	CONVEX CXdb Reference Guide	710-015430-002
CX-54	CONVEX The X Primer	710-022630-000

CX-60	CONVEX POSIX Impacts	710-002130-000
CX-61	CONVEX POSIX Concepts	710-005030-000
CX-62	CONVEX POSIX Conformance	710-002030-200
CX-76a	CONVEX Architecture Reference	081-009330-000
CX-78b	CONVEX OS Tutorial Papers	710-011130-000

Table 7.1 CONVEX Documentation [f]

CX-1 SECTION GUIDE TO HIDDEN FILES [d]

File	Description	Section
.cshrc	Executes when a C-shell is spawned	4.1.5
.exrc	Establishes environment for vi full screen editor	2.3.1, A-8
.forward	Tells mail where mail is to be forwarded	3.1.3, A-8
.login	Executes when you log into a UNIX machine	4.1.4
.logout	Executes when you log off a UNIX machine	4.2.3
.mailrc	Establishes environment for electronic mail	3.1.4, A-8
.rhosts	Allows use of rlogin, rcp and remsh	5.2, A-8

CX-1 SECTION GUIDE TO COMMANDS [f]

Command	Description	Section
adb	invoke object level debugger	5.2
alias	establish alternate command names or mail aliases	4.1.3
apropos	search for command related to keywords	8.1.4, A-8
bprof	profile execution by line	5.5.1.3
build	invoke the Application Compiler	5.7
cat	concatenate one or more files	2.2.7.3, A-8
cc	invoke standard C compiler	3.1
cd	change working directory	2.2.7.7, A-8
charge	change account	1.3.1
chgrp	change group ownership of file	1.3.1, A-8
chmod	change file permissions	2.2.8, A-8
clear	clear the screen	4.2.3
cp	copy one file to another	2.2.7.2, A-8
cpa	invoke performance analyzer	5.6
csd	invoke source code debugger	5.3
csh	invoke C-shell	2.1.3, A-8
cxdb	invoke X-based CXdb debugger	5.1
du	check disk usage	1.3.2, A-8
echo	echo message to screen	2.1.2, A-8
ed	invoke general purpose editor	2.3.2, A-8
emacs	invoke emacs editor	2.3.3, A-8
ex	invoke general purpose editor (subset of vi)	2.3.2, A-8
fc	invoke FORTRAN compile and load	2.1
fsplit	split FORTRAN source into separate .f files	3.3, A-8
ftp	transfer files between machines	5.3, A-8
gprof	build call graph	5.5.1.2
grep	search a file for a character string	3.2.2, A-8
imsldoc	on-line documentation for IMSL	7.1, A-8
info	CONVEX on-line help utility	8.1.3, A-8
kill	stop a process	4.2.2
larcdoc	on-line documentation for LARCLIB	7.3, A-8

ln	create a link to another file	2.2.7.10, A-8
lpr	route file to line printer	4.2, A-8
ls	list file names in directory	2.2.7.1, A-8
mail	electronic mail	3.1, A-8
make	utility to maintain large codes	3.3, A-8
man	obtain on-line documentation for UNIX commands	8.1.1, A-8
maschgrp	change group owner of mass storage file	5.5.8, A-8
maschmod	change permissions for mass storage file	5.5.7, A-8
masget	retrieve a file from mass storage	5.5.2, A-8
masls	list files on mass storage	5.5.5, A-8
masmkdir	create a directory on mass storage	5.5.3, A-8
masmv	move or rename file on mass storage	5.5.6, A-8
masput	store a file on mass storage	5.5.1, A-8
masrm	remove a file from mass storage	5.5.4, A-8
masrmdir	remove a directory from mass storage	5.5.9, A-8
math	invoke Mathematica	6.1
mkdir	create a directory	2.2.7.8, A-8
more	filter to output only 1 screen of information at a time	3.2.1, A-8
mv	move file to new location or rename	2.2.7.5, A-8
nohup	allow background process to proceed after logout	4.2.1
notes	read information on various topics	8.3, A-8
nm	list external symbols by location in executable	5.1
passwd	change your password	4.1.1
pmd	invoke post-mortem dump	5.4
printenv	check on status of environment variables	4.1.2
prof	invoke profile by execution time	5.5.1.1
pwd	print current working directory	2.2.7.6, A-8
qdel	remove running or queued CXbatch job	4.3.1
qlimit	display CXbatch batch limits	4.3.1
qstat	display status of CXbatch jobs	4.3.1
qsub	submit a job to CXbatch	4.3.1
quota	check file quotas	1.3.2, A-8
rcp	copy files between networked computers	5.4, A-8
rlogin	connect to remote host	5.2, A-8
rm	remove a file	2.2.7.4, A-8
rmdir	remove a directory	2.2.7.9, A-8
rsh	execute single command on remote host	5.2, A-8
set	set C-shell variable	4.1.2
senenv	set environment variable	4.1.2
sh	invoke Bourne shell	2.1.3, A-8
telnet	connect to a remote host	5.1, A-8
time	check execution and wall time	5.4
tpmount	allocate a tape drive	1.2.1.1
tpunmount	deallocate a tape drive	1.2.1.1
vi	invoke full screen editor	2.3.1, A-8

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 1993	Technical Memorandum

4. TITLE AND SUBTITLE
CONVEX Mini Manual

5. FUNDING NUMBERS
505-90-53-02

6. AUTHOR(S) Geoffrey M. Tennille and Lona M. Howser

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23681-0001

8. PERFORMING ORGANIZATION REPORT NUMBER
CSCC Doc. CX-1f

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA TM-107564
(Revised)

11. SUPPLEMENTARY NOTES
Tennille and Howser: Langley Research Center, Hampton, Virginia

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Category 60

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This document briefly describes the use of the CONVEX computers that are an integral part of the Supercomputing Network Subsystem (SNS) of the Central Scientific Computing Complex of the Langley Research Center. Features of the CONVEX computers that are significantly different than the CRAY supercomputers are covered, including; FORTRAN, C, architecture of the CONVEX computers, the CONVEX environment, batch job submittal, debugging, performance analysis, utilities unique to CONVEX and documentation. This revision reflects the addition of the Applications Compiler and X-based debugger, CXdb.

The document is intended for all CONVEX users as a ready reference to frequently asked questions and to more detailed information contained with the vendor manuals. It is appropriate for both the novice and the experienced user.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Computing environment, FORTRAN C, debugging	51
	16. PRICE CODE: A03

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std Z39-18
298-102