Chapter 10: Accessing Software Products

In this chapter you will learn how to get information about the software products that are provided and supported by the Computing Division. We describe how to access products already installed on your system, and how to obtain a product from the Fermilab `KITS` area and install it on your system.

Appendix B discusses the Fermilab UNIX Product Support structure, **UPS**. We recommend that you read it first to get a good basic understanding of how products are handled. At a minimum, you can refer to it to clarify the terms and concepts used in this chapter.

The information in this chapter and Appendix B has been taken from *UNIX Product Methodology at Fermilab: Guide to Using UPS v3 and UPD v2 for Product Maintenance, Installation, Distribution and Development* (GU0014), but covered here in much less detail, appropriate to the different audience. We refer you to that document if you need further information.

> **Notice of Upcoming Changes**
>
> **UPS** and **UPD** are currently undergoing redevelopment with a significantly different design. The new versions and accompanying documentation are due for release in the first half of 1998.

Off-site machines need to register for the product distribution services, **UPD** and anonymous **ftp**, discussed in sections 10.3 and 10.4, respectively. To do this, the responsible party for the machine needs to fill out the *UNIX Product Distribution Request Form* and mail it to *compdiv@fnal*. To find this form, go to the Computing Division home page, and under the heading **Services** click on *Forms*. Or look in the top level anonymous **ftp** directory (see section 10.4).

10.1 Finding Information about Available Software

The ever-changing list of software provided and/or supported by the Computing Division is maintained in a database accessible via the Web, as described in section 1.6.1. Here you can search for information on any **UPS** product, in particular the product name and number[1], an abstract describing the product, flavors for which it is available, and which versions are available in `KITS`. In addition, documents associated with the product are provided in HTML, PostScript, and/or ASCII text formats, as available.

An on-line report is available that lists the current version of all the **UPS** products. From the Computing Division home page, select *UNIX Applications* under **Documentation & Software**, and then select *Kits Report*.

1. This is an alphanumeric designation assigned by the Computing Division to identify each product and its associated documents.

10.2 Accessing Installed UPS Products

10.2.1 Get Information About Products Installed on Your System

First find out if multiple databases are defined on your system for your use by running the command:

% echo $PRODUCTS

Note the capital letters; *PRODUCTS* is an environment variable set by **UPS**.

If there is only one value, you can find out what **UPS** products are installed on your system with the command:

% ls $PRODUCTS

If the variable *PRODUCTS* is set to multiple values, you can check each directory (i.e. each database) separately.

To find the current version of a **UPS** product and its location, use the command:

% ups list [-a] [*product*]

More than one version, or instance, of a **UPS** product may be installed on your machine. If **-a** is specified, all instances of the product(s) are listed. If ***product*** is omitted, all **UPS** products are listed.

10.2.2 Setup a Product Instance

Each installed, declared **UPS** product instance requires that the **setup** command be issued prior to use (unless it is a dependent product of one that is already setup)[1]. At a minimum this command sets the environment variable *${PRODUCT}_DIR* (e.g., *WWW_DIR*, *EMACS_DIR*) to point to the root directory of the product instance. If the product has a setup file maintained in `${PRODUCT}_DIR/ups/` (most products do), **setup** also sources this file which makes the appropriate changes to your software environment in order to make the product available for use.

Typically users need to setup a chained instance of a product, for example the current instance on their machine. To setup the current instance of **www**, enter:

% setup [-c] www

The current instance is, in other words, the default (**-c** is unnecessary). To setup any other chained instance, include the chain flag (described in section B.10) in the command. For example, to setup the instance of **www** declared as test, enter:

% setup -t www

To setup an unchained instance of a product, include its version number. For example, to setup version 2_6a of **www**, enter:

% setup www v2_6a

1. There are exceptions to this rule; in particular, you should *never* setup the core FUE products **futil**, **fulib**, **funkern** and **shells**. **UPS** gets setup for you under FUE.

☞ We recommend that you *not* specify **UPS** products by their version in general, but instead use primarily the *current* version for production, the *test* version (or new or development) for testing, or an *old* version *only if necessary.* We cannot maintain all versions forever, and unless there are overwhelming reasons, old versions will be removed in a timely manner.

☞ Note that you can only have one instance of a product setup at a time. Each time you run **`setup`** on an additional instance of the same product, the previously active instance is automatically unsetup first.

Examples

`% setup prod1`

Sets up the current instance of product **prod1** and all of **prod1**'s use requirements.

`% setup -d prod1`

Sets up the instance of product **prod1** chained to development, and all of **prod1**'s use requirements.

`% setup -v -f IRIX prod1 v1_1`

Sets up the instance of **prod1** that has a version of v1_1 and a flavor of IRIX, and all of **prod1**'s use requirements. Sets verbose on.

10.2.3 Unsetup a Product Instance

`unsetup` sources the unsetup script, also maintained in `${PRODUCT}_DIR/ups/`. In general this undoes the changes to your software environment made by **`setup`** in order to make the product no longer available for use. You may need to unsetup a **UPS** product if you are short on environment variable space and want to get rid of extra environment variables, or shorten the *PATH* variable length.

When you no longer need to access a product (or its use requirements), simply type:

`% unsetup` ***product***

10.2.4 Invoke the Product

Normally **UPS** products are invoked by entering the product name on the command line, with any appropriate options and/or arguments, e.g.,

`% setup prod1`

`% prod1 [`*options*`]`

☞ Note that you must know the name of the application or its invoking command, **which in a few cases is different from the UPS product name used in the `setup` command**. If the product comprises multiple applications, you can usually get a listing of them by checking the product's `bin` directory. After you setup the product, run:

`% ls ${PRODUCT}_DIR/bin`

For example, say you want a tool to format a document in PostScript. First you need to setup the **UPS** product **psutils**, which includes all the available tools. Then run the command:

% ls $PSUTILS_DIR/bin

to see what your choices are:

```
a2ps         fixmacps     fixtpps      includeres   psmerge      showchar
epsffit      fixntps      fixwfwps     psbook       psnup
extractres   fixpsditps   fixwpps      pscover      psresize
fixdlsrps    fixpspps     fixwwps      psduplex     psselect
fixfmps      fixscribeps  getafm       pslabels     pstops
```

To invoke one of them, just enter the name of the executable on the command line. Be aware that for some products, files other than executables may also be included in `bin`.

10.3 Obtaining Products from KITS

The **UPD** menu interface was designed for easy distribution of software products, either directly via the network or by an intermediate tar file stored on disk or tape. Tar files of the currently available versions of **UPS** products for the supported UNIX flavors are maintained in the Fermilab `KITS` area on the node UPD.FNAL.GOV. **UPD** provides functionality to:

- list products in `KITS` or any other **UPS** product distribution node
- list the contents of a remote tar file
- download a product instance or copy an individual file from a remote tar file

UPD can also be used to:

- unwind the tar file once it's on the local node
- declare the product instance to the **UPS** database

As a user, you should be able to largely rely on your system administrators to install **UPS** products for you. Occasionally you may need a product that has not been installed on your machine.

If you want to make a **UPS** product available for general use on your system, you should have it added to your local system's **UPS** database. Contact your system administrator in this case.

If a product is for personal use, we recommend that you download the tar file to a temporary area using the **UPD** function provided, and unwind the retrieved tar file into your personal area. Be careful that you have enough disk space available.

Recall that off-site systems must register for **UPD**.

10.3.1 Steps for Installing a Product

Before installing any product:

1) Create at least the directory in which the product root directory will reside; you can create the product root directory itself, too, but it is not necessary.

2) (optional) Change to the product root directory or to the directory in which it will reside.

3) Run **`setup upd`**.

4) Run **`upd`** to invoke the **UPD** menu interface.

You should see the following main menu (version v2_8d of **UPD** was used here):

```
Welcome to the main menu interface for upd.
       Choose one of the following options:

        1       upd list: list available products
        2       upd copy: copy a product from a remote node
        3       upd unwind: unwind product into local directory
        4       upd tarcont: list the table of contents of the tar file
        5       upd extract: extract a file from tar file
        6       upd getreq: determine the build and use requirements
        7       ups declare: declare the product to UPS
        8       ups tailor: execute product's ups/tailor file
        9       fork a login shell

        q       quit
Enter choice :
```

Products are installed in a maximum of six steps, all of which can be accomplished via this menu:

1) List available products from the remote distribution node, and determine which product instance to install (option 1).

2) Copy the tar file associated with the chosen product instance to a temporary tar file on the local node (option 2).

3) Unwind the temporary tar file into its product root directory (option 3).

4) Declare the product to the **UPS** database (option 7), with or without a chain.

5) Tailor the product if **UPD** informed you that it is necessary (option 8).

6) Declare the product as "current" to **UPS** (option 7, again), if not already done.

☞ Note that **UPD** also allows you to download individual files from within a remote tar file. If, for example, one or two files of a previously installed product get corrupted, you can use UPD TARCONT to list the contents of the tar file and UPD EXTRACT to download an individual file from it.

A few notes on installing products:

- If additional actions are required by the installer, the product's `INSTALL_NOTE` is automatically typed to the screen during the unwind process to inform you about them.
- Very few products require tailoring. **UPD** informs you during the declaration if tailoring needs to be run.
- If you are using a version of **UPS** previous to v3_9, we do not recommend simultaneously declaring a product to the **UPS** database and declaring it current (steps 4 and 6). This is due to a slight design flaw that is corrected in later versions (described in GU0014).
- If the product has use requirements (you shouldn't need to worry about build requirements in most cases), you are informed of this fact during the unwind step. Use option 6 to find out what the requirements are. You will need to install each required product instance individually if it does not already exist on your system.
- According to your group's policy, communicate to the users of your system if you make a new product instance available for their use.

10.3.2 UPD Menu Interface Operations

Each step of the installation has a screen associated with it, similar to the sample screen below:

```
UPD LIST:   * implies value required
        *1      [upd.fnal.gov] remote node name
         2      [] product name
         3      [current] version or chain (e.g., v1_0 or current)
         4      [SunOS+5] flavor
         5      [] remote database, if different from default

         e      execute upd list
         q      quit and return to main menu
 Enter choice :
```

The screens each have a list of numbered parameters to set. Defaults are provided where possible, and you can change or remove them. If you exit **UPD** and then restart it, you will lose any values associated with the particular product instance you were working with. The option of forking a shell from within **UPD** is provided in order to avoid this problem.

To set a parameter, enter its number at the `Enter choice :` prompt, and press carriage return. You set the parameters individually in successive operations. The parameters labelled with an asterisk (`*`) require you to set a value. All other parameters are optional. When an optional parameter does not have a value, the screen function generally executes for all possible values.

Once you've set all the parameters that you want, enter **`e`** to execute the screen function. Press **`q`** to quit the screen and return to the main menu. If you're on the main menu, pressing **`q`** quits out of **UPD**.

10.4 Using Anonymous ftp to Download a Product

UPS products are also available via anonymous **ftp** from the host FTP.FNAL.GOV. We recommend that after copying over the product tar file using **ftp** you use **UPD** to install the product.

☞ If you have a local **UPS** database, we recommend distributing via **UPD** because it provides a better interface for declaring products once the tar file is copied. You can always invoke **UPD** to do this after downloading via **ftp**, of course.

10.4.1 Access Anonymous ftp

To access anonymous **ftp**, enter the command:

`% ftp ftp.fnal.gov`

You will get a response similar to the following, to which you enter **`anonymous`** at the `Name` prompt:

```
Connected to fsui01.fnal.gov.
220 fsui01 FTP server (Version wu-2.4(2) Thu Jul 20 12:03:52 CDT 1995) ready.
Name (ftp.fnal.gov:your_username): anonymous
```

Once that is accepted, you will be prompted to enter your email address as a password (e.g., `qjones@fnal.gov`), which is not echoed to the screen:

```
331 Guest login ok, send your complete e-mail address as password.
Password:
```

The screen will then display the following information[1]:

```
230-This is the FTP service for FTP.FNAL.GOV. Use of this
230-service is for authorized usage only.
230-
230-The purpose of this service is to provide access to the
230-following categories of files:
230- - The FermiTools area: publicly available software under the
230-   /pub directory (all users). Files available listed in ls-lR.Z.
230- - The UNIX KITS area: This provides alternate access to files
230-   normally accessed via the upd program. (registered users)
230-   Files available listed in index.
230-
230-This server allows for on-the-fly compression  and uncompression
230-of files retrieved. It also allows you to tar up entire directory
230-structures.
230-
230-You are registered to retrieve files from the UNIX kits area.
230-
230-Please read the file readme
230-  it was last modified on Wed Sep 27 16:47:32 1995 - 415 days ago
230 Guest login ok, access restrictions apply.
```

In the top level directory a `readme` file is provided that discusses some of the available tools and features.

If you are on an off-site node, it must be registered with Fermilab in order to use anonymous **ftp** to obtain tar files[2] from the `KITS` area. If you need to register, use the UNIX Product Distribution Registration Form in the top level directory under the filename `registration`. All machines in the `fnal.gov` domain are automatically registered.

1. The “last modified” information will change, of course.
2. Note, the FermiTools products are not subject to this restriction; you do not need to register your node to obtain tar files from the FermiTools `/pub` area.

Here is a top-level directory listing for reference:

```
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
total 66
dr-xr-xr-x   8 ftp      root        1024 Nov 19 22:00 .
dr-xr-xr-x   8 ftp      root        1024 Nov 19 22:00 ..
drwxrwxr-x   7 ftp      995          512 Oct 31 11:21 .incoming
drwxr-xr-x  20 kits     sys         1024 Nov 19 22:00 KITS
lrwxrwxrwx   1 root     other          7 Jul 27  1995 bin -> usr/bin
dr-xr-xr-x   2 root     other        512 Jul 27  1995 dev
dr-xr-xr-x   2 root     other        512 Apr 24  1996 etc
drwx------   2 root     root        8192 Jul 27  1995 lost+found
-rw-r--r--   1 root     other      11824 Nov 19 22:00 ls-lR.Z
lrwxrwxrwx   1 root     other          8 Sep 15  1995 pub -> KITS/pub
-rw-r--r--   1 root     other       1566 Sep 27  1995 readme
-rw-r--r--   1 root     other       1937 Mar 26  1996 registration
dr-xr-xr-x   5 root     other        512 Apr 24  1996 usr
226 Transfer complete.
827 bytes received in 0.17 seconds (4.6 Kbytes/s)
```

Most **UPS** products can be found under the `KITS` directory. Access to this directory is restricted to registered users. Directly under `KITS` is a directory for each supported OS type, under which the product instance tar files appropriate to each one are maintained. The `declared` directory containing the product declaration files is also available under `KITS`. The public **ftp** area `KITS/pub`, available to all (registered and nonregistered) users, includes the Fermilab Software Tools (FermiTools). This collection of software is intended to provide the internet community with many of the Fermilab-developed software packages that we believe have general value to other application domains.

To see the contents of `KITS`, run the following (the output has been abbreviated for this manual):

```
ftp> cd KITS
250 CWD command successful.
ftp> dir
drwxr-xr-x 102 kits     sys         2048 Oct 29 17:31 AIX
drwxr-xr-x  53 kits     sys         1024 Nov 19 10:53 GENERIC_UNIX
drwxr-xr-x  25 kits     sys          512 Aug 15 13:57 HP-UX
drwxr-xr-x 153 kits     upd         2560 Oct 29 17:38 IRIX
drwxr-xr-x   2 kits     sys          512 Aug 27 14:47 LINUX
drwxr-xr-x  84 kits     upd         1536 Nov  6 13:29 OSF1
drwxr-xr-x 111 kits     sys         2048 Oct 29 17:46 SunOS
drwxr-xr-x  41 kits     sys         1024 Aug 15 13:56 ULTRIX
drwxr-xr-x   2 kits     sys         3584 Nov 19 12:08 declared
drwxrwsr-x  22 bin      frmtools     512 Nov 14 22:29 pub
```

10.4.2 Select a Product Instance Tar File

From the root directory (`/`), **`cd`** to the appropriate OS type subdirectory under `KITS` (`SunOS` is used as an example here). You'll need to "get" the `index` file since you can't list a remote file from **ftp**. Since `index` is an ASCII file, make sure you set the mode to ASCII first, as shown below:

```
ftp> cd KITS/SunOS
250-A complete listing of files available under this directory
250-structure is available in the index file.
250-
250 CWD command successful.
ftp> ascii
200 Type set to A.
ftp> get
(remote-file) index
(local-file) index_SunOS
200 PORT command successful.
150 Opening ASCII mode data connection for index (74193 bytes).
226 Transfer complete.
local: index_SunOS remote: index
76388 bytes received in 0.89 seconds (84 Kbytes/s)
```

Now you have a local (renamed) copy. Use the **ftp** command **`!`** to run **`less`** as a shell command to list the file contents:

`ftp> !less index_SunOS`

You'll find that the file lists everything under `KITS`; all directories, subdirectories and tar files.

10.4.3 Copy the Tar File

Any file that is not a straight text file must be transferred in binary mode. Set the mode to binary (**`bin`**) and then use the standard **ftp** **`get`** function to copy the tar file. Refer to section 13.1.1 for information on standard **ftp** commands.

After the tar file is downloaded, use **UPD** to unwind it into the appropriate location on your system and declare it, as described in section 10.3.1.

Chapter 11: Editors

Several text editors are available at Fermilab. In this chapter we present our view of the advantages and disadvantages of the available editors, and we provide some basic information on the setup and use of each one. You will learn how to invoke each editor, and how to create, edit, and save a file in each one using a small subset of commands and features. We include only minimal usage information for the VMS-style editors.

11.1 The Available Editors

vi — A native UNIX screen editor. It is not generally considered to be one of the better available editors, however knowing the basics is useful for two reasons: 1) you may occasionally encounter an application that throws you into a **vi** session, from which you'll at least want to exit, and 2) **vi** is the one editor you are guaranteed to find on all UNIX machines[1].

emacs — A powerful modern public-domain screen editor available for both VMS and UNIX. You will find **emacs** installed on most UNIX machines.

xemacs — An incarnation of the advanced, self-documenting, customizable, extensible real-time display editor **emacs**. It provides many powerful display and user-interface capabilities not found in **emacs**.

NEdit — A popular and intuitive X-based editor written at Fermilab for UNIX.

The remaining available editors are UNIX implementations of Digital's products for VMS:

nu/TPU — An emulation of TPU

fermitpu — A locally ported UNIX version of the newest EVE, layered on **nu/TPU** (from VMS V6.1)

EDT+ — A UNIX version of EDT[2]

The VMS-style editors are legacy products, and as such are not guaranteed to remain available beyond the VMS-to-UNIX migration period. It is also unlikely that you will find these third party editors on UNIX systems outside of Fermilab. They are provided solely to ease the transition to

---

1. In addition, UNIX provides **sed** for non-interactive editing and **awk** for more sophisticated and complex non-interactive editing. These two products are not covered in this manual.
2. Due to budgetary constraints and soaring maintenance costs, support for **EDT** under UNIX is now frozen at release v6_3a. We expect that this release will continue to function properly. However, if future hardware or software changes cause **EDT** to break, it will not be replaced or upgraded. **EDT** will *not* be purchased for any new operating systems which may be supported in the future.

UNIX for users already familiar with these products on VMS. **Users not already familiar with these VMS editors are strongly discouraged from choosing one of them for use in UNIX!** *All* users are encouraged to begin learning one of the permanent UNIX editors.

11.2 Comparison of Editors

The following table discusses the advantages and disadvantages of each of the above-mentioned editors.

Editor	Advantages	Disadvantages
vi	Available on all supported platforms and systems as a native UNIX tool. Well-documented. It is set as the default editor for many applications on many systems.	Considered to be very primitive in its screen capabilities. Non-intuitive interface. Lowest common denominator of screen editors.
emacs	Widely available public domain software package. Well documented. Many macros and enhancements available via the internet. Language-specific text editing modes (e.g., English, Lisp, C, FORTRAN). Lots of expertise and consulting available via newsgroups, etc. Very configurable. **EDT**-style keypad emulation available. X and ASCII modes. Supports file versions.	Requires installation of **emacs**. Interface not very intuitive for users accustomed to VMS-style editors.
Xemacs	Same as for **emacs**, plus more. Intuitive, GUI interface. Language-specific syntax-highlighting. Multi-windowed, interactive, object-oriented class browser.	Requires installation of **Xemacs**.
NEdit	Very intuitive and well-documented. Customizable. Gaining popularity world-wide. X-based.	Requires installation of **NEdit**. Requires X-based terminal. Not available on all systems/platforms.
nu/TPU	Nearly identical to Digital's **TPU** for people familiar with **TPU** programming. Sufficient for a wide variety of simple editing tasks. Close enough to **EVE/EDT**-style keypad to be familiar to VMS users. Does not exhibit screen-repainting problem of **fermitpu** (see below).	Duration of availability is uncertain. Implementation includes many faults. Limited documentation available on-line. Uses old version of **EVE** (from VMS V5.0). Does not match the interface that VMS users are used to. Not X-based. Requires VT-100 or VT-200 compatible keyboard mappings.

Editor	Advantages	Disadvantages
fermitpu	Nearly identical to Digital's **EVE** of VMS V6.1 with the exception that the **EDT**-style keypad is used by default. Licensing is handled on the local node (i.e., no network license server problems). User-configurable and extendable. Does not require X.	Duration of availability is uncertain. Not available/installed on all systems/platforms. Not the same as **fermitpu** on the central facility VMS systems, although similar. Requires the FUE environment and the installation of **nu/TPU** and the **fermitpu** layered products. Not X-based. Requires VT-100 or VT-200 compatible keyboard mappings. On some platforms, the screen is repainted after every keystroke which is very slow.
EDT+	Very similar to an enhanced **EDT** for people familiar with **EDT** programming. Incorporates a powerful word processor, disaster recovery system, an extensive help facility, full feature programmable text processing and the GOLD-KEY style of editing. Does not require X, although X interface is available. Minimum learning curve for users familiar with Digital's **EDT**.	Duration of availability is uncertain; maintenance contract terminated. Requires the FUE environment and the installation of the **EDT** product. Not available/installed on all systems/platforms. Distributed licensing (implies that network problems and/or problems with the license server could prevent **EDT+** from being usable on occasion). Not X-based. Requires VT-100 or VT-200 compatible keyboard mappings.

☞ Note: For all editors used with X display, the *DISPLAY* variable (see section 9.2) must be set properly.

11.3 Getting Started with the Editors

In this section we present a few important commands for each editor. The minimal information necessary for you to edit and save a simple file is provided. All these editors have many commands and sophisticated features that we do not cover here.

11.3.1 vi

As we mentioned in section 11.1 above, you may occasionally find yourself thrown into the **vi** editor unexpectedly, and you will certainly want to know how to exit, if nothing else. For that reason alone you should become familiar with a few **vi** commands. As is typical in UNIX, the **vi** commands are case sensitive.

vi requires no setup. Invoke it using the command:

`% vi [`*`filename`*`]`

Once the file is opened, you are in *command mode*. From this mode you can issue commands, move the cursor, and invoke insert mode.

To enter *insert mode*, type `i`. Text you type will be inserted *before* the cursor. From insert mode you can enter new text in the file. Press the Escape key to exit insert mode and return to command mode. On many keyboards the Escape key is labelled; if not, the sequence <Ctrl-[> is mapped to the escape function.

Some useful vi commands available in command mode

`h,j,k,l`	move cursor left, down, up, right, respectively
`H`	move to top line of screen
`L`	move to bottom line of screen
<Ctrl-f>, <Ctrl-b>	scroll forward, backward one screen
/pattern	search for ***pattern***
`/`	repeat search in forward direction
`x`	delete current cursor position
`X`	delete back one character
`dw`	delete current word
`dd`	delete current line
***n*`dd`**	delete ***n*** lines starting with current
`p`	insert (**p**aste) last deleted text after cursor
`:r` ***filename***	read in contents of ***filename*** after cursor
`:x`	quit **vi**, writing file only if changes were made
`:w`	write file, do not quit
`:w` ***file***	save copy to ***file***, do not quit
`:q!`	quit file, discarding edits

Most UNIX guides contain a complete description of **vi**.

11.3.2 emacs and xemacs

We'll cover these two editors in the same section because although they are separate products, they are closely related.

emacs is a popular editor available on the net. It can be invoked in windows mode or ASCII mode. Many UNIX books cover **emacs**, and a good reference for **GNU emacs** is *Learning GNU Emacs* (O'Reilly & Associates). We also refer you to an on-line GNU EMACS manual accessible under **Editors** on the *UNIX Resources* Web page. Further documentation can be found from the man pages.

xemacs is a graphical, X window implementation of **emacs**, with a few extra bells and whistles. (It is somewhat fancier than **emacs** in windows mode, and not to be confused with it.) **xemacs** is very similar in appearance and operation to many PC text processing applications for Windows. It is menu/mouse driven, with keyboard shortcuts available. **xemacs** has commands for passing single command lines to shell processes; it can also run a shell interactively. All standard **emacs** keyboard commands can be used in **xemacs** instead of/in addition to the pull-down menu commands. An **xemacs** manual is accessible under **Editors** on the *UNIX Resources* Web page.

Setup and Invoke emacs

To use **emacs**, the product needs to be installed. To set it up, include in your login script or enter:

```
% setup emacs
```

The mode (X or ASCII) in which **emacs** attempts to start-up is determined according to your *DISPLAY* variable. To invoke, type:

% emacs [*options*] [*filename*]

To invoke it in ASCII mode without a new window using an X terminal, type:

% emacs -nw [*options*] [*filename*]

Setup and Invoke xemacs

To use **xemacs**, the product needs to be installed. To set it up, include in your login script or enter:

```
% setup xemacs
```

To invoke **xemacs**, type:

% xemacs [*filename*] [&]

Help Facilities

emacs/xemacs has an extensive interactive help facility, but the facility assumes that you know how to manipulate **emacs** windows and buffers. <Ctrl-h> enters the Help facility. <Ctrl-h>-t enters the help tutorial, which can teach beginners the fundamentals of **emacs** in a few minutes. <Ctrl-h>-a enters Help Apropos, to help you find commands by function. For **emacs** in windows mode, there is also a Help menu. <Ctrl-h>-i enters the Info facility which brings up the on-line documentation browsing system. The initial page (the Directory node) gives a menu of major topics. The information is presented in a hierarchical tree format. **xemacs** provides this via an Info button.

Keyboard Commands

emacs/xemacs commands use the *Control key* and the *Meta key*[1]. In the following list **C-** indicates that the control key is pressed at the same time as the character that follows. Similarly, **M-** indicates the use of the Meta key, although it's not necessary to keep the Meta key pressed down while typing the next character. Note that some command sequences use multiple keystrokes, with and without the Control and Meta keys. A sequence like **C-x u** means hold down control while you press **x**, then just press **u**. Following is a list of the **emacs** commands used most often:

C-h	enter on-line help system
C-h i	enter the information browser which provides a menu of major topics (use tab and enter keys or 2nd mouse button to select a topic; navigation information is provided)
C-p, **C-n**, **C-f**, **C-b**	move up (to **p**revious), down (to **n**ext), **f**orward, or **b**ackward by one line or character, respectively
C-v, M-v	move forward, backward, by one screen
C-s	search forward for characters (system will prompt you for string). To continue search, type **C-s** again.
C-r	search backward

1. If you have a key labelled Meta, use it; if you don't, try the Alt key or the Escape key (if you're running a native X window). As a last resort, the sequence <Ctrl-[> should always work as a Meta key.

C-d	delete a character
C-k	delete (kill) from cursor to end of line
C-y	restore what you've deleted
C-x u	undo last edit
C-g	get out of current command operation
C-x i	insert file at cursor position (system will prompt for filename)
M-q	fill paragraphs
C-@ or **C-spacebar**	set the mark for the start or end of a region to select
C-w	delete all between mark (see **C-@**) and cursor's current position (paste back with **C-y**)
M-w	copy all between mark (see **C-@**) and cursor's current position (paste back with **C-y**)
C-x C-x	exchange mark and cursor's current position (since the mark is invisible, this allows you to find it)
C-x C-s	save the file
C-x C-w	write to file (system will prompt for filename)
C-x C-b	display buffer list
C-x o	move cursor to other window (when more than one displayed)
C-x C-c	exit emacs

Note, if the serial port or terminal device you are typing on is configured for <Ctrl-s>/<Ctrl-q> flow control, you may find that <Ctrl-s> (written above as **C-s**) within **emacs** causes the terminal to stop sending characters, the same as when used at the shell prompt. If you want to use the usual **emacs** key bindings and to have **C-s** work properly within **emacs**, you'll need to reconfigure your line to not do flow control. Where and how you do this depends on how you're connected. If you're connected via a modem, you may need to reconfigure your modem, as well as the pseudo-terminal on your UNIX host. The latter can be done via the command:

```
% stty stop undef start undef
```

(which sets the stop and start characters to "undefined"). You could include this statement in your `.login` or `.profile`. The intermediate step, between the on-site modem and the Cisco router/terminal server, has flow control turned off by default.

Language-Specific Text Editing Environments

emacs/xemacs supports several text editing environments (called modes), each geared to a particular language (e.g., English, Lisp, C, FORTRAN). When the editor is started, it normally loads the file `$HOME/.emacs`, if present, which contains Lisp commands for initialization. In particular, by setting up a mapping in this file between file extensions and languages, you can configure **emacs/xemacs** to come up in the appropriate mode according to the extension of the file you specify on the invoking command line. The text you need to include in `.emacs` has the syntax:

```
(setq   auto-mode-alist (append '(
    ("\\.extension1$" . language1-mode)      ("\\.extension2$"   . language2-mode)
    ("\\.extension3$" . language3-mode)      ("\\.extension4$"   . language4-mode)
    ...
      ) auto-mode-alist))
```

For example:

```
(setq   auto-mode-alist (append `(
    (“\\.asm$” . asm-mode)      (“\\.s$”   . asm-mode)  (“\\.awk$” . awk-mode)
    (“\\.cc$”  . c++-mode)      (“\\.C$”   . cc-mode)   (“\\.hh$”  . c++-mode)
    (“\\.c$”   . c-mode)        (“\\.h$”   . c-mode)    (“\\.i$”   . c-mode)
    (“\\.m$”   . objc-mode)     (“\\.csh$” . c-mode)
    (“\\.cdf$” . fortran-mode)  (“\\.cin$” . fortran-mode)
    (“\\.for$” . fortran-mode)  (“\\.f$”   . fortran-mode) (“\\.F$” . fortran-mode)
    (“\\.inc$” . fortran-mode)  (“\\.car$” . fortran-mode)
    (“\\.cra$” . fortran-mode)  (“\\.crb$” . fortran-mode)
    (“\\.tex$” . TeX-mode)      (“\\.txi$” . Texinfo-mode)
    (“\\.el$”  . emacs-lisp-mode) (“\\.icc$”  . c++-mode)
      ) auto-mode-alist))
```

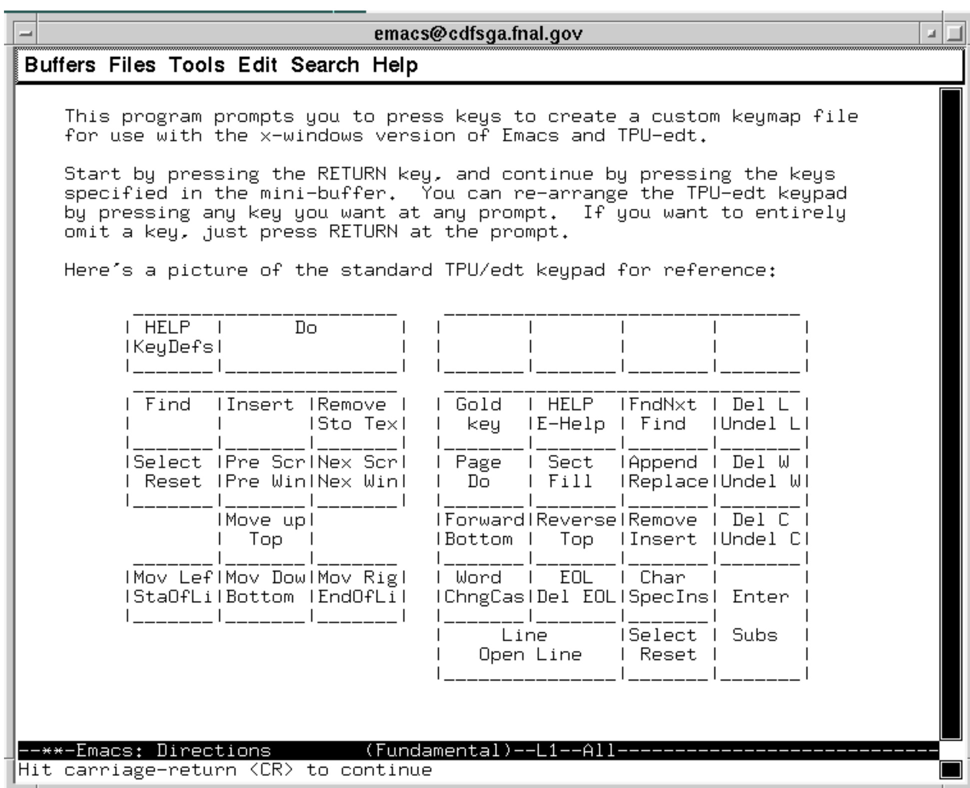
The on-line GNU EMACS manual provides more information on creating and modifying this file.

Use with EDT-Style Keypad

emacs/xemacs can be set to have an **EDT**-style keypad. This is documented more thoroughly in our Web pages; here we provide some start-up information.

emacs (windows) and **xemacs** — The key bindings are in the control of the editor emulation. When you first start up the **tpu-edt** emulator (instructions follow) it will prompt you to setup a keyboard mapping (see screen below). You need a separate mapping for each different keyboard type you use.

emacs in non-windows mode — Here you are at the mercy of your window emulator. There are things you can do to remap keys on most vt100 window emulators, but it is different for each OS/emulator. There are too many permutations to document.



To invoke the emulation, put the following text at the top of your `$HOME/.emacs` file:

```
(tpu-edt)              ;; Basic Emulation
(tpu-set-scroll-margins "10%" "15%") ;; Set scroll margins 10% (top) and 15% (bottom).
(load "vt-control" t)   ;; VT terminal controls (No complaint if not available)
;;
;; TPU-edt treats words like EDT; here's how to add word separators.
;; Note that backslash (\) and double quote (") are quoted with '\'.
(tpu-add-word-separators "]\\[-_,.\"=+()'/*#:!&;$")
```

To try this out without changing your `.emacs` file, first invoke the editor, then press Alt–**x** (hold down Alt while pressing **x**), and type **tpu-edt** followed by a carriage return.

On some platforms you may have trouble with the Gold key. The problem and its solution are dependent on the type of terminal and keyboard you are using. We know the solutions for some combinations. We are planning to provide this information soon in a Web page under *UNIX Resources*, and to keep it updated as we learn more. You may also find some helpful information under the heading X-terminals from the CD home page; see *Misc. Information - Hints, Notes, User Experiences, Etc.*

The xemacs GUI Interface

Due to the user-friendly nature of the product, we present only a few basic commands to give you a flavor of this type of editor if you are not familiar with it:

Open a file	Open from the toolbar or File menu (you can also open in another window or in a new frame); choose an existing file from popup window
Create a new file	Open from the toolbar or File menu; type in the new file name
Include a file	Insert File from the File menu
Select text	Use the mouse
Highlight special syntax in color (for use with language modes)	Syntax Highlighting from Options menu.
Cut/Copy/Paste text	Cut/Copy/Paste from the toolbar or Edit menu
Search for text	Search from the Edit menu
Spell check	Spell from the toolbar or Edit menu.
Save the file	Save from the File menu
Close the file	Delete Buffer from the File menu (use the Buffer menu to select the buffer to close)
Exit **xemacs**	Exit EMacs from the File menu

All standard **emacs** commands can also be used. In particular, if you call up one of the bells and whistles that you didn't really want, use:

C–g	get out of current command operation

The xemacs oo-Browser

The OO-Browser is a multi-windowed, interactive, object-oriented class browser. It currently supports seven object-oriented languages (Eiffel, C++, Objective-C, CLOS (Lisp), Java, Python and Smalltalk), one non-object-oriented language (C), and one documentation language, (GNU Info). On the Web you can find documentation for the oo-Browser with the **xemacs** product.

Before using the browser, you must create a database of information for each set of source code files you plan to use.

Here is a brief set of instructions for creating a database:

1) Run **`setup xemacs`**

2) Change to the directory in which you want the browser database output to go (we'll call it `outputdir`), and invoke **xemacs**.

3) Select OO-Browser from the Tools pull down menu. In the prompting window at the bottom of the window it will say: `Load/Create OO-Browser Environment: {outputdir}/`

   Enter a filename. This file is used to store the answers to the following questions. The application will then read this file to build your browsing environment. The file can be reused in future sessions. Here we'll use the filename `OOBR`, to create the file `{outputdir}/OOBR`.

4) Next it prompts for a language: `Choose: 1) C++/C; 2) Eiffel; 3) Info; 4) Java; 5) Lisp; 6) Obj-C; 7) Python; 8) Smalltalk`

   Enter the number corresponding to your choice.

5) Some error messages may rapidly scroll by. Ignore them[1] and wait for the following prompt: `Please specify the "OOBR" Environment (Hit RET to begin).`

   Enter Return as requested.

6) Next it prompts for a list of "system" directories and then "library" directories. You can specify each directory using an absolute or a relative path name (relative to your current working directory). Specify your own source code files as "system", and any library source code files you need as "library". Terminate the list by entering a carriage return on a fresh line.

7) The next prompt is: `Build Environment from spec in file, "{outputdir}/OOBR"? (y or n)`

   Enter **`y`**

8) The final prompt is: `Build Environment in a background process? (y or n)`

   Enter **`n`** (in order to monitor what happens)

The oo-Browser starts scanning all of the files in the directory tree underneath the directories you specified. When it finishes, your database is made and you are ready to start browsing.

A couple of useful keys are f and v. f displays an expanded (full) summary of the member functions belonging to the selected class. If you enter v with the cursor on a class name, the source file that defines the class (usually a header file) is displayed for viewing. Enter e to display it for editing. If you enter v with the cursor on a member function name, it displays the source code for that member function (usually an implementation file). If there are many functions in the same file, the browser places you at the correct line number for the selected function.

1. A knowledgeable source suspects that these messages are the result of a bug and will go away in a future release. She has heretofore ignored them with no ill effects.

11.3.3 NEdit

NEdit is very similar in appearance and operation to many PC text processing applications for Windows. It is menu/mouse driven, with keyboard shortcuts available. Some UNIX shell commands are available from within the editor. Make sure the **NEdit** product is installed on your system. To set up **NEdit**, include in your login script or enter:

`% setup nedit`

To invoke **NEdit**, type:

`% nedit [`*filename***`]`**

Due to the user-friendly nature of the product, we present only a few basic commands to give you a flavor of this type of editor if you are not familiar with it:

Open a file	Open from the File menu; choose an existing file from popup window
Create a new file	New from the File menu
Include a file	Include from the File menu
Select text	Use the mouse, or the shift and arrow keys together
Cut/Paste text	Cut/Paste from the Edit menu
Search for text	Find from the Search menu
Fill paragraph	Fill Paragraph from the Edit menu
Spell check	spell from the Shell menu
Save the file	Save from the File menu
Close the file	Close from the File menu (you are prompted about saving)
Exit **NEdit**	Exit from the File menu

Further information is available from the man pages and a plain text document in the distribution kit, but the on-line help in the program is complete and more convenient. The documentation is also available on the Web in the product documentation area.

11.3.4 nu/TPU

nu/TPU is an excellent port of Digital's TPU (Text Processing Utility) programming language to UNIX. The major component in **nu/TPU** is software that can be used to build new text processors and batch-oriented text manipulation routines. **nu/TPU** is distributed with two editing interfaces:

- SI (Simple Interface), an **EDT**-style keypad map. This should be sufficient for users who do minimal editing and wish to use an **EDT**-style keypad.
- a clone of the old **EVE** interface (from VMS V5.0) complete with the EVE_functions (**but not the EVE$functions!**) from that version. This will probably not be satisfactory to users familiar with the later version of **EVE** (VMS V6.1), since it lacks many of the enhancements and fault corrections.

The product **tpu** needs to be installed prior to use. To set up **nu/TPU,** include in your login script or enter:

`% setup tpu`

To edit using the SI interface by default, type:

`% tpu [`*filename***`]`**

To edit using the older **EVE** interface, enter:

```
% tpu –section=tpusec.ini [filename]
```

Once you are in the editor, on-line help is available via the PF2 key or the Help command.

For complete documentation on **nu/TPU**, including documentation on all command-line options and all **nu/TPU** programming statements, you can order the *nu/TPU Reference Manual.* Look under **Editors** on the *UNIX Resource*s Web page for documentation on the **TPU-edt** editor for GNU Emacs.

11.3.5 fermitpu

The **fermitpu** product is layered upon **nu/TPU** and contains a locally developed port of Digital's **EVE** editing interface from VMS V6.1 (the version of VMS which is current during the VMS to UNIX migration). Note that this is not the same as the VMS product **fermitpu**, which contains many enhancements beyond the VMS V6.1 **EVE** editor. Users who are familiar with programming in **EVE** and **TPU** will probably be most comfortable with this editing interface.

The products **fermitpu** and **nu/TPU** must be installed prior to use. To invoke, include in your login script or enter:

```
% setup fermitpu
```

```
% tpu [options] [filename]
```

Once you are in **fermitpu**, the on-line help (accessible via PF2 or the Help command) contains documentation on all of the EVE callbacks and TPU built-ins.

Look under **Editors** on the *UNIX Resource*s Web page for documentation on the **TPU-edt** editor for GNU Emacs.

11.3.6 EDT+

Support for **EDT** under UNIX is now frozen at release v6_3a.

This is a clone of Digital's **EDT** which includes the enhancement to remove the hard-coded 24-line display limit. The product **edt** needs to be installed. To setup **EDT+**, include in your login script or enter:

```
% setup edt
```

To invoke for editing within the same terminal window, type:

```
% edt [options] [filename]
```

To edit in a separate display window (requires X), type:

```
% xedt [options] [filename]
```

EDT+ allows both line mode and full screen editing. Full screen editing employs a keypad, which is located on the right-hand side of your keyboard. If you are in line mode when you first enter **EDT+**, simply type `c` to change to full screen mode. Once in full screen mode, you can start entering text, and your keypad is available for commands.

To return to line mode, enter <Ctrl-z>. At the prompt (`*`) you can type `exit` to exit and save, or `quit` which exits without saving the file updates.

Remember that when you're at the UNIX command prompt <Ctrl-z> is used as the suspend character.

Further documentation is available from the man pages, and from the vendor Boston Business Computing, Ltd.

Look under **Editors** on the *UNIX Resource*s Web page for documentation on the **EDT** emulation package for GNU Emacs.

Chapter 12: UNIX Mail Systems

This chapter describes how mail forwarding is managed at Fermilab and discusses the UNIX mail handlers that are currently available and supported.

Notice of Upcoming Changes

Electronic mail is an area that is currently undergoing rapid growth and development, and it is important that we take advantage of new technologies that are becoming available. In particular, demand has been growing at Fermilab for a distributed mail system that allows users to access their mail from any machine in the network, including a variety of UNIX and non-UNIX platforms.

Of the available distributed mail technologies, IMAP (Interactive Mail Access Protocol) is the one that offers the most flexibility and greatest performance. It is a client-server mail protocol designed to permit manipulation of remote mailboxes as if they were local. A particular advantage it offers over its competitors is access to remote saved-message *folders*. We are currently investigating mail handlers that support IMAP, and will announce changes within the next few months regarding the mail handlers that will be available to Fermilab UNIX users.

As of the release date of this manual, we are providing and supporting the same mail handlers as previously documented, namely **MH** (including **exmh** and **mh**) and **pine**. We plan to continue providing them after IMAP is implemented. However, new users should think about whether they will eventually want to move to an IMAP-based mail handler *before* choosing **MH** or **pine**. They should also be aware of a couple of disadvantages associated with the **MH** products:

MH does not and never will support IMAP. Furthermore, the **MH** suite of products is no longer under development, at least partly because its underlying structure doesn't support any of the emerging distributed mail technologies. The future of **MH** is therefore uncertain. It is unlikely that we will continue to build more versions of **MH** for OS upgrades once the new mail recommendation is made. On the other hand, users who have already configured their mail in accordance with **MH** should be able to continue using **exmh** or **mh** as long as it works, albeit without the advantages of a distributed mail system.

New users may want to choose **pine** instead, since its native structure does support IMAP. The disadvantage to **pine** is, of course, its non-GUI, single-window user interface. If you plan to move over to an IMAP-based mail handler when it becomes available, and you choose **pine** in the interim, we recommend that you *not* configure **pine** to store messages in **MH** format.

Since **pine** is well documented elsewhere, this chapter provides only some start-up information in section 12.2.1 along with pointers to further information.

We cover the UNIX **MH** (Message Handling) products including both the standard line mode mail reader (referred to as **mh**), and a GUI version of **MH** called **exmh**. And we provide the information needed to get you up and running in **mh** or **exmh** as quickly and easily as possible, with a minimum of up-front configuration.

Appendix F contains information on how to customize **exmh** and **mh**, and how to take advantage of some of the less essential but very useful features. If you wish to do more in-depth customization of the **MH** products, we recommend the excellent reference book *MH and xmh - Email for Users & Programmers* by Jerry Peek, published by O'Reilly & Associates. An HTML version of this book can be accessed through the *UNIX Resources* Web page.

Appendix G provides a quick reference for the **mh** commands described in this chapter.

Appendix H contains instructions on moving your VMS mail folders to UNIX and converting them to the **MH** system format and organization.

You'll also find a brief discussion of **Berkeley mail** at the end of the chapter, only because it is a mail reader available on all UNIX machines.

12.1 Mail Forwarding

12.1.1 The Fermilab Mail Server: FNAL

First, a general word on mail handling at Fermilab. FNAL formerly referred to the central Fermilab VAX computing cluster, where most people at the lab had an account. As computing became more distributed, a single mail serving node was set up to handle the mail flow. This mail server was designated as FNAL, and the central VAX cluster was renamed to FNALV. FNAL will continue to function as the Fermilab mail server.

The central mail server provides a simple, uniform mail address for all users at the lab:

***username*`@fnal.gov`**

We strongly recommend that you take advantage of this capability and have all your mail routed through FNAL. It is your responsibility to choose a node for reading mail and to make sure your forwarding address on FNAL is set to this node. We step you through the forwarding process in section 12.1.3. However, we recommend that you work your way through the material in this chapter in the order it is presented, and change your forwarding address on FNAL only when you're ready to actually switch your mail activities to your chosen mail system on UNIX.

12.1.2 Forwarding on File-Sharing UNIX "Clusters"

Before getting started, you will need to choose a node for your mail activities. Once you do, forwarding mail to that particular node requires creating a simple `.forward` file in your `$HOME` directory. We show you how to do this in section 12.3 and again in F.1. If your chosen mail node is part of a UNIX "cluster" like FNALU on which a file-sharing system is installed (for example AFS or NFS), then creating a single `.forward` file in your common `$HOME` directory is sufficient for the entire cluster. The contents of this file indicates your chosen mail node within the cluster.

You might wonder why you need to choose a particular node in the case of a file-sharing cluster. The reason is that incoming mail on UNIX is sent to a non-shared area of *one* node; you can't change that, you can only specify *which* node you want used. From here the mail needs to be incorporated into the mail system of your choice; we cover this topic in sections 12.3.3, 12.4.3, and F.4. If you are working on a different node from the one receiving the incoming mail, you will be able to send mail, but you will only see previously incorporated mail; you will not receive (and cannot incorporate) new mail because it is inaccessible to this node.

12.1.3 Recommended Forwarding Procedure

Advantages

The procedure we describe here for setting your mail forwarding has three advantages:

- Every time you change your mail node, you only need to change the forwarding address on three nodes: FNAL, the old (or current) mail node, and the new mail node. If you have accounts on many systems, this is a timesaver.
- All Fermilab users have a standard, stable email address: `{username}@fnal.gov`.
- No mail is left sitting in the system mailbox (except as yet unincorporated mail on your chosen mail node).

Set Forwarding on your New Mail Node

As a first step, in your `$HOME` directory on your chosen UNIX mail node, create a `.forward` file using your preferred editor. Enter the following expression as the contents of this file:

```
\{username}@{node}.fnal.gov
```

where `{username}` is your login id on your chosen mail node, `{node}`. For example, user "fred" who wants to receive mail on the FSUI01 node of the FNALU cluster would enter the line:

```
\fred@fsui01.fnal.gov
```

This expression is an instruction to put incoming messages in this designated node's spool area. The initial backslash (\) is required to prevent looping.

Later, you may decide to set up mail notification and/or unattended autoincorporation on your chosen mail node. These options are discussed in Appendix F. If so, you will need to replace the above expression with a different one in the `.forward` file on this node (or cluster) in order to prevent the redirection of incoming messages to two or more places.

Set Forwarding on FNAL

As a second step, you need to log on to the mailserver node FNAL and use the menu-driven program to change the forwarding to your new mail node. If you don't know or don't remember your FNAL password, send mail to *compdiv@fnal* and request a new one.

Make sure you set forwarding here on FNAL to your new mail node BEFORE you change the forwarding on your old mail node to FNAL! Otherwise you get an infinite mail loop.

Once you're logged on to FNAL, you'll see the following menu:

```
                              VMS Menu System V3.0
User: USERNAME
Date: 11/10/95
Forwarding Address:   username@FNALV.fnal.gov

Fermilab Mail Server Menu
    F  Set Forwarding
    P  Set Password
    U  Unsubscribe from a LISTSERV
    X  Exit the Menu


Time: 12:33 PM
Select Option:
```

Select option `F Set Forwarding`. This brings you to the following screen:

```
     Please enter your new forwarding address below

      Supported addressing formats

      DECnet     Node::User-ID                        FNALV::JONES
      BITNET     User-ID@Node                         JONES@FNALV
      INTERNET   User-ID@Node.domain                  JONES@FNALV.FNAL.GOV
      Quickmail  User-ID.Mailcenter@QMGATE.FNAL.GOV  JONES.Mailcenter@QMGATE.FNAL.GOV



      Please do not enter quotes or standard prefixes (smtp%, JNET%, IN%...)


      Your current forwarding address is:     username@FNALV.FNAL.GOV


      Enter forwarding address [ ? for help]:
```

Set your new forwarding address here in the format:

`{username}@{node}.fnal.gov`

where `{username}` is your login id on your chosen mail node, `{node}`.

Set Forwarding on all Other Nodes (including old mail node)

On all other external systems where you might receive mail (excluding the cluster containing your new mail node), UNIX or otherwise, set your forwarding address to your FNAL address.

To do this on other UNIX systems, create a `.forward` file in your `$HOME` directory on each separate node or file-sharing cluster. The contents of this file should be a single line containing the expression:

`{username}@fnal.gov`

where `{username}` is your login id on FNAL, the mail server. Note that there is no initial backslash here.

For information on how to set forwarding on non-UNIX machines, consult your system-specific documentation.

Set your “Reply To” Address

In most mail handlers, when a user directly “replies” to a message that you have sent (as opposed to starting a new message in the normal fashion to send to you), the reply is sent by default to the address from which you sent the original message, i.e., to the address in the “from” field. At Fermilab, this is not desirable; we prefer that all messages, including replies, get addressed to `{username}@fnal.gov`. In order to accomplish this, the outgoing message must specify a “reply to” address, and the mail handler that receives the message should be configured to send replies to that address in preference to the “from” address.

For **pine**, you need to set some configuration parameters. We describe this under **Configuring pine** in section 12.2.1.

For the **MH** mail handler, you can set your “reply to” address in the file `components`. We describe how to do this in section F.2.2. When replying to a message, the default behavior is to use the address in the “reply to” field if it has been supplied, otherwise to use the “from” address.

12.2 Overview of Mail Systems Available at Fermilab

As is typical for UNIX, there are many mail handling systems from which to choose. Here we provide a brief introduction to the currently available mail handlers at Fermilab. The Computing Division currently provides both **pine** and the **MH** system (which includes **exmh** and **mh**). Please see the remarks at the beginning of this chapter regarding upcoming changes in available mail handlers.

The simpler, less functional **Berkeley mail** is a mail reader that you will find on all UNIX machines. Therefore we give a cursory treatment of it in this chapter, although we explicitly do not recommend it.

> First choose one UNIX node for your mail activities.
>
> Secondly, we advise you to become familiar with the different mail systems and select the one you want to use as your primary mail handler *before* you start your conversion process. Although **MH** and **pine** can be made to work together (see section 12.2.1), they were designed independently of each other, and by default organize and handle mail differently.

12.2.1 pine

pine (Program for Internet News and Email) is a popular menu-driven, non-graphic UNIX mail system which the Computing Division supports. It can be configured to be compatible with the **MH** message storage structure, although it does not come this way “out-of-the-box”.

The Computing Division no longer recommends using the **MH** message storage structure for **pine**.

We have not documented **pine**’s functionality in this manual; however we provide some start-up information for you. You will need to select from the wide variety of available documentation on the Web and elsewhere to learn how to use **pine**.

As a starting point for documentation on **pine**, see the *UNIX Resource*s Web page. You’ll find information under the **Mail** heading. In addition, also from the *UNIX Resources* page, select **Other UNIX Resources** followed by the *UNIX Reference Desk*, and click on *Applications* to find a good introductory document on **pine**.

Using pine and MH Interchangeably

This is no longer recommended; see the introductory remarks to this chapter.

If you have converted your mail to UNIX using the methods described in Appendices A and H, your mail folders are in **MH** format. See section 12.4.2 for more information on **MH**-style folders. If you plan to use **pine** and one of the **MH** interfaces interchangeably, you should leave your folders in **MH** format, and configure **pine** to recognize them. This involves creating appropriate mail folder collections. Follow these steps to make message storage in **pine** compatible with **MH**:

1) Run **`setup mh`** or **`setup exmh`** to create your **MH**-style `inbox` folder.

2) Run **`setup pine`**

3) Invoke **pine** (from your `$HOME` directory) using the command **`pine`**. This creates the **pine** configuration file `$HOME/.pinerc`.

4) Exit **pine** (type **`q`** to quit).

5) Make sure the product **mailtools** is setup, then run the command **`fmh2pine`**[1]. This modifies the **pine** configuration, setting your **pine** inbox path to the default **MH** `inbox` folder, and adding an **MH** folder-collection containing all your **MH** folders. This command also makes your nested **MH** folders available to **pine**.

6) Run **`pine`** again.

Type L to select a folder to view. When you expand the folder collection lists and view the folders, you will find that the contents of your **MH** `inbox` folder is included in `INBOX` in **pine**'s `<mail/[]>` folder collection. Your other **MH** folders show up in the `<#mh/[]>` folder collection with their original names. Any nested **MH** folders you have defined will show up in separate folder collections according to their "parent" folders.

Anytime you add or delete an **MH** folder, run **`fmh2pine`** to update **pine**'s configuration accordingly.

If you normally use **exmh** and only use **pine** from home, for example, include the commands **`setup mh`** and **`inc`** in your `.login` or `.profile` file so that new mail gets incorporated and is displayed in **pine**.

Configuring pine

We recommend that you configure a few additional options.

☞ **`fmh2pine`** sets `folder-collections` for you to the two values shown below thus configuring **pine** for use with **MH**. (This is no longer recommended.)

☞ In **pine**, messages are not by default moved from the system spool area on your mail node (where they are initially deposited) into your home area. See **The UNIX System Mailbox** under section 12.4.3 for a discussion of the spool area. We strongly recommend that you properly set the options that move messages into your home area (the variables `default-fcc` and `read-message-folder`, and the on/off feature `auto-move-read-msgs`). There are several reasons for this, one of which is that if your mail node goes down, you can not access any messages in the spool area.

Go to **pine**'s SETUP CONFIGURATION menu. There are many options available, and they come with on-line help. With the cursor on the option you want, press **`?`** to get help on it. Some options are variables which take one or more values. To set a value for one of these variables, first type **`A`** to add, then enter the value. Or type **`C`** to change an existing value. Other options are in the form of a list of features that you can turn on or off individually. Enter **`X`** to enable a feature; leave unwanted features blank. Here we provide some recommendations:

`default-fcc`	The default folder for messages you send out; make sure it is *not* set to (uppercase) `INBOX`. Default is `sent-mail`.
`read-message-folder`	The folder in which read messages are placed. Make sure it is set, and that it is *not* set to (uppercase) `INBOX`.
`user-domain=fnal.gov`	This value specifies the domain part (right-hand side) of your return address on outgoing email, thus setting your "reply to" address to `{username}@fnal.gov`. This value is also used as the default domain for email that you send to a local username.
`reply-always-uses-reply-to`	Set this option so that **pine** uses the "reply to" field, if present, when you reply to incoming messages.

1. **`fmh2pine`** is part of **mailtools v2_2**.

`folder-collections=mail/[]`

This enables your basic set of **pine** mail folders.

`folder-collections=#mh/[]`

This enables your **MH** set of mail folders (no longer recommended; see introductory remarks for this chapter).

`default-fcc=#mh/outbox` Here we are assuming that in **exmh** you send your copies of outgoing mail to the folder `outbox`. Set this to your default outgoing folder to make your copies of outgoing mail available to you when you get back to **exmh**.

The option list is rather lengthy, so we provide some further suggestions for you. We have found the following features to be particularly useful. Explanations for them are in the on-line help:

```
auto-move-read-msgs
auto-zoom-after-select
auto-unzoom-after-apply
customized-hdrs
default-composer-hdrs
enable-aggregate-command-set
enable-alternate-editor-cmd
enable-bounce-cmd
enable-flag-command
enable-full-headers-command
enable-jump-command
enable-suspend
enable-unix-pipe-cmd
enable-expanded-view-of-addressbooks
preserve-start-stop-characters
quell-user-lookup-in-passwd-file
```

Using pine Exclusively

If you choose to use **pine** exclusively, you will need your folders in the **pine** format. First, get ready to run **pine**:

1) Run **`setup pine`**

2) Invoke **pine** (from your `$HOME` directory) using the command **`pine`**.

3) Complete the configuration as described in the above sections, however do not run **`fmh2pine`** and do not set `inbox-path` to `#mh/inbox`.

You can copy entire folders between the **pine** and **MH** collections as follows:

1) Go to the `FOLDER INDEX` screen for the desired folder.

2) Enter **`;`** (semi-colon) to issue the **pine** Select command.

3) Enter **`A`** to select All messages.

4) Enter **`A`** to Apply the command.

5) Enter **`S`** for Save.

Use <Ctrl-n> or <Ctrl-p> as necessary to choose the destination folder collection, then enter the new destination folder name. Or just type in a folder collection and name. Respond `Y` to create the new folder.

To use **pine** exclusively, you'll want to move any folders in the `<#mh/[]>` folder collection to the `<mail/[]>` folder collection.

Printing from pine

You will find it useful to set up printing from **pine**.

1) Go to **pine**'s SETUP PRINTER screen (enter **`S`** followed by **`P`**).

2) Move down to the `Personally selected print command` option. Enter **`S`** to select this option.

3) Enter **`A`** to add a printer, then as prompted, enter the name of the printer (this name is for your reference only) and the full command for the printer. For example:

Printer:	***printer_name***
Command:	**`flpr -q`*****printer_name***

4) You may enter several printers. Enter **`S`** to select the printer you want to use.

5) Exit (**`E`**) and save changes.

12.2.2 MH Graphical Interface: exmh

exmh is the GUI implementation of **MH**, requiring the X environment. It is built with **Tcl/Tk**. It is quite customizable, yet for the most part satisfactory with a minimum of configuration. **exmh** provides interfaces to many other powerful tools including a spell-checker, a fast-search mechanism, **www**, and more. **exmh** can read and send MIME (Multipurpose Internet Mail Extensions) multimedia mail.[1] Note that **exmh** displays nonprinting characters as their ASCII hex value. For example, embedded line-feeds (often included in mail messages originating from a PC) are shown as `\x0d`.

☞ Although we used to recommend **exmh** for users who most often work in the X environment, we no longer recommend that new users choose it, due to its uncertain future.

12.2.3 MH Line-Mode Interface: mh

mh is the command line implementation of the **MH** system, and as such is suitable for occasional use (when X-capability is unavailable) by those whose primary mail handler is **exmh**. **pine** may also be used in this capacity (see section 12.2.1).

mh is a very rich, flexible, and configurable mail system on its own. **MH** commands are not a part of a monolithic mail environment; each command is a separate program and the commands are issued directly from a UNIX shell prompt. Therefore, all the features of UNIX shells (pipes, re-direction of I/O, and so on) work with **mh**. You can use the man pages to get information on the **MH** commands.

12.2.4 Berkeley Mail

Berkeley Mail is an older mail system, and it is available on all UNIX systems. We do not recommend using it, but like the **vi** editor, knowing a little about it may come in handy sometimes! See section 12.5 for information on basic commands.

1. MIME allows you to read and send image files, sound files, and other items that require encoding to be sent via the mail system. See section 9.6. The O'Reilly book referenced at the beginning of the chapter includes an introduction to MIME.

12.3 The exmh and mh Mail Handlers

In this section we provide a guide to the basic commands and operations in both **exmh** and **mh**. Be aware that many more features and functions are available in these products than you'll find documented here.

We want to remind you that both **mh** and **exmh** are highly configurable interfaces to the powerful **MH** system. You can customize your personal setup to your heart's content, however we emphasize that neither **mh** nor **exmh** *requires* tweaking and tailoring beyond a very manageable minimal amount. We provide you with the information necessary to set up these interfaces for effective use in three places:

- We show you how to change fonts and editors in the basic command guide that follows in this section.
- In section 12.4 you'll find information on signature lines, mail aliases, and customizing the **exmh** folder display. We also provide information here on using **exmh** functions to incorporate mail.
- In Appendix F we discuss several nonessential but commonly tailored items, including customized mail notification (F.1.2) and message headers (F.2).

Recall that **exmh** and **mh** use the same underlying commands, although they have very different user interfaces. We've organized this section by function, showing you how to perform each selected function in both **exmh** and **mh**. This should make it easier to switch back and forth between the two interfaces if you sometimes find yourself at a graphics-capable terminal, and sometimes at an ASCII-only terminal.

We recommend that you perform couple of initial steps before starting to practice. These steps relate to mail forwarding, and we recommend them at this stage because of the way incoming mail is stored under UNIX (mail storage is discussed in section 12.4.3).

- Choose your mail node, and plan to do your practicing there.
- Create a file named `.forward` in your `$HOME` directory on this node containing the single line:

```
\{username}@{node}.fnal.gov
```

where `{username}` is your login id on your chosen mail node, `{node}`. This ensures that your "practice mail" will be available to you on the right node if you should need it later.

If you have an account on another machine, why don't you send a message or two to yourself on your chosen mail node before getting started so you have some messages to read? This isn't necessary, you can always send messages to yourself on your mail node.

12.3.1 Run Setup and Invoke the Application

exmh

`% setup exmh`

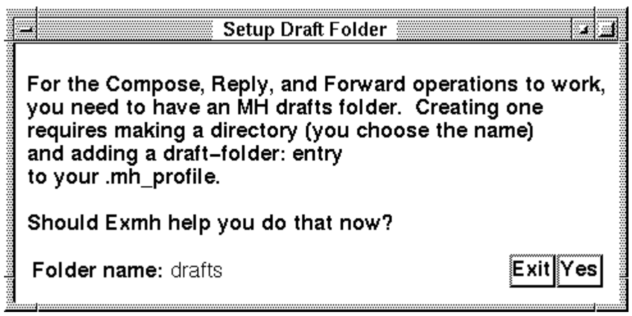
The use of **exmh** requires that you set your *DISPLAY* environment variable appropriately (see section 9.2).

When you setup **exmh**, you also setup several other products[1] which **exmh** either needs or with which it can interface. Among these is **mh**. If you've never run **`setup mh`**, then the first time you run **`setup exmh`** the file `.mh_profile` is created in your `$HOME` directory, and a message is sent to you (ignore the content of this message; it may be out of date). The `.mh_profile` file defines your basic **MH** configuration.

To invoke **exmh**, enter:

`% exmh [&]`

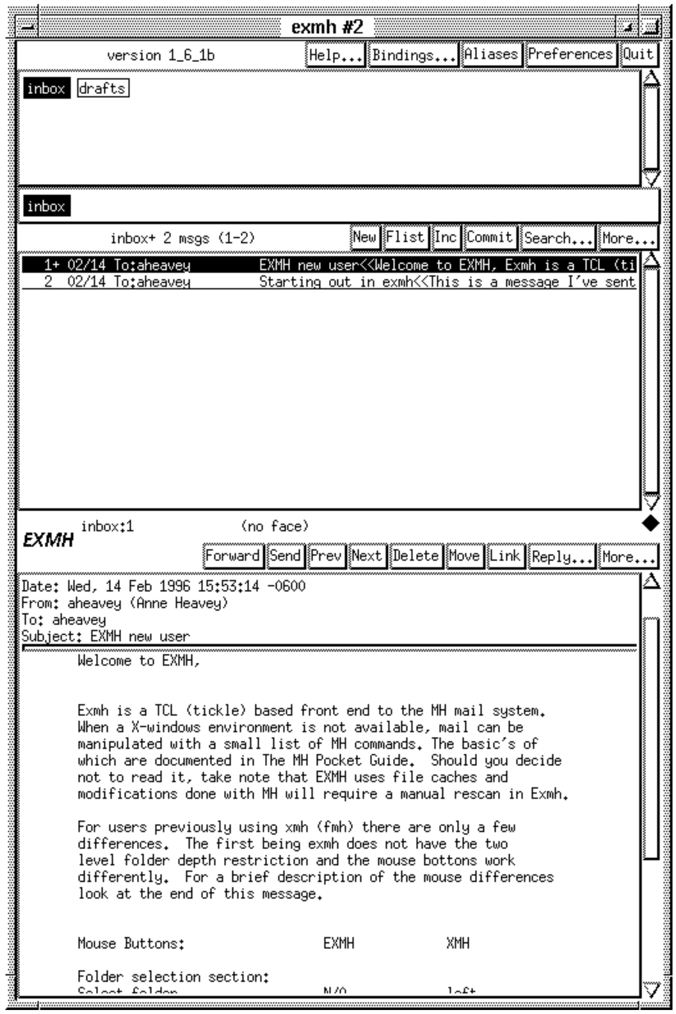
The first time you invoke **exmh**, you are asked if you want to create a message folder for message drafts in your `Mail` subdirectory. If you respond Yes, it adds a line to your `.mh_profile` identifying the folder.[2] By default the folder is called `drafts`, but you are given the opportunity to name it as you wish.



Click on Yes if you wish to continue.

1. Run the command **`ups list -l exmh`** to see the list of product dependencies.
2. It will also add the lines `unseen-sequence: unseen` and `repl:` to your `.mh_profile`.

Now an **exmh** window appears. The **exmh** window is split into three areas, which we also call windows. The title bars in each of the three windows have command or pull-down menu buttons. Actions are performed by clicking on the buttons and/or choosing from menu items.



The three windows have different functions:

Top	displays one "button" for every defined message folder. The current folder button is highlighted. Select a folder (make it current) by clicking on it with the left mouse button. **MH** message folders are discussed in section 12.4.2.
Middle	displays message headers in the current folder (the folder selected in the top box). The title bar for this area displays the current folder name and the number of messages. The header of the current message has a '+' next to its number and is highlighted.
Bottom	displays the current message (the message highlighted in the middle box). The title bar for this area displays the current folder name and the current message number.

A general usage note: You can display any message by clicking the message header in the middle window with the left mouse button to make it the current message. To select multiple headers (useful for refiling, removing, etc. several messages at a time), hold down the shift key while you press the left mouse button to select each one.

> You may find that on your terminal the default font size for **exmh** is rather small. If you really want to change it, here's how.
>
> You'll need to include two lines at the top of your `$HOME/.exmh-defaults` file (above the three comment lines ending with: `!!! Do not edit below here`). You will need to choose which font specifications to use. The format of these two lines is as follows (don't include the curly brackets, see example below):
>
> `*font: {new-font-specification}`
>
> `*fl_font: {new-folder-label-font-specification}`
>
> The `*font` line is for the text in the windows, and `*fl_font` is for the text in the folder label buttons. We recommend that you choose a fixed-width (as opposed to a proportionally spaced) font for text so that columns align properly in the middle (header) and bottom (message display) windows.
>
> The font specifications from which you can choose may be platform-specific. Execute the command **`xlsfonts | less`** to see what fonts are available for you[a]. Note that some of the font specifications will look short (for example `vt33014`); they are font aliases, and will work here, too.
>
> We suggest the following font specifications, if they are available on your system. Just add these two lines to your `.exmh-defaults` file as discussed above:
>
> ```
> *font: -b&h-lucidatypewriter-medium-r-normal-sans-14-100-100-100-m-80-iso8859-1
> *fl_font: -b&h-lucidabright-demibold-r-normal--14-100-100-100-p-84-iso8859-1
> ```

a. **xfontsel** is a point and click interface for selecting font names; you may want to use this in addition to **xlsfonts** to determine which fonts are available on your system.

mh

To setup **mh**, enter:

`% setup mh`

You don't need to do this if you've already setup **exmh**.

The first time you do this, it creates a file `.mh_profile` in your `$HOME` directory. It also automatically sends a message to you (ignore the content of this message; it may be out of date).

Now all the **mh** commands are available for use. As mentioned earlier, there is no single invoking command; you simply type **mh** commands at the shell prompt as you need them.

12.3.2 Compose and Send Messages

Remember to send yourself a message or two so you'll have something to read later on.

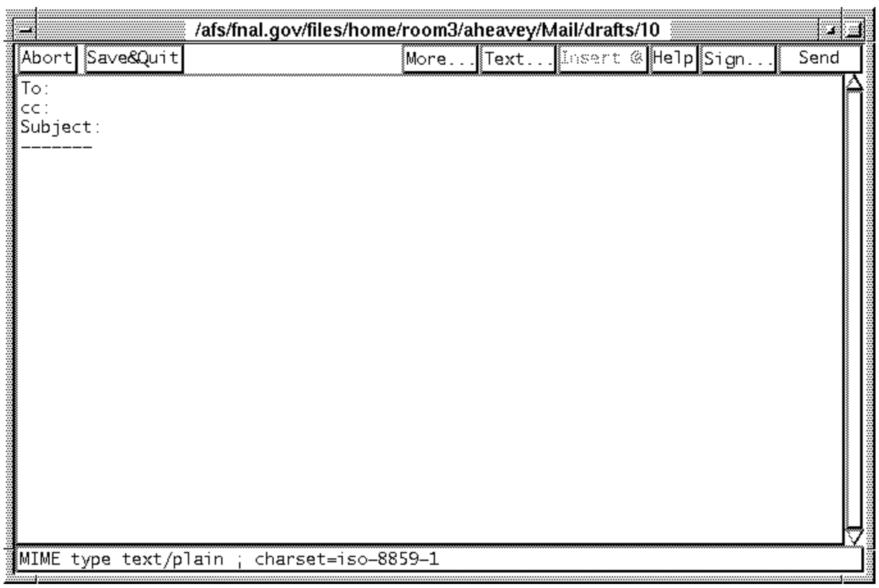
For both **exmh** and **mh**, a default message header with the following contents is provided for sending messages:

```
To:
cc:
Subject:
--------
```

If you wish to change the contents or appearance of the message header, you can create your own `components` file. See section F.2 for information.

exmh

To compose your message click on send in the bottom window. The word *comp* (for compose) will appear in the command menu bar. Then a new drafts window will pop up in which you will write your message. **exmh** uses **sedit** (Simple Editor) as the default editor for composing messages. As a new user, you may just want to use this editor for a while to simplify your initial setup.



Fill in the information, using a tab between the header fields[1], and then type in the body of your message below the dashes. On many terminals <Ctrl-h> is used for character deletion. Alternately, you can select the text to delete and press your Backspace or Delete key[2]. Most mail programs expect a carriage return at the end of each line of the message, so it is always wise to include it.

1. Note, however, that if you tab twice in rapid succession, the cursor will move down to the message content area. You can use this feature to pass over all remaining fields if you don't want to enter information in them.
2. If the backspace key doesn't work properly when sending mail, you can rebind it by first clicking on Bindings... in the top line, then click on the Simple Edit submenu. In the backspace field, type **`Key-Delete`**. Click on Save.

Note that if you set your preferences (go to Preferences button in top window) for Simple Editor to Format Mail Default *On*, you don't need to explicitly include the carriage return at the end of each line, it will be done automatically for you after you exit the editor. This feature is only available for the **sedit** editor.[1] Once you finish in the "Simple Editor Preferences" window, click Dismiss. Then click Save on the "Exmh Preferences" window.

Once you have completed the message, you can send it to the recipient(s) by clicking on the send button. If you decide not to send it, you can either save it in the `Mail/drafts` folder by clicking on Save&Quit, or discard the composition by choosing Abort. There are many useful options available on the drafts window under More... such as Insert File and Spell. Under Text... you have some options for fonts, which only are useful if the destination machine/mail reader supports them. If not, the recipient will see things like `<bold>{your message text}</bold>` in the message.

If you don't want to use this editor, you can define a different default editor, or you can define a secondary editor and then select it from the More menu in the compose window (this secondary editor option is only available with **sedit** as primary editor).

To define a different default editor, select Editor Support under the Preferences menu option in the top window. A pop-up window will appear. If your preferred editor "knows how to" create its own X window, e.g., **emacs**, **nedit**, enter in the Editor Command field:

`exmh-bg` ***editor***

where ***editor*** is the editor of your choice. If your preferred editor does not have this capability, e.g. **vi**, you'll need to enter:

`exmh-bg xterm [-`*options*`]-e` ***editor***

where ***–options*** are any options (i.e. font specification, preferred colors, geometry, etc.) to the **`xterm`** command. If you don't know about your editor, you'll find that one of these strings works and the other doesn't.[a]

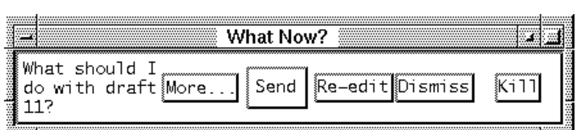
Your selected editor must be in your *PATH*; you cannot use an alias definition instead.

Click on Dismiss when you're done, and then be sure you click on Save in the "Exmh Preferences" menu.

a. If you have X-Authority, you can configure an editor using **`exmh-async`** ***editor***.

To send a message in which you want to include a file, compose the message, and use the "include file" command appropriate for the editor you're using.

When you save the composed message you are presented with a "What Now?" dialog box (a feature of **MH**) that gives you options to Send, Re-edit, Dismiss (this "dismisses" the pop-up window and leaves the message in the drafts folder), or Kill the message.



1. However, be aware that the line breaks inserted by **sedit** after you exit the editor may not be the places that *looked* like line breaks while you were in the editor. The line wrapping width is controlled by the value of Max Line Length in the Simple Editor Preferences window, not by the width of the editing window.

mh

mh will throw you into **vi** by default for composing a message. This is also true for forwarding and replying to messages. You can specify a different editor on the command line if you like.

To compose a message in **vi**, enter:

`% comp`	This brings up a **vi** session. Compose your message. Then save the message file, and exit the editor.

Remember to have the handy list of **vi** “survival commands” from section 11.3.1 with you! Just in case you get stuck, here’s your emergency exit: hit the Escape key (or <Ctrl-[>) and then enter:

`:x`	to save and exit, or
`:q!`	to quit without saving

Once you exit the editor, **MH**’s **What Now?** program is automatically invoked to prompt you for an action. At the `What Now?` prompt, you can enter a carriage return to get a list of options. Normally you will enter one of these options:

send	to send the message
quit	to save the message in `Mail/drafts` but not send it
quit -delete	to quit and not save.

To compose a message in a different editor, enter:

`% comp -e` *editor*	This brings up a session in your selected editor. This option will override any editor defaults. Compose your message. Then save the file, exit the editor, and respond to `What Now?`

If you *really* want to avoid **vi** at all costs, we’ll show you how to set a different default editor.

> To change your default editor for **mh**, add the following line to your `.mh_profile` file:
>
> `Editor:  {editorname}`
>
> where `{editorname}` is your preferred editor, for example:
>
> `Editor: emacs`

If you want to send a text file rather than a message, use the **mhmail** command. Here is an example to illustrate the syntax (more options are available, see the man page):

```
% mhmail joe@fnal -cc sue@fnal -subject "Here's the file!" < \
  filename
```

To send a message in which you want to include a file, compose the message, and use the “include file” command appropriate for the editor you’re using. In **vi** enter:

:r *filename*

In **emacs**, use [Ctrl-x] followed by **i**, then type in the filename at the prompt.

12.3.3 Incorporate and Read Incoming Messages

Mail incorporation refers to the process of moving incoming mail messages from the default mail holding area on your UNIX node into your personal area. Section 12.4.3 discusses this in more detail. You can’t read messages until you incorporate them. As messages get incorporated according to **MH** default standards, they go to the directory `$HOME/Mail/inbox (inbox` is called a folder), which was created for you during setup. Each message becomes a separate, numbered file in `inbox`. When you incorporate messages, you get all the messages in the spool; you cannot choose to incorporate some and not others.

exmh

There are a few ways to incorporate mail during an **exmh** session. You can incorporate your messages manually, or set options to incorporate mail either immediately upon invoking **exmh**, and/or periodically during your mail session.

To incorporate messages manually from within **exmh**, click inc from the middle window's menu. By default your spooled mail is placed in your `inbox` folder. You should now see the headers of the messages you sent yourself in the middle window. (If you didn't already send yourself some mail, go back and do it now.)

Once you incorporate your message(s), you can display any message by clicking the message header in the middle window with the left mouse button to make it the current message. The message displays in the bottom window.

To set up immediate or periodic incorporation during your **exmh** session, see section 12.4.3.

mh

First, incorporate the new message(s) into the **MH** system using the command:

% **inc**

This will display the headers of the incorporated messages, with a plus sign (+) next to the current message. The current message is the first unread one, usually the first one you just incorporated unless you had unread messages already in the folder. (If you didn't already send yourself some mail, go back and do it now.)

After incorporating, you can use **scan** to display the message header(s):

% **scan**	Display message headers from the current folder (since we haven't created any other folders, the current one is `inbox`).
% **scan last:*n***	Display the last ***n*** message headers from the current folder.
% **scan *m*–*n***	Display message headers ***m*** through ***n***, inclusive, from the current folder.

Here's our first example of using **mh** commands together with standard UNIX commands. To select message headers in `inbox` according to a ***pattern***, you could pipe **scan** to **grep** in this way (the **+inbox** specifically requests the `inbox` folder, and the **–i** option of **grep** indicates a case-insensitive search for ***pattern***):

% **scan +inbox | grep –i *pattern***

To read the current message, enter:

% **show [*n*]**	Display the current (or specified) message.

☞ Note that the **show** command can be used with more than one message number. For example **show *n*** can be expanded to **show *m n* ...** or to **show *m*–*n***.

Other commands to read messages are:

% **next**	Display the next message.
% **prev**	Go back and display the previous message.

12.3.4 Reply to Messages

The default reply header looks like this:

```
To:
cc:
Subject:
In-reply-to:  Your message of "Day, date time"
              <message-id>
--------
```

If you don't like the standard reply header, you can reformat it by creating your own `replcomps` file. See section F.2 for information.

When you want to include the original message in your reply, it is often convenient to precede each line of the original message text by a `>` symbol, to distinguish it from your text. FUE sets this up automatically for **exmh** and **mh**.

This formatting can be inconvenient if the recipient needs to work with the original text, and thus needs to remove all the `>`'s. You can avoid this by *forwarding* the message back to the original sender, instead of replying. Forwarding is covered in section 12.3.5.

If you want to change the format of your replies, you can create a file called `.mh_filter` in your `$HOME` directory. The O'Reilly **MH** book describes the different possible ways of setting this up. The contents of the default `.mh_filter` is:

```
body:component="> ",compwidth=0
```

exmh

Select the message to which you want to reply so that it displays in the message (bottom) window. Then click on the Reply button in the bottom window. You get a menu of options. Reply to sender sends it just to the sender, while Reply all sends it to everyone listed in `To:` and `Cc:`. To include the original message in the reply, choose one of the Reply menu options with /include.

Once you have made a selection from the Reply menu options, a composition window pops up. The header fields will be filled in. You can edit the header if needed. Move the cursor to the body area and type in your reply.

For **sedit**, click on the send button to send the reply. To skip the reply, click on Abort.

For other editors, save the message and exit the editor. Then respond to the "What Now?" pop-up menu.

mh

By default, the **`repl`** command uses the **vi** editor. To avoid this, either use the **`-e`** ***editor*** option or set a different default editor in your `.mh_profile`.

To reply to a message, enter:

`% repl [`*n*`]` Reply to the current (or specified) message. By default, everyone who got the original message gets a copy (the sender, plus everyone in `To:` and `cc:`).

`% repl_inc [`*n*`]` Like **`repl`**, but include the original message. (This FUE command replaces the standard **mh** command **`repl -filter`** *filter_file*.)

Either of these opens an editing session for the reply. As with **`comp`**, save your message and respond to the `What Now?` prompt.

The **-query** option used with either of these commands prompts about each recipient separately. To reply to selected addresses in the `To:` and `cc:` fields of the original message, enter:

% repl -query Reply to the current message. You will be prompted for each address in turn, to which you reply `y` or `n`, for example:

Reply to user1@FNAL.GOV? **y**

Reply to user2@xxx.yyy? **n**

Use the **-nocc** ***headerfield*** option with either command to prevent all addresses in a particular header field from receiving the reply; for example:

% repl -nocc cc [*n*]

Reply to the current (or specified) message, including all addresses except those in the original `cc` list.

12.3.5 Forward Messages

By default, forwarding uses a header file called `forwcomps` in the **MH** library directory. This file is similar to that used for the compose function (`components`). If you want your header for forwarding messages to be identical to that for sending messages, you can use the components file for both. Just include the following line in your `.mh_profile` file:

```
forw: -form components
```

If you want to customize the header for forwarding mail, you need to set up your own file called `forwcomps` in your `$HOME/Mail` directory. Edit it in the same manner as `components` (see section F.2).

exmh

To forward a copy of a message to someone, first select the message. You can select more than one message at a time. Then select Forward from the bottom window. A composition window comes up with a blank header and the message text. Fill in the header, and edit the message if you wish. Send or abort as explained earlier, according to your editor.

mh

By default, the **forw** command uses the **vi** editor. To avoid this, either use the **-e** ***editor*** option or set a different default editor in your `.mh_profile`.

To forward a message, enter:

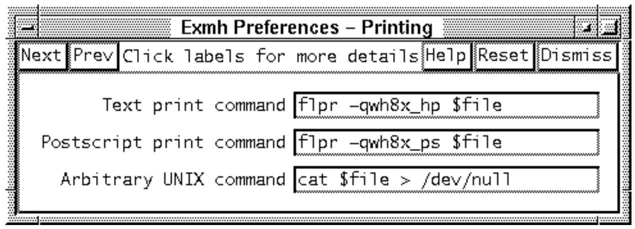
% forw [*n*] Forward the current message (or the specified message).

This opens an edit session. Fill in the header, and edit the message as you like. As with **comp** and **repl**, save your message and respond to the `What Now?` prompt.

12.3.6 Print Messages

exmh

First set your print queue by selecting Printing under Preferences in the top window. It should look like the following, with your own print queue substituted:



Click Dismiss when you've completed the change on the "Printing Preferences" window, then click Save on the "Exmh Preferences" window.

To print the current message, use the Print option from the More... menu in the bottom window.

To print several messages from a single folder, select them in the middle window, and use the Print option from the More... menu in the bottom window.

mh

To print a mail message, you use an **mh** display command piped to a UNIX print command (normally **`flpr`** under FUE). Any printing defaults you've set up for printing files will be used here (printing is covered in Chapter 8).

To print message number ***n*** on ***queue***, enter:

% **show** ***n*** **| flpr -q** *queue*

The option **`-showproc`** with the **`mhl`** argument formats the message a little differently so that it's easier to read (and nicer for printing), and pipes it to **more**. You would type:

% **show -showproc mhl** ***n*** **| flpr -q** *queue*

Especially good for printing multiple messages is the argument **`pr`** which separates mail messages onto successive pages:

% **show -showproc pr** ***m n*** **... | flpr -q** *queue*

12.3.7 Extract Messages

In **MH** format, each message is already a separate file, and can be manipulated as any other file. You may find however, that extracting messages is easier or quicker than standard UNIX file manipulation for some tasks.

exmh

To extract the current message to a file, use the Save to file option from the More... menu in the bottom window. A window pops up to allow you to choose the file to create or to which you want to append the message.

mh

To extract a message into a file use standard UNIX output redirection:

```
% show n > filename
```

This is essentially equivalent to the command:

```
% cp $HOME/Mail/folder/n filename
```

The show command includes one extra line that the **cp** command doesn't create:

```
(Message {folder}:{n})
```

12.3.8 Remove Messages

First, a note about message removal: When you use **rmm** in **mh**, or the delete button in **exmh**, the files are not actually deleted. They are renamed with a comma (**,**) or pound sign (#) in front so that **exmh/mh** doesn't recognize them as messages. The files are still there consuming disk space, and they don't go away unless YOU do something about them.

You can choose to periodically go through and clean out the old files that you no longer need in order to recover disk space. In **exmh**, use the Purge All Folders button (under the More menu in the middle window). This will delete the message files older than (default of) 7 days which you had previously "renamed". Or manually use the UNIX command **rm** on all files beginning with a comma (**,**) or pound sign (#).

A better solution is to add an entry to your `.mh_profile` changing the behavior of the Delete button /**rmm** command:

```
rmmproc: /bin/rm
```

This will really delete the message files when you use **rmm** (**mh**) or when you hit the Commit button after deleting messages (**exmh**). For clarity in the following text, we simply refer to deleting messages.

exmh

To delete one or more messages, first mark them for deletion by selecting the headers in the middle window then click on Delete from the bottom window. The message headers you selected will be highlighted; on some terminals they will be a different color or have a background pattern. If you are sure that you want to delete the marked messages, click on the Commit button in the middle window. If you change your mind before you commit, select the messages you do not want to delete and unmark them (UnMark is an option under More... in the bottom window).

☞ Note that after selecting a message, the cursor may move down to the next one, making it current rather than the one you selected. Make sure the cursor is placed correctly before unmarking a selection.

mh

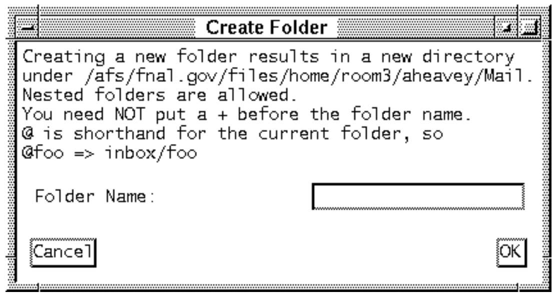
Use the **rmm** command to remove messages:

`% rmm [n m ...]`	Delete the current (or specified) message(s).

12.3.9 Create, Change and Remove Folders

exmh

To create a new folder from within **exmh**, select New from the middle window. The following window pops up for you to fill in. When you're done, click OK to create your new folder.



To switch current folders, click on the desired folder name in the top window with the left mouse button and the message headers from that folder will display in the middle window. It is now the current folder. You can read and work with the messages from this folder.

To view nested folders and their contents (see section 12.4.2), click the "parent" folder with the middle mouse button, then select a nested folder with the left mouse button.

To remove a folder, select it, then click on Delete folder under More... in the middle window.

mh

When you refer to a folder on the command line, precede it with a plus sign (**+**). To create a new folder, enter:

% **folder +*new_folder_name***

There are a few ways to reset the current folder to a different one. The most explicit command for this is:

% **folder +*new_current_folder***

Or, just running one of several commands with a different folder name changes the current folder until you again change it to something else. For example:

% **scan +*folder_name***	Display message headers from ***folder_name*** and set this folder to current.
% **show +*folder_name***	Display the current message in ***folder_name*** and set this folder to current.

Note that the **refile** command (see section 12.3.10) accepts folder name arguments but doesn't change the current folder.

To remove a folder, enter:

% **rmf +*folder***	Delete the specified folder.

12.3.10 Refile Messages

exmh

To refile one or more messages into a different folder, first select the header(s) in the middle window, then click the right mouse button on the destination folder in the top window. If they're moving to a nested folder, first click on its parent folder with the middle mouse button, and then on the nested folder with the right mouse button. The messages are now marked for refile. The message headers will be highlighted; on some terminals they will be a different color or have a background pattern. Click the Commit button (middle window) to complete the refile operation. If you change your mind before you commit, select the messages you do not want to refile and unmark them (UnMark is an option under More... in the bottom window).

☞ Note that after selecting a message, the cursor may move down to the next message header, making it current rather than the one you selected. Make sure the cursor is placed correctly before unmarking a selection.

mh

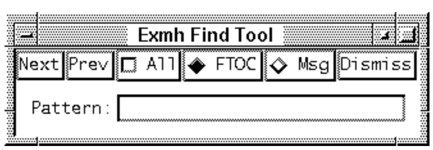
To move the current message (or specified messages) from the current folder to a different folder, enter:

% **refile +*folder* [*n m* ...]**

12.3.11 Search for Messages

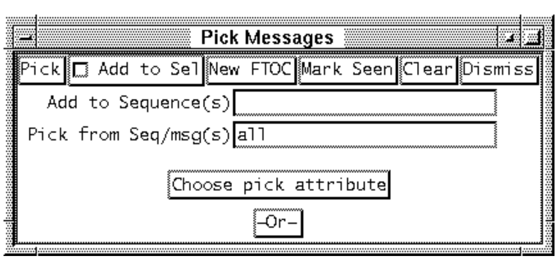
exmh

To search messages in a folder, first select the folder in the top window. Then select Search... from the middle window's menu and select the relevant option. You can get help on various items by looking into the search help. If you choose Find in table of contents, for example, you get a Find Tool window where you can type in a search pattern.



The headers of the "found" messages are highlighted. You can then display only the headers of these messages by choosing List only selected messages from the More... menu.

The Pick by Attributes from the Search... menu provides an interface to the MH **pick** command. This is a more sophisticated feature that allows you to set up complex logical search expressions. If you use the "before" or "after" search criteria, the date format accepted is `nn-mmm-yyyy`, for example `12-jan-1996`.



You have the option of saving the *sequence* of the picked messages. **MH** sequences are stored lists of message numbers. When you've finished setting up your search criteria, click Pick.

Another option you have is to use the **glimpse** utility, also available under the Search menu for full text searches. This requires building an index (use the menu option index under Glimpse full text), which may take a long time. For more information see the man pages for **glimpse**.

mh

`Pick` searches messages within a folder for the specified contents, and then identifies those messages. Two types of search primitives are available: pattern matching and date constraint operations. See the man pages for more information on **`pick`**.

`% pick -help`	Display a list of pick options.
`% pick` *options*	Select messages by content, date, sender, recipient, etc.

You can use **`pick`** to find messages sent from someone, sent to someone, on a given date, before or after a date, and even search every line of every message in the current folder. To tell **`pick`** to scan through every line of every message in the folder `recent_work` and return the message numbers with the word "quark" in them, enter:

```
% pick -search quark +recent_work
```

Here is an example showing the selection of messages within a date range:

```
% pick -after 1-dec-1995 -and -before 1-jan 1996
```

You can use **`pick`** with sequences, which are described above in the **exmh** portion. Commands that accept message numbers accept sequences as well; just type the sequence name. For example, find all messages received from ***sendername***, and store them in the sequence **`picked`**:

```
% pick -from sendername -sequence picked
```

The result also tells you how many messages are picked. You can then scan these messages by typing:

```
% scan picked
```

You can search for a string within the sequence by typing, for example:

```
% pick -subject 'quark substructure' -sequence picked
```

You can use **`pick`** and **`scan`** together (note the use of backquotes here):

```
% scan `pick -from sendername`
```

12.4 Basic Configuration for MH

12.4.1 Configuration Files

In order to take advantage of the variety of features provided by the **MH** package, several configuration files can be used for both **exmh** and **mh**. In this section we list all the files discussed in this chapter and in Appendix F, and provide brief descriptions for reference.

You're already familiar with two of them, `.mh_profile` and `.forward`.[1] See sections F.2.1 and F.1, respectively, for more information on these two files.

1. The `.forward` file is not specific to the **MH** system; it may be used with any mail handler.

`.mh_profile` defines your basic configuration (where mail is stored, the editor to use, various MH options, etc.). Running **`setup mh`** or **`setup exmh`** the first time automatically creates a default `.mh_profile` for you in your `$HOME` directory.

`.forward` stores your forwarding address. It is used in mail forwarding (routing), notification, and unattended autoincorporation. You must create it in your `$HOME` directory.[1]

The following three files allow you to customize your mail headers. See sections F.2.2, F.2.3, and F.2.4, respectively, for more information on them:

`components` defines message header format for sending mail. If you don't create your own copy in your `Mail` directory, the system default file in the **MH** library section will be used.

`replcomps` defines message header format for replies. If you don't create your own copy in your `Mail` directory, the system default file in the **MH** library section will be used.

`forwcomps` defines message header format for forwarded messages. If you don't create your own copy in your `Mail` directory or elect to use the `components` file header format, the system default file in the **MH** library section will be used.

In sections 12.3.4 and 12.4.5 we show you how to create:

`.mh_filter` allows you to include the original message in a reply. It goes in your `$HOME` directory.

`{alias file}` stores distribution lists and local aliases for long mail addresses (you define its name in your `.mh_profile`). It is assumed by default to be in your `Mail` subdirectory, but you can define it otherwise.

In Appendix F we provide information on yet a few more configuration files!

`scan-form` provides the format of display when you scan mail. The system default files in the **MH** library section will be used if you don't have your own in `$HOME/Mail`.

`inc-form` provides the format of display when you incorporate mail. The system default files in the **MH** library section will be used if you don't have your own in `$HOME/Mail`.

`.maildelivery` controls how local delivery is performed with manual or background incorporation in **exmh** when the presort option is selected. It also controls how delivery is performed when unattended automatic mail incorporation is defined in `.forward` using **slocal**. No default is provided; if you need this, you must create it in your `$HOME` directory.

12.4.2 MH Mail Folders

MH uses the concept of mail folders to store messages. Folders are simply subdirectories of your `Mail` directory which contain mail messages, one message to a file. To distinguish folders from files in **MH** commands, folders are prefixed with a plus sign (**+**) in a command.

1. If you only have an account on one machine (i.e. one node that is not part of a file-sharing cluster) besides FNAL and if you don't want automatic mail incorporation or notification, you don't need the `.forward` file.

For example:

```
% refile +admin
```

is an **MH** command that moves (refiles) the current message in the current folder to the folder `admin`.

Folders can be multi-level, for example `outbox/feb96`. Within a folder, the message filenames are incremented numbers starting at one (1). Running **`ls -l`** on a folder, you will see a numbered file for each message (in "alphabetical" order), for example:

```
total 296
-rw-r--r--   1 username  g020        339 Nov  9 11:36 1
-rw-r--r--   1 username  g020        864 Nov  9 11:36 10
-rw-r--r--   1 username  g020       3297 Nov  9 11:36 11
...
-rw-r--r--   1 username  g020        753 Nov  9 11:36 2
-rw-r--r--   1 username  g020       2628 Nov  9 11:36 20
```

The command **`ls -lt`** is convenient for listing them in reverse chronological order.

After a lot of refiling and deleting of messages, the messages in your folders may not be in order by date anymore, and you will undoubtedly have numerical gaps in the file names. To reorder and renumber the message files:

exmh use the Sort folder and Pack folder options under the More... button in the middle window

mh use the commands **`sortm`** and **`folder -pack`**, respectively

12.4.3 Incorporation of Incoming Mail into Folders

The UNIX System Mailbox

Due to the wide variety of mail readers on UNIX and the configurability provided by many of them, incoming mail is not by default delivered to any particular file or area in your home directory. Instead it is collected in a *system mailbox*, a directory defined as the mail holding area, on each node. In FUE, the environment variable *MAIL* is defined to reflect the system-specific location of your mailbox file, set to `/usr/spool/mail/{username}` or `/usr/mail/{username}`, depending on the UNIX flavor.[1] For example, user `fred` has his unread, unincorporated incoming mail stored in chronological order in the file `/usr/mail/fred`.

Your mail resides in this file until you incorporate it into a different file (mail folder). How and where it gets incorporated depends on the mail reader you decide to use. Here we discuss incorporation into the **MH** mail folder structure.

Options for Incorporating Mail

When new mail arrives, by default it resides the system mailbox until you incorporate it into your mail folder. There are a few ways to incorporate mail during an **exmh** session (i.e. "attended" incorporation), as we mentioned in section 12.3.3. You can incorporate messages manually, or set options to incorporate mail either immediately upon invoking **exmh**, and/or periodically during your mail session.

---

1. See the file `fermi.login` (for C shell family) or `fermi.profile` (Bourne shell family) in Appendix C. On a SunOS 5 (Solaris) system, you may have to set the variable *MAIL* manually due to a bug.

If you want to configure **exmh** so that it incorporates your mail at regular intervals and/or at particular times during your **exmh** session, first go to the Preferences menu in the top window.

- Choose the Incorporate Mail menu. Do you want to incorporate new mail when you invoke **exmh**? Check the box. Do you want to incorporate waiting mail when you click to open the **exmh** icon? Check the box. Dismiss this window and save your preferences.
- Choose the Background Processing menu. Do you want to incorporate new mail periodically? Check the Inc box. The period is set to 10 minutes by default, but you can change this value. Dismiss this window and save your preferences.

Would you like **exmh** to sort your mail for you, automatically putting some messages in a particular folder? If so, you need to do two things:

1) Create a `.maildelivery` file describing how you want your mail sorted (see section F.2.6).

2) Tell **exmh** to use this file. Go to the Preferences/Incorporate Mail menu, and under Ways to Inc, check the Presort box.

We discuss unattended automatic mail incorporation in section F.4. This is different from attended automatic incorporation in that mail is automatically incorporated whether or not you're running **exmh**, and whether or not you're even logged in. We do not recommend this method, although we include it here for completeness.

12.4.4 Signature Lines

To include a boilerplate signature to messages that you compose and send in **exmh** or **mh**, you need to create your own `$HOME/Mail/components` file, and simply add the signature to the bottom of this file, underneath the dashed line. For information on the `components` file, see section F.2.2. This file does not govern replying and may not govern forwarding (see section 12.3.5).

To add a boilerplate signature when replying to messages, you can add a signature to the end of your filter file, called `.mh_filter` in your `$HOME` directory (see section 12.3.4). This file gets used only when you're using the option to include the original text in the reply.

If you're using your own `$HOME/Mail/forwcomps` file for forwarding messages, include the signature lines at the end of this file. Note that your signature will appear *above* the text of the forwarded message.

12.4.5 Mail Aliases

You can send mail to many people at a time by defining an alias for the addresses of the recipients. You may also want to define aliases for long single addresses. Create a file (we call it `mh_alias` for example purposes; you can call it what you like) in your `$HOME/Mail` directory, and include an `Aliasfile` line like the following in your `.mh_profile`:

```
Aliasfile: mh_alias
```

This example alias file shows the required format:

```
d0spokes: mont@fnal.gov,\
          pgrannis@fnal.gov

cdfspokes: carithers@fnal.gov,\
       bellettini@fnal.gov
```

There cannot be any blanks after the backslash (\) in the aliases. To see your aliases you can use the command **`ali`**. For example if you type:

`% ali d0spokes`

you will get the response:

`mont@fnal.gov, pgrannis@fnal.gov`

12.4.6 Folder Order and Header Display (exmh)

Some further options available for **exmh** (but not for **mh**) to include in the `.mh_profile` file are:

```
Folder-Order: inbox folder2 folder3 * drafts
Header-Suppress: .*
Header-Display: Date To Cc From Subject
```

`Folder-Order` sets up the order of display of folders on your **exmh** window. Wildcards as used in filename expansion are allowed, so you only need to specify folders that you want displayed in particular places. In our example, we specify what folder to show first, second, third, and last. All the others will be sorted alphabetically between the third and the last.

`Header-Suppress` and `Header-Display` control which headers you see when you read a message. The example above indicates that everything is suppressed except the `Date:`, `To:`, `From:`, `Cc:`, and `Subject:` lines.

12.5 Berkeley Mail

In this section we discuss the simpler, less functional mail handler **Berkeley mail** because it is one that you will find on all UNIX machines. We describe how to read and send messages from the command line using the command **`mail`** or **`fermimail`**. In the Fermi files that run when you log in, the command **`mail`** is set to point to **Berkeley mail** by default (via the alias **`fermimail`**).

12.5.1 Send Messages and Files

To send a mail message, the command format is:

`% mail [-s` ***subject*****`]`** ***recipient*** **`[<`** ***filename*****`]`**

If you don't include the **`-s`** option with a ***subject*** (subject must be enclosed in quotes if it's more than one word), the system will prompt you for one after you hit carriage return. The ***recipient*** is a user name in standard format. If you want to send a file instead of composing a message, use the input redirection (**<**) with the ***filename*** (and path, as necessary). If you don't include a file, the cursor will be placed at the start of the first line under the command (no prompt) for you to start composing your message. Conclude the message by going to a new line and typing a period (**.**) followed by carriage return. You will be prompted for a `Cc:` before exiting. To abort a message that you are composing, use the standard *kill* control sequence (see section 2.4).

A very simple example command line using this format to compose a message is:

% mail butler@fnal

An example using the subject option and sending a file is:

% mail -s "Meeting minutes" butler@fnalb.fnal.gov < minutes

12.5.2 Read Messages

Assuming you have no autoincorporation set up, all of your unread incoming mail resides in the file designated by the variable *MAIL* (see section 12.4.3). When you log in, the system notifies you if this file contains any messages. It will not automatically notify you of incoming mail at other times during your session, unless you are in a mail reader.

To read your messages in the system mailbox, simply type:

% mail[1]

This puts you in command mode from which you can read messages. Enter **?** to get a list of commands. If there are messages, after you read them and when you are exiting mail, the system automatically sends the messages to your `$HOME/mbox` file, and deletes them from your system mailbox file. If there are no messages to read, the system will tell you so and exit.

For more information, see the man pages for **mail**.

1. Note that on some systems the command may be **mailx, Mail** or **fermimail**.

Chapter 13: Connecting to Remote Systems

Several utilities are available to enable you to transfer files between systems, to log into other systems on which you have an account, and to execute commands remotely. These features are described in this chapter.

Note that to use *any* of the communication methods with a VMS node, the VMS node must be running software to allow tcp/ip communication. All Fermilab VMS machines are licensed to run TGV's **MultiNet** software, which provides the necessary tcp/ip transport capability.

13.1 Transferring Files

13.1.1 ftp

ftp (File Transfer Protocol) can be used to transfer files to and from a remote system on which you have an account, and to manipulate the remote file system. The command format is:

`% ftp [-n] [-i] [`*host*`]`

If you specify ***host*** on the command line, you will be prompted for your (remote) username and password, and login will proceed. The option **`-n`** inhibits autologging on those systems on which it is enabled. The option **`-i`** eliminates prompting by the system, which can be tiresome when you're transferring many files.

Some commands available in **ftp** are described here; for a complete list, see the man pages for **ftp**.

`user` ***username password***	Identification for the remote system. If auto-log is enabled, you will be prompted automatically for the username and password. If you don't specify the password, it will prompt you.
`bin` or **`binary`**	Set transfer mode to binary image transfer.
`ascii`	Set transfer mode to ASCII (for text files).
`put` ***local_file*** [***remote_file***]	Transfer a local file to the remote system. If the remote filename is not specified, the local filename is used.
`mput` ***local_files*** [***remote_files***]	Similar to **`put`**, but you can use standard wildcard characters to transfer a series of files at a time.
`get` ***remote_file*** [***local_file***]	Transfer a remote file to the local system. If the local file name is not specified, it is given the name on the remote machine.
`mget` ***remote_files*** [***local_files***]	Similar to **`get`**, but you can use standard wildcard characters to transfer a series of files at a time.

help [*command*]	Display information about the meaning of ***command***. If no command is specified, **ftp** displays a list of the commands.
quit	Terminate the **ftp** session with the remote server and exit **ftp**.
open *host*	Establish connection with remote host. Needed if ***host*** was not specified on command line.
close	Close connection with remote host and return to **ftp** prompt. Doesn't exit **ftp**.
dir [*remote_directory*] [*local_file*]	Print listing of the directory contents of the remote directory and optionally put the output in ***local_file***.
cd *remote_directory*	Change working directory on the remote machine to ***remote_directory***.
lcd [*directory*]	Change working directory on the local machine. If no directory is specified, your home directory is used.
delete *remote_file*	Delete the file ***remote_file*** on the remote system.
! *shell_command*	Run a shell command without exiting **ftp**.

The following is an example **ftp** session where first one ASCII file, then a few binary files, are transferred from one UNIX machine to another. The username is unnecessary if the assumed one is correct. The password, as usual, is not displayed.

```
<fsui02> ftp cdfsga
Connected to cdfsga.fnal.gov.
220 cdfsga.fnal.gov FTP server ready.
Name (cdfsga:aheavey):
331 Password required for aheavey.
Password:
230 User aheavey logged in.
ftp> ascii
200 Type set to A.
ftp> put README
200 PORT command successful.
150 Opening ASCII mode data connection for 'README'.
226 Transfer complete.
local: README remote: README
121 bytes sent in 0.00091 seconds (1.3e+02 Kbytes/s)
ftp> bin
200 Type set to I.
ftp> mput appx*
mput appx_MH.ps? y
200 PORT command successful.
150 Opening BINARY mode data connection for 'appx_MH.ps'.
226 Transfer complete.
local: appx_MH.ps remote: appx_MH.ps
63902 bytes sent in 0.33 seconds (1.9e+02 Kbytes/s)
mput appx_awk.ps? y
200 PORT command successful.
150 Opening BINARY mode data connection for 'appx_awk.ps'.
226 Transfer complete.
local: appx_awk.ps remote: appx_awk.ps
56271 bytes sent in 0.24 seconds (2.2e+02 Kbytes/s)
mput appx_convrt.ps? n
mput appx_exmpl.ps? n
mput appx_impatient.ps? n
mput appx_login.ps? n
mput appx_ups.ps? y
200 PORT command successful.
150 Opening BINARY mode data connection for 'appx_ups.ps'.
226 Transfer complete.
local: appx_ups.ps remote: appx_ups.ps
107683 bytes sent in 1.3 seconds (83 Kbytes/s)
ftp> quit
221 Goodbye.
<fsui02>
```

Note that the CERN utility **zftp** (**z** for Zebra) is available for transferring FZ and RZ files (ntuple files) between systems[1]. To use **zftp**, it must be installed on the server system, and you need to set up CERN library during the session (**`setup cern`**).

13.1.2 rcp

Another way to transfer files is with the **rcp** utility (stands for **r**emote **c**o**p**y). You can use this command if both local and remote hosts support **rcp** protocol. The basic format is:

`%` **`rcp [`*options*`]`** ***file1 file2***

where ***file1*** and ***file2*** are the source and destination filenames, respectively; or

`%` **`rcp [`*options*`]`** ***file1 file2 ... directory***

where the files listed are to be copied to the directory. On a UNIX system the format of a remote directory or filename is ***hostname*:*path***, and the **MultiNet** implementation is consistent with VMS usage, namely ***hostname*::*path*** (note the double colon). If only filenames are given, they are interpreted relative to your home directory. Both the source and the destination may be on nodes other than the current machine.

The available options are:

`-p`	Give each copy the same modification times, access times, and modes as the original file.
`-r`	Copy each subtree rooted at ***file1***, ***file2***, etc. (for those that are directories); in this case the destination (last argument) *must* be a directory.

rcp does not prompt for passwords. Your current local user name must exist on the remote host and allow remote command execution by **rsh**. **rsh** is a utility that connects to a remote host and either executes a specified command or logs you on via **rlogin** (see section 13.3).

More information is available in the **rsh** and **rcp** man pages.

When you are using **rcp** between VMS and UNIX, you must protect special characters from local interpretation by enclosing them in quotes. An example command to copy a file `$HOME/prog1.c` from UNIX to the VMS node FNALV is:

`%` **`rcp prog1.c fnalv:"[.c]prog1.c"`**

Here is an example showing how to copy a file from remote UNIX node FNSG01 to the home directory on the VMS node (**`$`** represents the VMS prompt):

`$ rcp fnsg01::"/usr/products/bufio/v1_00/src" bufio.src`

If you are using the C shell family, **rcp** will not work if your `.cshrc` or `.login` on UNIX, or your `LOGIN.COM` file on a remote VMS host, executes interactive or output-generating commands. You should check for non-interactive access, and exit before such commands are executed, as is done for a UNIX system in the FUE default `.cshrc` file. The lines in `.cshrc` that check this are the following:

```
# Place any items that you want executed even for non-interactive use here
# Skip if not interactive shell
if ( $?USER == 0 || $?prompt == 0 ) exit
```

1. **zftp** is known to be likely to break in the AFS environment.

On a VMS system, check for **OTHER** (TCP/IP is not **NETWORK**):

```
$ IF (F$MODE () .EQS. "OTHER") THEN $ EXIT
```

Although you can specify a username and password with **rcp** (see the local man pages or **HELP**), you can also have a kind of "proxy login" by creating a special file (named `.rhosts`) on the remote machine containing the names of the systems and users on those system that should be allowed access.

13.1.3 The .rhosts File

To use many of the file transfer utilities, you need to set up what is known as a `.rhosts` file on the remote machine. This file must include a list of the names of the systems to be allowed remote access without login, and the usernames, where each system-username combination is separated by a space. If the username is left off, it is assumed to be the same as on the current system. If there are problems with name servers, you can be most assured that your address will be understood if you specify three lines for each connection requested:

1) one line with the complete Internet address,

2) one with the Internet number[1], and

3) one with the local name.

An example `.rhosts` file showing a single connection in the three specified formats:

```
fnsg01.fnal.gov
131.225.8.178
fnsg01 judy
```

When writing a `.rhosts` file on a UNIX host to allow access via remote commands (known as *r-commands*[2]) from a VMS cluster, you must include an entry for *each* cluster member and username from which you might access the UNIX host.

The following command line is an example of the command to copy a `FINGER.PLN` file from an account on FNALA to the file `.plan` in the same person's account on the UNIX node FNSG01. In order for this example to work, there must be a file named `.rhosts` on FNSG01 containing a line `fnala.fnal.gov`, and the username must be the same in both places.

```
$ RCP FINGER.PLN FNSG01::.PLAN
```

where **$** is the VMS prompt.

1. Use **nslookup** ***node*** to get the number. On VMS, you must enter **MULTINET NSLOOKUP** ***hostname***.
2. r-commands execute on remote hosts, and use `.rhosts` for authentication. They are variants of UNIX commands without the "r", and include for example **rsh** (**r**emote **sh**) and **rlogin** (**r**emote **login**).

13.2 Logging in to Other Systems

13.2.1 telnet

telnet is a remote terminal protocol which allows you to establish a connection to a login server at a remote site. **telnet** in the UNIX world is analogous to **`SET HOST`** in the VMS world. You can connect to other UNIX machines, or you can connect to VMS machines if the appropriate software is installed on the system (e.g., with **MultiNet** for which Fermilab has a site license). The format is:

`%` **`telnet [`*host*`]`**

If you include the host, you are prompted for the username[1] and password. If you do not include the host, you go into command mode. You can enter **`help`** at this point for help on the commands.

13.2.2 rlogin

Another way of logging into a remote computer from another system on the network is by use of the **rlogin** utility. One difference between **telnet** and **rlogin** is that if the communications package is configured for it and you create an appropriate `.rhosts` file (see section 13.1.3), you can log in to the remote system using **rlogin** without typing the password. The format from a UNIX machine is:

`%` **`rlogin`** ***rhost*** **`[-l`** ***username*****`]`**

where:

rhost	is the remote host to which you want to log in
username	is the username if the name is different on the remote system than the originating system

From a VMS machine, the format is:

`$ RLOGIN /USERNAME=`*username* *rhost*

If your system uses AFS and you experience permission problems after using **rlogin**, run **`pagsh`** followed by **`klog`**. Please see the discussion of these commands in section 7.3.1.

13.3 Executing Commands Remotely: rsh

You can execute commands on a remote system and have the output displayed on your terminal with **rsh**. **rsh** stands for **r**emote **sh**ell. In order for this to work, there must be an appropriate `.rhosts` file on the remote machine (see section 13.1.3).

The format is:

`%` **`rsh`** ***rhost*** **`[-l`** ***username*****`]`** ***command***

1. Some systems may not prompt you for a username; they will use your current login id. If you need to log in with a different username, enter a carriage return at the password prompt. Then the system will prompt you for a username.

where:

rhost	is the remote host on which the command is to be executed
username	is the username on the remote system if different than the originating system
command	is the command to be executed on the remote system

The command may be a single command or a shell script to be executed on the remote system. Metacharacters in the command must be quoted for proper interpretation on the remote machine.

The **MultiNet** implementation on VMS is:

$ RSHELL[/INPUT=*infile*][/OUTPUT=*outfile*][/USERNAME=*username*]- *hostname commandline*

The **MultiNet** implementation does not handle metacharacters like the UNIX implementation does.

The **/INPUT** and **/OUTPUT** options in the **MultiNet** implementation allow you to execute a task on a UNIX machine which reads and writes files on VMS.

If the command is omitted, you will be logged in to the remote host using **rlogin**.

Chapter 14: Batch Processing Environment

In this chapter we provide introductory information on **LSF** (Load Sharing Facility), the standard batch processing system at Fermilab, and on **fbatch**, the locally-written interface to **LSF**. We also list the related software components that can be used with **LSF/fbatch**.

You should be able to run and manipulate most batch jobs easily after reading this chapter.

14.1 The Standard Batch System at Fermilab: LSF

LSF, developed by Platform Computing, is a general purpose resource management system that unites a group of UNIX computers into a single system to make better use of the resources on a network. The single system is referred to as a cluster. **LSF** collects resource information from all nodes in the cluster, and uses it to allocate the available host machines for execution of batch jobs.

LSF distinguishes between client machines and server machines. A job can be submitted from either type, but run only on a server (a host). Under **LSF**, jobs that are run remotely behave just like jobs run on the local host. Even jobs with complicated terminal controls behave transparently to the user as if they were being run locally.

LSF is fully documented on-line; see the *LSF User's Guide*. To find it, select *Batch Services* from the Computing Division home, and look under *LSF v3.0 Documentation*. There is also a link to it in the Computing Division document database under **fbatch**.

For the purposes of this chapter, a batch job (also called simply a *job*) is any UNIX executable that is submitted to the **LSF** batch system. Job control information (e.g., name of executable, queue, required resources, and so on) is passed to **LSF** via command line arguments supplied when submitting a job.

14.1.1 Job Queues

Batch jobs are submitted to **LSF** via *job queues*. **LSF** administrators generally configure job queues to control host resource access according to user and application type. A queue can be defined to use a particular subset of the hosts in the **LSF** cluster; the default is to use all hosts.

Each queue represents a different job scheduling and control policy. Users select the job queue that best fits each job. All jobs submitted to the same queue share the same scheduling and control policy. There is a **nice** value associated with each queue (see section 5.5.1), and jobs submitted to a queue are automatically "re**nice**d" accordingly.

14.1.2 Load Monitoring on Hosts

LSF monitors the load of each host in the batch cluster by comparing the values of several built-in load indices against the allowable load thresholds defined by the **LSF** administrator. A load index is simply a measurement of the processing load on a batch host. On an overloaded host, batch jobs can begin interfering with each other or with interactive jobs. Therefore, **LSF** begins suspending jobs on a host when it becomes overloaded (i.e. when one or more load indices exceed the predefined *suspension threshold*). **LSF** resumes any suspended jobs once all the load indices read below the *release threshold*.

If a job queue has been defined with a time window (measured in real time), **LSF** suspends any jobs running on that queue when the current time falls outside of the window. These jobs get released when the time window reopens.

14.1.3 Host Selection

The resources available for processing **LSF** jobs on each host are defined by an **LSF** administrator. Only nodes having resources that match or exceed the resource requirements of a given job are potential hosts for that job. **LSF** compares the resource requirements specified for the job against the load on each of these nodes, and chooses the most favorable host.

If no resource requirements are specified for a job, a host of the same model and type as the machine on which the job was submitted is chosen.

14.1.4 Job Priority

LSF schedules, suspends, and releases submitted jobs by balancing job priority and available resources. Job priority is governed by several factors:

- the options and arguments specified on the command line during batch submission
- the priority of the queue on which the job was submitted; according to **LSF**'s FCFS (first come first serve) protocol
- the number of shares that a job has used, according to the FSS (Fair Share Scheduling) protocol. A *share* is a portion of the resources available on the host or hosts; queues may be defined to limit the number of shares jobs can utilize.

When a host's suspension threshold is reached, **LSF** suspends lower priority jobs first unless the scheduling policy associated with a particular job dictates otherwise. A suspended job can later be resumed by **LSF** if the host's release threshold is again reached (or, if the suspension was due to a time window, as mentioned above, the job resumes when the time window reopens).

☞ **LSF** does not override the UNIX scheduler.

14.2 Local Interface to LSF: fbatch

The **UPS** product **fbatch** supplies the commands that you enter to run and manipulate batch jobs. It is a set of locally-written shell scripts and C programs that provides a wrapper around **LSF**, and thus characterizes the batch processing environment on a cluster. **fbatch** is installed on many Fermilab systems, including FNALU. **fbatch** supports all the **LSF** batch functionality, and provides in addition:

- a consistent interface to **LSF** commands, regardless of UNIX flavor
- supplementary commands not directly supported by **LSF**
- an interface to the Fermilab products **spacall** and **needfile** (see section 14.3)
- AFS token renewal at job execution time
- a remote shell facility allowing commands to be executed from any host; if the host is neither a client nor a server, **fbatch** issues the **LSF** command via the **rsh** facility (see section 13.3)
- case-insensitive resource names

You need to run the command **`setup fbatch`** before accessing **fbatch** commands.

When you setup **fbatch**, you will also be able to access the man pages for these commands. Running **`man fbatch`** returns a list of all the commands supported under **fbatch**. For a complete description of the **fbatch** product, refer to the *fbatch User's Guide* (PU0152).

Several of the **fbatch** commands are illustrated below, organized by function. For complete information on each of the commands, see the man pages.

14.2.1 View Host Information

To see which hosts and resources are defined in your cluster, you can issue the command:

`% fbatch_hostinfo`

The configuration information returned includes: host name, host type, host model, CPU factor, number of CPUs, total memory, total swap space, whether the host runs **LSF** servers or not, available resources denoted by resource names. The host name, host type, and host model fields are truncated if too long. The CPU factor is used to scale the CPU load value so that differences in CPU speeds are considered by LIM[1]. The faster the CPU, the larger the CPU factor.

The output is returned in this format:

```
HOST_NAME      type    model  cpuf ncpus maxmem maxswp server RESOURCES
fsgi02          SGI R4400Ch2  84.0    16   511M  2755M    Yes (irix any fsgi02)
fsui02       SUNSOL ULTRA167  93.0     4   320M   889M    Yes (sparc any sun fsui02)
fibb01          AIX     I560  39.0     1   192M   400M    Yes (aix any fibb01)
fncl10          AIX     I370  49.0     1   128M  1136M    Yes (aix any clubs fncl10)
fibi01          AIX     I590  62.0     -      -      -     No ()
fsgi01          SGI   I4D420  30.0     -      -      -     No ()
```

14.2.2 View Queue Information

The **`fbatch_queues`** command lists the available **LSF** batch queues:

`% fbatch_queues`

The output returned is in the following format. A dash (−) in any entry means that the column does not apply to the row. In this example some queues have no per-queue, per-user or per-processor job limits configured, so the `MAX`, `JL/U` and `JL/P` entries are dashes. The man page describes each of the fields.

1. Load Information Manager (LIM) is a daemon process that keeps track of the load indices.

QUEUE_NAME	PRIO	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP
test_queue	99	Open:Active	-	-	-	-	0	0	0	0
e831_long	16	Open:Active	1	1	-	-	0	0	0	0
e831_short	14	Open:Active	-	10	-	-	0	0	0	0
30min	10	Open:Active	-	5	-	-	1	1	0	0
30min_disk	10	Open:Active	-	5	-	-	3	3	0	0
4hr	8	Open:Active	-	5	-	-	2	0	1	1
4hr_disk	8	Open:Active	-	5	-	-	5	2	3	0
12hr	6	Open:Active	-	5	3	-	3	0	3	0
12hr_disk	6	Open:Active	-	5	2	-	7	4	3	0
1day	4	Open:Active	-	5	1	-	0	0	0	0
1day_disk	4	Open:Active	-	5	1	-	7	0	7	0
4day	2	Open:Active	-	5	0	-	33	17	12	4

You can submit jobs to a queue as long as its `STATUS` is `Open`. However, jobs are not dispatched unless the queue is `Active`.

14.2.3 Submit a Batch Job

The **`fbatch_sub`** command is used to submit a job to the batch system. The most common arguments used are **`-q`** (queue name), **`-R`** (resource requirements), **`-o`** (stdout redirection), **`-e`** (stderr redirection), and **`-N`** (notify via email).

As an example, here we submit a script called `myjob` to the `4hr` queue, specify an IRIX host, and request notification. The stdout is redirected to `myjob.out`, and the stderr to `myjob.err`:

`% fbatch_sub -N -q 4hr -o myjob.out -e myjob.err -R "irix" myjob`

On systems running AFS, **fbatch** will prompt you for your AFS password[1]:

```
Enter AFS Password...
Reenter AFS Password...
fbatch_sub executing LSF command locally on fsui02....
```

When your job begins, you will automatically receive a renewed AFS token on the execution host.

14.2.4 Monitor Submitted Batch Jobs

fbatch provides several commands that allow you to monitor your job. The usage examples below use a sample job number 1022:

Display a listing of running jobs

`% fbatch_jobs`

If no options are supplied, the list will contain only your running jobs. To see *all* running jobs, use the **`-u all`** option. Output is returned in this format:

```
JOBID USER     STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
 1022 aheavey  PEND  30min      fsui02                  sleep1     Sep 10 09:56
```

Display the stdout and stderr of a job

`% fbatch_peek 1022`

The format of the output varies according to the files.

1. Your password gets encrypted by **fbatch** using PGP.

Display history information about a job

% fbatch_hist 1022

Output is returned in the format:

```
Summary of time in seconds spent in various states:
JOBID USER    JOB_NAME   PEND    PSUSP   RUN     USUSP   SSUSP   UNKWN   TOTAL
 1022 aheavey sleep1     7       0       35      0       0       0       42
```

14.2.5 Control Submitted Batch Jobs

For jobs that are in a queue awaiting execution, **fbatch** provides commands to move jobs within the queue, and to modify the resource requirements of the job. The usage examples below use a sample job number 1022:

Move Job within Queue

Move job to the bottom of the queue:

% fbatch_bot 1022

Move job to the 2nd position from the top of the queue:

% fbatch_top 1022 2

☞ The **fbatch_bot** and **fbatch_top** commands, above, move jobs within queues *relative to the user's own jobs*. You cannot move your job ahead of another user's job with these commands.

Change Job Parameters

Change the resource requirements of job:

% fbatch_modify -R aix 1022

Migrate a batch job to another host:

% fbatch_mig -m newhost 1022

Suspend, Resume, or Kill a Job

Suspend (stop, but do not cancel) job:

% fbatch_stop 1022

Resume job:

% fbatch_resume 1022

Cancel job:

% fbatch_kill 1022

14.3 Related Software Components

This section describes other software components that can be used with the **LSF/fbatch** batch system.

needfile

needfile is an interface to the **HPSS/Unitree** central mass storage system. The **needfile** system provides two functions. If the user's data does not yet exist in **HPSS**, **needfile** tells the system to copy the data there from its original medium. **needfile** then provides access to this data for the user. This data remains in the HSM based on a least-recently-used policy. There is no guarantee how long data will remain in the HSM.

To access **needfile**, first run the command `setup nt` (setup not required if accessed via **fbatch**).

A **needfile** reference manual exists on the Web. To find it, start on the Computing Division home page, select *Mass Storage* under **Systems and Networking** and then *fmss* to find **needfile tool**. There is also documentation there on HPSS.

spacall

The **spacall** utility (space allocator) provides scratch disk storage for a job. **spacall** is invoked under **fbatch** by submitting a job to a specially defined queue; for example, on FNALU the `*_disk` queues have been configured for it. The path to the scratch space is stored in the environment variable *$CLUBS_WORKDIR*.

Chapter 15:   Tape Handling

In this chapter we discuss the principal tape handling software and facilities available at Fermilab. Start-up information for running and monitoring **OCS** tape mounts is provided, and the **OCS** X interface is introduced. Several tape I/O packages are briefly described.

OCS manages tape drive allocations and operator-assisted mounts, whereas **RBIO** and **DAFT** are tape I/O packages. **FTT** is a low-level C subroutine library common to **OCS**, **RBIO** and **DAFT**; it can also be used directly for tape I/O and tape drive testing. **FMB** includes features for both tape mounting and I/O. These are all **UPS** products, and are accessible in the standard way via the **`setup`** command.

Since many groups and experiments have customized their tape-handling routines, some or all of the applications described here may not be available or appropriate for you. Contact your group leader to find out what procedures have been established for your group.

15.1 Operator Communications Software (OCS)

OCS is a package that performs and manages tape drive allocations, operator-assisted tape mounts and tape drive use statistics. Its logical-to-physical tape device name translation helps not only human communication, but hides many platform-specific idiosyncrasies from users. The latest version as of this writing is v3.0.

OCS is not a tape I/O package. It works well *with* such packages (e.g., **RBIO**, **FMB**, **DAFT**).

OCS functions are available via three separate interfaces:

- FORTRAN/C library of subroutines
- Command line tools
- X Motif tools

The features available in each of these interfaces overlap to a great extent. However, since the different interfaces are by their nature geared towards different uses, the functionality is not 100% duplicated across them. The FORTRAN/C subroutines provide the most flexibility and functionality for users requiring multiple mount requests, the command line tools are generally used in shell scripts, and the X interfaces are very useful for monitoring tape drive statistics.

The X interfaces are fairly intuitive, so we give you enough information to bring them up (section 15.1.2), but we do not describe them in detail.

The **OCS** functions are fully described in the *OCS Reference Guide*, document number GA0012, available on the Web. The reference guide and other documents are also available wherever **OCS** is installed and setup, in the directory `$OCS_DIR/doc`.

15.1.1 OCS Basics

Monitoring Tape Mounts

During the course of your work, you may need to monitor different aspects of the job and possibly contact the operators. **OCS** provides functionality to perform these job management activities. For your convenience, we list the appropriate command next to the function in the table below, and indicate whether you can perform the function using one of the X interfaces. You will need to see the *OCS Reference Guide* for usage information on these commands and for the associated FORTRAN/C subroutines.

Function	Command	X interface
Mark a tape drive broken so that it will not be allocated until it is repaired	**ocs_broken**	**xocs**
Log statistics as to how much the drive was used	**ocs_report_stat** **ocs_init_stat**	
Send an attention message to the operator	**ocs_message**	
Send a request to the operator to run a cleaning tape through a tape drive	**ocs_clean_it**	
Display status of tape drives in the **OCS** database	**ocs_tape**	**xocs**
Display pending mounts	**ocs_pending**	**xocs** **xtapeview**
Display tape drive statistics	**ocs_devstat** **ocs_stats**	**xocs**
Display tape mount log	**ocs_mrlog**	**xocs**
Display tape drives that may need to be cleaned	**ocs_clean_list**	**xocs**

Sample Tape Mounting Process

Here is a sample tape mounting process using the command line tools. Read through it to see what steps are involved and what kind of response to expect from the system at each step. Note that the **OCS** commands come with many options that are not shown here.

First, display the list of available tape drives:

% ocs_tape

```
HOSTNAME DEVICE   TYPE     ALLOCATED STATUS  VSN             USERNAME UID   AUTH
bastet   dumdlt4  DLT4000  allocated working -               root     0        n
bastet   isis2    EXB-8505 unallocd  working -               -        -        n
bastet   horus    EXB-8200 unallocd  working -               -        -        y
```

Request allocation of a tape drive so no other user may access it (notice only one device shows authorization as “yes”):

% ocs_allocate

```
bastet horus
```

Send a request to the operations staff to mount a tape on the drive you have allocated:

% ocs_request -t horus -w -v FGMS04

```
ocs_request: success
```

Verify that the tape was mounted correctly:

% ocs_check_label -t horus -w -v FGMS04

```
ocs_check_label: Success
```

Set the tape drive characteristics according to your needs (we recommend that you always do this; don't assume the drive has been left in any particular state):

% ocs_setdev -t horus

Request the appropriate device file for reading and/or writing the tape according to the characteristics you've set:

% ocs_devfile -t horus

```
/dev/rmt/tps0d3nrv
```

Perform your task on the loaded tape. Normally this involves running a program that calls tape I/O routines from **RBIO** or another I/O package. For simplicity in this example, we just use the UNIX **dd** utility to get the tape contents (we have loaded an ANSI initialized tape with no data):

% dd if=/dev/rmt/tps0d3nrv conv=unblock cbs=80

```
VOL1FGMS04              ftt                                                    4
HDR1
0+2 records in
0+1 records out
```

A useful feature to include in this example is the device statistics function. It provides more detailed information for the end user than the X interface implementations, which were designed more for administrative purposes.

% ocs_devstat -t horus

```
----------------------------------------------------------------------------
Collecting Tape Drive Statistic ------------------- Thu Oct  2 12:43:27 1997
----------------------------------------------------------------------------
Host Name            = bastet
Device Name          = horus
Device Filename      = /dev/rmt/tps0d3
Device Type          = EXB-8200
Controller           = SCSI
Vendor Id            = EXABYTE
Product Id           = EXB-8200
Firmware Id          = 268N
Serial Number        = -
Number of Hours On   =         -1   In Motion =         -1
Count/KBytes    Read =       2200       Write =         -1
Errors          Read =          0       Write =         -1
Comp Ratio      Read =         -1       Write =         -1
Compresion           = NO
Density              = 8200
Media Type           = 133
Block Size           = 0
Block Total          = 2294048
Count Origin         = Exabyte_Extended_Sense
Remain Tape/KBytes   = 2290569
SCSI Sense Code      = 0
SCSI ASCQ            = 0
Track Retry          = -1
```

```
Stop/Start Count     = -1
SCSI Test Unit Ready = 0
SCSI Sense Key       = NO_SENSE
-----------------------------------------------------------------------------
Tape Not Present:  N | Write Prot:   N | Clean Bit:    N | Drive Cleaned: N
Beginning of Tape: N | At File Mark: N | End of Media: N | End of Tape:   N
Ready Bit:         Y | Power Fail:   N | SCSI ILI Bit: N |
-----------------------------------------------------------------------------
```

Dismount the tape (i.e. rewind and unload it):

% ocs_dismount -t horus

```
ocs_dismount: Success
```

Finally, deallocate the tape drive so that someone else can use it:

% ocs_deallocate -t horus

```
ocs_deallocate: Deallocated :horus
```

Tape Mounts with Batch Jobs

Most of the time, users run tape mounts in conjunction with batch jobs. We strongly recommend that you include the tape mount allocation, mount request, dismount and deallocation *within* your batch job. If you don't, you risk preventing others from using the tape drive while your job is waiting to run and after it has finished. A **-q** option is provided for the **ocs_allocate** command to allow you to queue for tape drive allocation so that your job won't fail if a drive isn't immediately available when it starts to run.

Here is a sample batch job script that incorporates these recommendations:

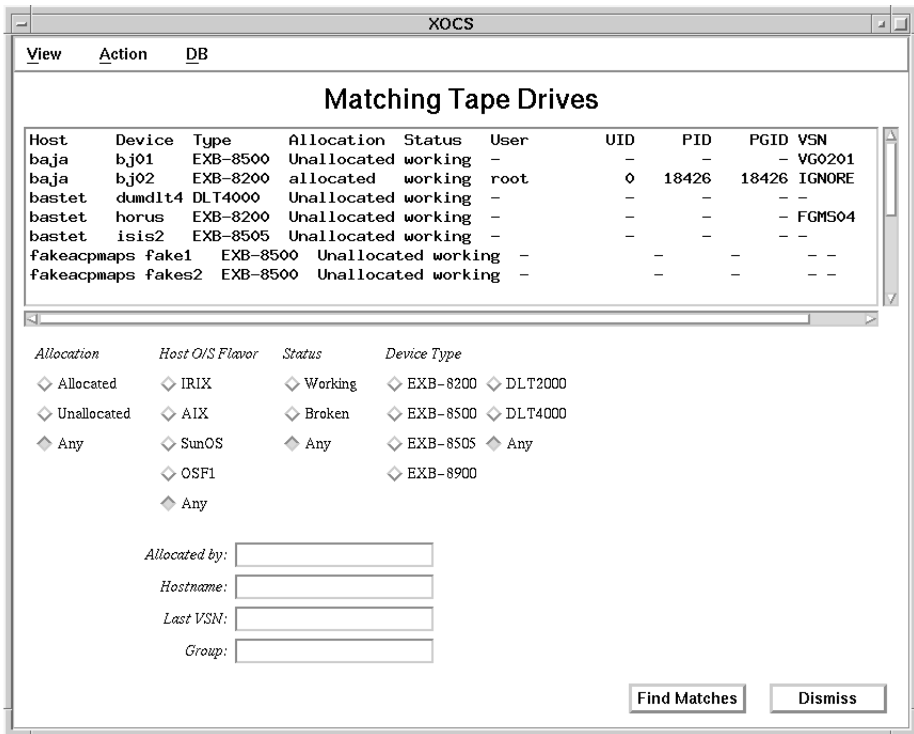
```
#!/bin/csh
setup ocs
set td=`ocs_allocate -q -h localhost -d exabyte_850x | cut -f2 -d” “ -`
if ( $status != 0 ) then
     echo ocs_allocate failed with message: $td
     exit 1
endif
set drive=`echo $td | cut -f2 -d” “ -`
ocs_request -t $drive -v fr3147 -r
if ( $status != 0 ) then
     echo ocs_request failed
     ocs_deallocate -t $drive
     exit 1
endif
set dfile=`ocs_devfile -t $drive`
if ( $status != 0 ) then
     echo ocs_devfile failed with message: $dfile
     ocs_dismount -t $drive
     ocs_deallocate -t $drive
     exit 1
endif
ocs_setdev -t $drive -v -d 8500
if ( $status != 0 ) then
     echo ocs_setdev failed
     ocs_dismount -t $drive
     ocs_deallocate -t $drive
     exit 1
endif
setenv MY_DEVFILE_VAR $dfile
e831job.run
ocs_dismount -t $drive
ocs_deallocate -t $drive
exit 0
```

15.1.2 The OCS X Interfaces

Remember that your *DISPLAY* environment variable needs to be set properly for X windows applications (see section 9.2).

xocs

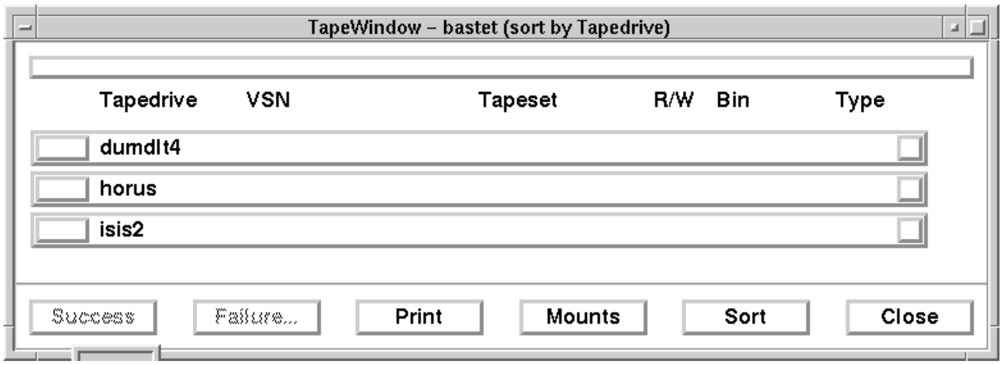
xocs is an X interface that you may find useful for viewing tape statistics and history. It allows you to perform a subset of the functions available through text commands. After setting up **OCS**, enter `xocs` at the command line. You'll see the following screen (shown for version v3.0):



The menu options under View and Action include most of the features you will need.

xtapeview

Another X interface to **OCS** which allows you to view pending mount requests is **xtapeview**. As a user, you are not permitted to respond to mount requests, only view them. Invoke this interface with the command `xtapeview`. A window will appear from which you can choose a node. Click on the node you want, then you'll see a window like the one below which shows the drives associated with that node, and any pending mounts.



If you have a color screen, the banner at the top is blue when no mounts are pending, and red if one or more are. The status button (to the left of the drive name) will be red if a mount is pending on that drive. If you don't have a color screen, there are gray-tone representations of blue and red. See `$OCS_DIR/doc/xtape.ps` (recall that *$OCS_DIR* is defined during setup) for a full description of this application, and to see how to customize the color scheme.

15.1.3 Using Provided Examples to Get Started

Examples are provided in the `$OCS_DIR` directory (defined during setup). You can look at `$OCS_DIR/doc/viewgraphs.ps` for more single-command examples and their expected output.

The directory `$OCS_DIR/examples` provides sample programs for FORTRAN, C and Bourne shell script which are intended to help you understand how **OCS** works. The executables are named `ocs_ctest`, `ocs_ftest`, and `ocs_btest` for C, FORTRAN and Bourne shell, respectively. Each program carries almost identical functionality. You may want to use the source of these programs (`ocs_ctest.c`, `ocs_ftest.fs` and `ocs_btest.sh`) as an aid in developing your own programs. The `$OCS_DIR/examples/README` file explains how to use the examples.

These example programs can actually send a mount request to operations staff: please keep this in mind if you experiment with OCS on Fermilab Computing Division systems such as FNALU.

15.2 Raw Buffered I/O (RBIO)

The **RBIO** (Raw Buffered I/O) library provides an I/O interface that is FORTRAN- or C-callable, medium-independent, and UNIX flavor-independent. It allows your FORTRAN or C program to perform buffered I/O on a file object, which may be a regular file or a labeled tape. The **RBIO** library is used in the compilation, linkage, and execution phases. The actual reading/writing level is at the primitive logical record level; **RBIO** does not understand record contents.

RBIO is documented in the *RBIO User's Guide*, document number SU0033.

15.3 DAta From Tape (DAFT)

DAFT (DAta From Tape) consists of a set of routines written by the DART collaboration to provide event-level reading and writing from DART-formatted records to/from disk and tape. Information on DART can be found under *Projects & Working Groups* on the Computing Division home page.

DAFT performs the following functions:

- retrieve data written on tape or disk in the format used by DART
- write data on tape or disk in the DART format

The routines in **DAFT** are widely portable, and currently run on the supported UNIX platforms IRIX, Solaris, AIX and OSF1. They offer a FORTRAN interface.

DAFT was written in two layers to allow for a lower level that is easily portable. Therefore, two sets of access routines are provided:

- a “low-level” set that embeds the different levels of platform-dependence and delivers records to the higher level routines
- “high-level” routines that unpack events from the records and provide the event interface

The DART on-line format packs events into logical records, where each record may contain more than one event, in order to keep the drives streaming at full speed. The routines handle large events spanning records (the maximum record length is 64 Kbytes), and correct the byte ordering of data on platforms of different *endianship*.

For more information on **DAFT**, see the *DAFT (DAta From Tape) User’s Guide*, document number PN504.

15.4 Fermi Tape Tools (FTT)

The Fermi Tape Tools (**FTT**) product is a low-level C subroutine library. It provides:

- basic tape I/O
- support for obtaining and tracking available tape statistics
- a set of informational and support subroutines for tools that allocate and control tape usage (i.e. **OCS**)

FTT has been designed to provide platform- and drive-independent I/O on the UNIX platforms supported at Fermilab. It has also been designed to be portable to other platforms and drives. In addition to its use as a transparent layer of software in several products (namely **DAFT**, **OCS**, and **RBIO**), the **FTT** subroutine library can be called directly by user-written programs. **FTT** also includes an interactive test program which, among other things, can verify that tape drives are behaving properly.

FTT is documented in the *Fermi Tape Tools Library User’s Guide*, document number PU0236.

15.5 Fermi Modular Backup (FMB)

The Fermi Modular Backup (**FMB**) system is a locally-written set of utility scripts used for backup, restore, and other similar administrative tasks. It was designed for use on the supported UNIX flavors. **FMB** uses short, simple-to-write *modules* to provide different tape rotation schemes, tape mount systems, archive formats, and so on. It also supports backup of remote machines, and/or remote tape drives.

See the *Fermi Modular Backup System User’s Guide*, document number PU0164.

Chapter 16: Software Development

This chapter gives an introduction to UNIX software development tools in common use at Fermilab, providing information on:

- Supported languages
- Compiling and linking in C, C++ and FORTRAN
- Debugging

We do not include a discussion of general programming here, but rather, aspects of software development particular to UNIX. You will need to reference the man pages and the vendors' manuals for more system-specific details.

Useful documents relating to FORTRAN, C, and C++ programming can be found under **Software Development** on the *UNIX Resources* Web page.

Many software development subjects of interest to Fermilab users are beyond the scope of this manual. Among them are:

- Object Oriented programming techniques including NextStep
- CASE tools
- Neural Network methods
- Lattice Gauge techniques used in ACP-MAPS, CANOPY, et.al.
- Data Mining methods available in the CAP facility.
- Many excellent commercial tools for building Graphics User Interfaces, debugging, migrating and analyzing performance of user code, building Database interfaces, etc.

16.1 Overview of Programming Languages and Tools

This is a short overview of some common programming languages. Our aim here is to give you a general idea of what tools are available, and how they can be used. The list is ordered from low to high level, and indicates common uses:

Assembler	not used at Fermilab
C	system services, user interface, general utilities
FORTRAN	FORmula TRANslation (physics calculations)
C++	object oriented general programming
Perl	interpreter, general purpose
Python	object-oriented and/or general purpose interpreted scripting language
Tk	interpreter, GUI interfaces

Assembler

Traditionally, assembler has been needed for a couple of reasons:

- Access to system services and hardware
- Tuning program efficiency

Assembler programming is not generally necessary or desirable on the RISC based UNIX systems presently in use. RISC system performance is so heavily dependent on pipelining and various caches that it is extremely difficult, if not impossible, for an individual to write more efficient assembler code than is generated by the higher level language compilers. The C language provides all the hardware access and system service capabilities traditionally provided by Assembler.

C

As noted above, C has filled the programming niche traditionally occupied by Assembly language. In addition to its normal use as a high-level programming language, C can act as a universally portable Assembler.

Because the C language was created, has evolved, and has become standardized hand-in-hand with the UNIX operating system, it is the language of choice for applications involving system services and user interfaces. Using C is discussed in several of the following sections.

Now that an ANSI C standard exists and is widely implemented, portability of C code is much improved. ANSI C compilers are the default on most Fermilab systems. Likewise, adoption and availability of POSIX standards for operating system services has greatly improved program portability.

An excellent reference book for C programming is *The C Programming Language* by Kernighan and Ritchie published by Prentice-Hall.

FORTRAN

FORTRAN remains the most effective language for mathematical calculations. This is due partly to decades of research which has produced highly efficient optimizing compilers, and partly to the millions of lines of tested, portable code already in use.

C++

C++ adds object oriented programming constructs to the C language. At the risk of oversimplifying, it seems that C++ is substantially harder to learn and to use for writing new programs, but the resulting programs are much better structured, more maintainable, and more shareable than traditional C or FORTRAN programs.

An additional advantage of C++ is the availability of class[1] libraries. A Standard Template Library will come with most compilers soon. This library will contain many useful low level classes for such things as strings and streams I/O. Also, some vendors include other commercial class libraries as added value to their compilers. In addition to the vendor-supplied versions, C++ is available as a part of Gnu C. This has increased its popularity.

The C++ language standard is (still!) in the process of adoption by the ANSI and ISO standards committees. Since the C++ standard has not been finalized (although it changed significantly in December 1996), there are at this time (November 1997) cross-platform porting issues. None of our major vendors (SGI, IBM, GNU) are yet providing a draft-standard-compliant compiling and runtime toolkit. The safest bet for generating portable C++ code is to avoid using newer features such as exceptions and templates, which may be implemented differently (or not at all) by the

1. A *class* is similar to a *structure definition* in C.

various C++ compiler vendors, at least until the vendors catch up with the standard. As an alternative, use the `g++` command to run the Gnu integrated C/C++ compiler on all platforms. Using the `g++` command instead of `gcc` gives you appropriate C++ linking.

For documentation, you may refer to:

- the `CC` (upper case), `gcc` and `g++` command man pages
- *The C++ Programming Language*, Addison-Wesley, by Bjarne Stroustrup

Perl

perl is installed at Fermilab as part of the **shells** product. The man page for **perl** gives a good brief description:

> **perl** is an interpreted language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. It's also a good language for many system management tasks. The language is intended to be practical (easy to use, efficient, complete) rather than beautiful (tiny, elegant, minimal).

There is an excellent text published by O'Reilly & Associates, Inc. on **perl**.

Python

Python is an interpreted, interactive, object-oriented programming language often compared to **Tcl**, **Perl**, **Scheme** and **Java**. **Python** combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high level dynamic data types, and dynamic typing. There are interfaces to many system calls and libraries, as well as to various windowing systems (e.g., **X11**, **Motif**, **Tk**, **Mac**, **MFC**, **STDWIN**). New built-in modules are easily written in C or C++. Python is also usable as an extension language for applications that require a programmable interface.

An on-line document for **Python** is available in the CD documentation database.

Tk

Tk is an **X11** toolkit that provides the Motif look and feel, and is easy to use for building graphical interfaces largely because it is built on an interpreted language. It can be used with a variety of languages including **Tcl** (**Tk** used to be *solely* implemented using **Tcl**), **Perl**, and **Python**.

The best reference for **Tk** is the book *Tcl and the Tk Toolkit*[1], by John K. Ousterhout, published by Addison-Wesley. The `README` file under the `tk` directory in `$TK_DIR` (created during setup) points to a draft of this book. Also see the man pages for information on these languages.

Other Languages and Language-Related Tools

Other tools exist that are commonly used as languages in the appropriate context. These include, for example, the various UNIX shells (as discussed in section 4.4), and **awk**, **sed**, **yacc**, and **lex**, for which O'Reilly & Associates, Inc. provides excellent texts.

There are many other languages which are not widely supported or are not in general use at Fermilab. These include Pascal, Modula-2, Lisp, Forth, Bliss, and others. We do not discuss them in this document.

1. Some publishers' catalogues use an ampersand (&) rather than the word "and"; check both in database searches.

16.2 Introduction to C and FORTRAN on UNIX

16.2.1 The C Compiler: cc

`cc` (lower case) is the vendor-supplied C compiler command on all Fermilab-supported UNIX systems (except LINUX, where it is **`gcc`**). ANSI C compilers are the default on most Fermilab systems. To compile one or more C source files (***filename.c***), you run the **cc** utility. **cc** automatically invokes the link editor **ld** unless the option **`-c`**, which explicitly suppresses linking, is used.

16.2.2 The FORTRAN Compiler: f77

All the Fermilab supported UNIX systems have good FORTRAN 77 compilers, which provide some minimal extensions. These compilers also recognize most DEC-supported VAX-FORTRAN extensions.

`f77` is the FORTRAN compiler command on all Fermilab-supported UNIX systems (except LINUX, where it is **`g77`**). The **`f77`** command controls both compilation and linking functions automatically using appropriate FORTRAN runtime libraries. **f77** can produce object modules, partially linked objects, or executable programs, as appropriate. The option **`-c`** explicitly suppresses linking.

☞ Note that on AIX, to get help on **f77** you need to type **`man xlf`** rather than **`man f77`**.

16.2.3 C and FORTRAN Compiling Basics

UNIX compilers, including **f77** and **cc**, generally use both filename extensions and the file content to determine how to handle the files listed on the command line. The commonly used extensions are:

For C

`c`	C source, e.g., `myfile.c`

For FORTRAN

`f`	FORTRAN source, e.g., `myfile.f`

For both

`o`	object file, e.g., `myfile.o`
`a`	archive library, e.g., `mylib.a`
(none)	executable image, e.g., `myfile`

For historical reasons the UNIX linkers produce by default an executable named `a.out`; the option **`-o`** ***filename*** is available to override this default.

We list here some basic compiling and linking examples. In these examples, we use a source file named `foo.c`, where `c` is the standard extension for C. In all instances shown here `c` can be directly replaced by `f`, and **`cc`** by **`f77`**, for FORTRAN.

To produce:	... enter the command:
`foo.o` object module	**`cc -c foo.c`**
`foo` executable from source	**`cc -o foo foo.c`**

`foo` executable from source + object file

`cc -o foo foo.c myobj.o`

`foo` executable from source + library

`cc -o foo foo.c $CERN_DIR/lib/libmathlib.a`

`foo` executable from source + X11/Motif (standard system libraries)

`cc -o foo foo.c -lXm -lXt -lX11`

The options used with the **`cc`** and **`f77`** commands are discussed in later sections.

16.2.4 Linking Order

Most UNIX linkers process source, object and library files in the order that they occur on the command line. Backward library references, from a file to an earlier library, will not be satisfied, except under AIX. It may be necessary to list a library more than once for successful linking.

AIX loads the entire content of all libraries into memory, eliminating ordering problems, but potentially creating memory usage problems. Listing a library more than once is unnecessary, and positively undesirable under AIX.

16.2.5 Displaying Active Options

You may wish to know which options are active for a given compilation, in order to verify that the defaults are what you expect. Each platform seems to provide this information somewhat differently:

Platform	Option	Result
AIX	**`-q listopt`**	Source listing with options, to file `*.lst`
IRIX	**`-v`**	Options and other details, to `stdout`
OSF1	**`-V`**	Source listing with options, to file `*.l` (FORTRAN only)
SunOS	not available	

16.2.6 Option Passing

At each stage of compilation, any unrecognized command line options are passed on to the next stage. Options that could be valid for more than one stage can be explicitly passed to a particular stage. To direct an option to a specific phase of compilation or linking, use the option **`-W`** (for AIX, IRIX, OSF1) or **`-Qoption`** (for SunOS). The phase is identified by the letter immediately following the **`-W`**. Thus, for example, **`-Wl,-m`** tells **f77** or **cc** to pass the option string **`-m`** to the **`l`** (link) phase. The **f77** command line might read:

`%` **`f77 -Wl,-m foo.f`** for AIX, IRIX, or OSF1, and

`%` **`f77 -Qoption ld -m foo.f`**

for SunOS (where **`ld`** is the loader program)

16.3 Introduction to C++ on UNIX

CC (upper case), **g++** and **gcc** are all C++ compiler commands on the Fermilab-supported UNIX systems that provide C++ (see section 16.1 for a brief discussion). Just as C++ is a superset of C, the C++ compilers are very similar to C compilers in that their options are usually a superset of C compiler options. The basic compiling information about C in section 16.2.3 is also applicable to C++, with the following exceptions[1]:

- Source filename extension conventions are compiler-dependent. Check the man pages for **CC** (upper case) to determine the extensions used on your system. Extensions include:

`C`	(upper case)
`c`	(lower case)
`cxx`	
`cpp`	
`cc`	
`c++`	

- The C++ compiler is invoked with **CC** (upper-case), **g++** or **gcc**
- There are additional compiler options specific to C++; see the man pages.

The C++ compiler may not yet be installed on your Fermilab UNIX system.

16.4 C and FORTRAN Compiler Options

The default compiler options will produce a usable program on any of the supported platforms, however they may not be optimal for many situations. Several additional options are discussed in this section.

A caveat: One very annoying “feature” of the **f77** and **cc** commands is that some of the options must be specified with whitespace (at least one blank character) between the option identifier and the option value, others without whitespace and still others with or without, according to the user’s choice. For instance, on IRIX systems, there must be whitespace between **-o** and the name of the executable file but there must not be any whitespace between **-l** and the library file name.

If compilers are upgraded it is possible that some of these options could change. For full details on all the options, see the man pages and the vendor compiler documentation.

16.4.1 Commonly-Used Options

-c	suppress linking, produce object file `*.o`. The linker is called as part of the compilation process by default.
-L ***directory***	add ***directory*** to the default linker search list; not needed for user libraries. The **-L** option adds directories to the linker’s default search path. **-L** directories are searched ahead of system library areas. A user library could accidentally match the name of an obscure system library, with startling results.

1. This doesn’t apply to Gnu C++. The compiler command for both Gnu C and Gnu C++ is **gcc**, and the available Gnu compiler options are different from the vendor compiler options.

-l*file*	search library `lib`***file***`.a`, from the default areas (the `lib` gets prefixed and the `.a` appended automatically)

☞ With user-written libraries, specify the libraries on the command line with their true file names and a full path, without using the **-L** and **-l** options.

-o *file*	(lower case **o**) produce executable program ***file*** rather than `a.out`
-O*n*	(upper case **O**) optimize at level ***n*** where ***n*** is 0, 1, 2, 3, or 4. The meanings of the different optimization levels vary from system to system. See the man pages for details.
-w	suppress informational and non-fatal compiler warnings
-P	(upper case **P**) run **cpp** (C preprocessor) only to produce `*.i` source listing
-p	(lower case **p**) enable profiling; see the man pages for details

Other commonly-used C compiler options are:

-I *directory_name*	extend include path
-D	set value of preprocessor macro

Other commonly-used FORTRAN compiler options are:

-u	IMPLICIT NONE
-C	(upper case **C**) check runtime subscript range

16.4.2 Recommended Options for General Use

As mentioned earlier, the default options may not be optimal for a number of very common situations, namely for debugging, moving binary code between non-identical machines, or tuning for best performance. Even for general usage, some non-default options have proven helpful in avoiding internal compiler limits and providing better compatibility for migrated code.

This list shows options by platform which apply to all situations, and which we recommend for general use. They can be used in addition to other options you might choose based on your specific needs. As stated above, if compilers are upgraded it is possible that some of these options could change, so you should always consult the man pages.

AIX	**-qextname -qrndsngl**
	-qflttrap=invalid:overflow:zerodivide:enable
IRIX	**-trapuv -lfpe**
OSF1	(none)
SunOS	**-xl -fnonstd**

Discussion of General Options

-qextname	(AIX) append underscore (**_**) to external names (default at Fermilab). This is needed if you use any Fermilab or CERN libraries. Better to just make it a habit.
-qrndsngl	(AIX) standard IEEE math on intermediate `REAL*4` results. This produces bitwise identical results to IRIX/OSF1/SunOS for most programs, making port testing easier.

-qflttrap	(AIX) detect and report floating point errors. Without this, errors occur silently, and incorrect results may be produced. Unlike other platforms, these would not be NaN[1], but legitimate floating numbers. Do not use the **:imprecise** option here, as it appears to disable error detection entirely (AIX 3, XLF 2).
-trapuv	(IRIX) set uninitialized variables to NaN, to help catch nonportable code and latent bugs at runtime.
-lfpe	(IRIX) use the Floating Point Exception library, to report or core dump on errors. Without **-lfpe**, errors produce NaN values silently. Must be combined with the *TRAP_FPE* environment variable, or calls to `handle_fpes`, to be effective. *TRAP_FPE* must be set to the value: `OVERFL=ABORT;DIVZERO=ABORT;INVALID=ABORT` See section 9.1 for setting environment variables.
-xl	(SunOS) Extended Language, for Fermilab-required extensions (see section 16.2.2)
-fnonstd	(SunOS) trap floating point errors.

16.4.3 Debugging Option

The **-O**'s here are all upper case, for optimization.

-g	include full symbol table for debugger. This option interacts with the optimizer differently on each platform, so we provide some usage notes:
AIX	you should not use **-O** with **-g**
IRIX	**-g** forces optimization off; use **-g3** to permit optimization
OSF1	**-g** changes default optimization to **-O0** (dash, letter-O, zero) An additional OSF1 option, **-synchronous_exceptions**, puts traps after each floating point operation. This is very slow, but will precisely locate floating point errors. Use this only for code which is known to produce floating point exceptions.
SunOS	you should use **-O1** with **-g**

16.4.4 Portability Option for AIX

-qnomaf	Turns off Multiply/Add instructions in order to enhance portability.

16.4.5 ABI Options Under IRIX 6

Under IRIX 6, there are three Application Binary Interface (ABI) options available. The selected option specifies at compile time which subset of the processor chip's capabilities is to be used. In previous releases of IRIX, and in all releases of the other supported UNIX flavors (as of this

1. NaN is the IEEE standard Not-a-Number bit pattern produced when a floating point operation generates invalid results.

writing), the user is not given a choice of ABI. Under IRIX 6, choosing an ABI is generally necessary if you're doing mixed-language programming or using any libraries other than those supplied by the vendor. The available ABIs and their associated command line options are:

O32 (**-32**)	conforms to the ABI used with all prior IRIX releases
N64 (**-64**)	puts the processor into full 64-bit mode
N32 (**-n32**)	leaves the processor in 32-bit mode but takes advantage of a number of newer features and generally produces more efficient code on the newer processors

This topic is discussed in the Web page *IRIX 6 Application Binary Interface*s, available under *Physics Applications* on the CD home page (look under *Cern Library at Fermilab*).

16.4.6 Speed Optimization Options

Note that on most platforms a combination of options is required for best optimization. Recall that these options may be used in addition to general recommended and other options. The **-O**'s here are all upper case.

AIX	**-O2**
IRIX	**-O3 -mips2**
OSF1	**-O3 -feedback -tune host**
SunOS	**-O3 -cg92 -libmil**

Discussion of Speed Optimization Options

Platform	Options	Comment
IRIX	**-mips2**	Use full R4000 instruction set. This is important on R4000 and later systems.
OSF1	**-feedback**	Order routines for best cache performance, based on actual program execution statistics.
	-tune host	Use best optimization for given generation of chip. The code will run anywhere, but is tuned for local speed.
SunOs	**-cg92**	SuperSPARC V8 instruction set Sparcstation 2 and later. See the man fpversion document.
	-libmil	Hardware-specific inline math

16.4.7 Load Map Option

The load map option is actually a linker option. On AIX, IRIX and OSF1 this requires passing to the link phase the option that controls production of a load map.

Platform	Option	Output
AIX	**-Wl,-bmap:***file*	*file*
IRIX/OSF1	**-Wl,-m**	`stdout` (SysV style list of input/output)
IRIX/OSF1	**-Wl,-M**	`stdout` (BSD style primitive map)
SunOS	**-m**	`stdout`

Additionally, under AIX two related options are available:

-Wl,-bloadmap provides additional internal loader information

-Wl,-bxref provides additional cross reference information

16.4.8 Special FORTRAN Compiler Options

Source Code Listing Option

Each platform has an option that produces a source code listing. The file extensions for these listings are platform-dependent.

Platform	Option	Listing Extension
AIX	**-qsource**	`lst`
IRIX	**-listing**	`L`
OSF1	**-V**	`l`
SunOS	**-Xlist**	`lst`

Saving Local Variables

The FORTRAN 77 standard allows subprogram local variables to become undefined between calls, unless saved with a SAVE statement. Many UNIX **f77** compilers *require* the SAVE statements for retained local variables. SAVE'd variables are usually called *static*, and unSAVE'd variables are called *automatic*. For programs not yet properly equipped with SAVE statements, **f77** command line options are available as follows:

Platform	Static Option	Automatic Option
AIX	(default)	**-qnosave**
IRIX	**-static**	**-automatic** (default)
OSF1	**-static** (default)	**-automatic**
SunOS	(default)	**-stackvar**

C Preprocessor

The C preprocessor (**cpp**), a macro processor, is used in **f77** for conditional compilation, macro expansion and source inclusion. If you plan to run **cpp**, remember to double any backslash characters (****) in your code to prevent their misinterpretation as **cpp** escapes. The option shown in parentheses is the default option for the corresponding platform:

IRIX	(**`-cpp`**) **`-nocpp`**
OSF1	**`-cpp`** (**`-nocpp`**)
SunOS	any input file name `*.F` is automatically preprocessed by **cpp**
AIX	(see below)

The C preprocessor cannot be invoked through the **`f77`** command on AIX systems. It can, however, be invoked under AIX as a separate command:

`% /lib/cpp -P -C` *infile outfile*

See the man pages for **cpp** syntax and usage information.

Extend Search Path for INCLUDE

Use the **`f77`** command line option **`-I`** *path* to extend the search path for the INCLUDE statement on AIX/IRIX/OSF1 (not supported on SunOS), where ***path*** is the path to the directory where the include files are found.

Internal Compiler Limits

You may need to set one of these if you are compiling a very large single source file.

AIX	**`-NQ20000`**
	Larger internal compiler tables. This value should be big enough for almost all programs.
IRIX/OSF1	**`-Olimit 1500`**
	Allows somewhat larger routines to be optimized.

16.5 FORTRAN Programming

There is some additional information about using FORTRAN in the UNIX environment that you will find useful to know.

16.5.1 External Reference and Entry Point Names

In order to avoid conflicts with the C runtime library when FORTRAN and C programs are included in a single program, most UNIX **f77** compilers internally append an underscore to FORTRAN external references and entry point names. At Fermilab we have set up all **f77** compilers to do this by default.

16.5.2 Separate Compilation of FORTRAN Subprograms: fsplit

By default, most **f77** compilers pre-link all the source code being compiled, even when you specify the **`-c`** option. If you compile a library with a single **`f77`** statement, it will usually contain a single module, and be linked as a whole.

The **fsplit** utility identifies and extracts subprograms from the original FORTRAN source file into individual files in the current directory. These files can then be compiled separately so that they retain their identity when assembled into a library.

The names of the extracted individual files are taken from their corresponding subprogram names. On some systems **fsplit** will overwrite any pre-existing file, including the original source file, whose name matches any of the subprogram names.

See the man pages for more information on **fsplit**.

16.5.3 AIX-Specific Issues

You must `CALL EXIT` to avoid having a `STOP` message printed on `stdout` at program termination.

The `OPEN` statement parameter `READONLY` is unavailable. All files are opened with `WRITE` access. This makes file sharing problematic under AIX.

16.5.4 Loading Block Data Modules

Many UNIX **f77** compilers enforce the standard restriction that variables in COMMON must be initialized only in BLOCKDATA subprograms.

To ensure the loading of BLOCKDATA subprograms from libraries, declare the BLOCKDATA program name as EXTERNAL in some important module which you know will be loaded.

16.5.5 Program Control

Command Line Arguments

A FORTRAN program can easily evaluate arguments included on the command line that runs the program. A couple of examples follow.

- The IARGC function returns the number of command line arguments:

  `N = IARGC ()` Sets `N` to the number of command line arguments

- The GETARG subprogram returns the value of a specified argument:

  `CALL GETARG(I , STR)` Puts the `I`'th argument into string STR

Environment Variables

The GETENV subprogram provides the values of environment variables. For example, to copy the value of variable `MY_OUT` into string `OUTFILE`, include in your source file:

```
CALL GETENV ( 'MY_OUT' , OUTFILE )
```

Printing

The usual FORTRAN carriage control characters placed in the first column of formatted output files are not interpreted by most UNIX text handling utilities. Use the UNIX **asa** utility to convert such FORTRAN output files to an equivalent standard ASCII text form. **asa** handles blanks, 0, 1 and + in column 1, removing any other characters. See the man pages for **asa** for details.

16.5.6 Future FORTRAN Enhancements

AIX XLF 3

This document describes the AIX XLF 2 compiler. IBM's standard **f77** compiler, XLF 3, is not yet available at Fermilab.

FORTRAN 90

The FORTRAN 90 standard includes FORTRAN 77 as a subset, and makes standard many of the extensions in common use. FORTRAN 90 is not yet commonly installed at Fermilab, and in fact we recommend that you avoid using FORTRAN 90 extensions until it is widely available. This document is written for **f77** users.

16.6 Obsolete Programming Features

You may encounter these features in older code.

AIX XLF

Some years ago the AIX FORTRAN compiler command was **`xlf`**. The standard **`f77`** command is now available under AIX, and should always be used in place of **`xlf`**. However, as mentioned earlier, you still need to type **`man xlf`** to access the man pages.

Calling BLOCKDATA

Some systems (VS-FORTRAN) required an explicit call to each `BLOCKDATA` routine if you wished to force loading of that routine from a library. This is not necessary on any supported UNIX system.

BUFIO

The Fermilab **bufio** product for accessing raw variable length records on tape and disk is no longer supported. Improved capabilities are being supported in **RBIO** and **DAFT** (see sections 15.2 and 15.3).

RANLIB (SunOS/4)

The RANLIB utility added necessary library symbol tables under SunOS/4. This is done automatically under SunOS 5.

ar -s Option (ULTRIX)

The `ar -s` option added necessary library symbol tables under Digital's ULTRIX[1] and some earlier operating systems. This is done automatically under all Fermilab supported systems.

16.7 C and FORTRAN I/O

This section mainly applies to FORTRAN. For C, all you need to know is that the **RBIO** and **DAFT** libraries mentioned below are available.

Recall that file names in UNIX are case sensitive. It is customary to use lower case for normal files, reserving upper case names like README for documentation and control files.

Note that you cannot use the shell metacharacter tilde (**~**) to specify a home directory within a C or FORTRAN program; **~** is valid only on a UNIX shell command line (for all shells but **sh**). **`logdir`** can be used within programs for this purpose (see section 6.1.2).

16.7.1 Records

The UNIX operating system treats a disk file as a sequence of bytes. Interpretation of data as records is entirely up to individual applications. The FORTRAN I/O libraries provide the necessary record handling for FORTRAN programs. READ statements return only the content of the records, and not the control words mentioned below.

Formatted records are terminated by a new-line character <Ctrl-l> (lower case L), consistent with other UNIX text handling programs.

Unformatted records are both preceded and followed by a 32 bit integer containing an exclusive byte count.

16.7.2 Tapes

Tapes of course have real physical records, and must be handled differently than disk files. A tape file is sometimes called 'character special' to indicate that it is not accessed on a character-by-character basis. Tape handling is covered in Chapter 15.

There are no industry-wide standard tape I/O routines callable from FORTRAN. Fermilab makes use of several locally-written packages, including:

RBIO	labeled and directoried tape access (see document SU0033)
DAFT	raw binary tape access (see document PN0504)

You can find C and FORTRAN examples in Appendix B of the *DAFT User's Guide*, document PN0504.

1. ULTRIX has been superseded by Digital UNIX (**`uname`** returns `OSF1`).

16.7.3 Standard Input and Output

In conformance with the FORTRAN standard, `READ`s and `WRITE`s to unit `*` are directed to `stdin` and `stdout`. You can `READ` and `WRITE` to units 5, 6, and 0 without an `OPEN`. They are preconnected to `stdin`, `stdout`, and `stderr`, respectively. If you `OPEN` and write to any other unit number `#` without specifying a file name, a default name of `fort.#` will be used.

16.8 Performance Tuning for C and FORTRAN

16.8.1 Optimization

Using the compiler **`-O`** (upper case) options can improve program execution speed by factors of 3 or more, depending on the application, over unoptimized code. Note that your libraries must also be compiled at the same level in order for this to be effective.

Beware that there are some optimizer bugs. You should always do a limited run initially with and without optimizer options, and check your answers.

For production running, use the appropriate hardware-specific optimizations for the systems running the code. These options typically tune for cache sizes, instruction sets, and other internal hardware features, resulting in sizeable speed gains. On some systems this produces an executable that will run only on the targeted architecture.

It is common practice to retain debugger symbol tables in production programs, with only a small speed penalty. You may have to exercise care that the **`-g`** option does not also disable optimization of such production programs. Under IRIX, you must use **`-g3`** to get both optimization and symbol tables.

See the suggested speed optimization options, and vendor documents for details.

Floating Point Errors

You can obtain substantial speed increases on some systems, especially AIX, by disabling the detection and trapping of floating point errors such as overflow, division by zero, and invalid values.

On the systems with the biggest gains, this practice can produce apparently normal, but incorrect, results. For example, 1000./0. can produce the result 1000. It is hardly necessary to point out that this sort of thing can produce surprising physics results! For this reason our recommended options for general use are set to at least detect and report floating point errors.

Qualifiers which force precise trapping of floating point errors are generally only used when tracking down known problems, as they can impose a large performance penalty.

16.8.2 Word Length

It may be tempting to use arrays of short words to ‘save memory’. On previous generations of computers this could also speed execution. On RISC systems there is a big performance penalty for this practice.

The current generation of RISC processors are optimized for 32 and 64 bit operations. Operations on 8 bit or 16 bit words are performed several times more slowly. The processor must extract the necessary data into a longer word, perform the operation, and mask the result back into the original location.

Alignment of variables is important for the same reason. A misaligned 32 bit word requires even more shifting and masking than a 16 bit word, with an even greater performance penalty. If you must combine different length variables in a data structure such as a COMMON, place longer words earlier in the data structure.

16.8.3 Feedback

The speed of a program can be limited as much by memory access as by processor speed. Effective use of memory cache is critical to getting good performance.

Cache usage can depend on the details of the linking process. Arbitrary changes in the ordering of modules in the executable can result in nearly 20% differences in execution speed, for typical physics code. Small changes like switching between static and shared libraries, or modifying a single subroutine call in your code, can result in substantial changes in linking order and hence in performance.

For this reason, some vendors provide mechanisms for setting optimized module ordering in the executable, based on data from a trial run of the application. Under OSF1, use the compiler **`-feedback`** option; see the **f77** or **cc** compiler man page for details and examples.

16.8.4 Inlining

Many compilers provide options for replacing calls to external modules with equivalent inline code, to permit better optimization and reduce subroutine call overheads.

Physics code does not generally benefit measurably from such inlining. Inlining within a library makes the inlined modules nonreplaceable at link time, leading to confusing results and difficult debugging. In our recommended speed optimization options we stop short of the levels that introduce inlining.

16.9 C and FORTRAN Mixed Programming

It is possible, with a little care, to combine C and FORTRAN modules in the same program. Some of the issues that need attention include:

- variable types
- array indexing
- external names
- arguments
- commons
- I/O
- linking

For newly written C programs, you may wish to use the `cfortran.h` header file available in the **cern** product.

The July/Sept 1994 *CERN Computer Newsletter* (217) contains an extensive tutorial by Alfred Nathaniel on C and FORTRAN interfacing, available locally under the heading **Software Development** from the *UNIX Resources* Web page.

If you're programming under IRIX 6, you will need to choose an ABI. Refer to section 16.4.5.

We give here a summary of the techniques used on the Fermilab UNIX systems. In Appendix I you can find an example that illustrates much of the information in this section.

16.9.1 Variable Types

Generally, these variable types are equivalent:

FORTRAN	C
INTEGER*1	char
INTEGER*2	short
INTEGER*4	int
REAL	float
REAL*8	double
LOGICAL	(unavailable)

C strings are zero-terminated, and have no intrinsic length. FORTRAN character variable lengths are given by an internal descriptor. FORTRAN character variables passed to C routines should be copied and zero-terminated before they are used.

The internal representation of FORTRAN LOGICAL variables is usually non-0/0 for .TRUE./.FALSE. respectively, but it is best not to count on this.

16.9.2 Array Indexing

By default C starts indexes at 0 and FORTRAN starts them at 1. C and FORTRAN multiple index ordering is reversed. FORTRAN substring selection appears as the first C string index. See the following equivalence table:

FORTRAN	C
`intv(j)`	`intv[j-1]`
`intv(j,k)`	`intv[k-1][j-1]`
`char(j)(k:k)`	`char[k-1][j-1]`

16.9.3 External Names

By default on Fermilab UNIX systems, the **f77** compiler modifies FORTRAN subprogram and other external names. It forces each name to lower case, and appends an underscore. Thus FORTRAN label `SUBPROG` would become C label `subprog_`. For AIX, make sure the **`-qextname`** option is set (see section 16.4.2).

16.9.4 Arguments

FORTRAN subprogram arguments are always passed as addresses (C pointers). C programs can specify arguments as either pointers or values. FORTRAN CHARACTER arguments are passed as pointers, followed by a set of additional values (not pointers) at the end of the argument list, giving the length of each CHARACTER argument.

C routines can always call FORTRAN routines, with due attention being given to arguments. FORTRAN routines cannot call arbitrary C routines.

16.9.5 Commons

FORTRAN COMMON's are accessible in C as *extern struct*s, with the same name mapping as is used for entry points.

FORTRAN	C
`COMMON /FOO/ I`	`extern struct { int i ; } foo_ ;`
`K = I`	`k = foo.i ;`

It is best to keep your FORTRAN COMMON variables aligned on natural boundaries[1], in order to avoid potential padding words which may be inserted differently by various FORTRAN and C compilers. You get natural alignment easily by putting longer variables before shorter variables in the COMMON.

16.9.6 I/O

Mixed C/FORTRAN I/O to the same file is not advisable. Mixed C/FORTRAN I/O to `stdout`, where `stdout` is the terminal, will usually work reasonably well, making debugging easier.

16.9.7 Linking

The easiest and safest way to link C/FORTRAN programs is to use the **`f77`** command, which automatically includes both C and FORTRAN run time libraries. If you insist on linking with the **`cc`** or **`ld`** commands, remember to add the options:

```
-lF77 -lI77 -lm
```

16.10 Executing a Program

Once you create an executable, you run it the way you do a normal UNIX command, that is by typing its name followed by appropriate options or parameters.

1. Keep all ***n***-byte variables' addresses an exact multiple of ***n,*** for example 0, 4, 8,... for a 4-byte quantity.

You must be aware that if you have not included "dot" (`.`) in your path, whenever your executable is in a directory not explicitly included in your path, you will need to prefix the executable name with **`./`** to run it. This was also mentioned in section 9.2 under *PATH*.

16.11 Debugging

16.11.1 FORTRAN Source Code Analyzer: FLINT

FLINT is a FORTRAN source code analyzer, a proprietary product of IPT. At Fermilab, **FLINT** is available for SunOS 5 and IRIX 5. The **FLINT** User Manual for UNIX is available on the Web (document number PU0099) and on-line in the file `$FLINT_DIR/MANUAL` wherever **FLINT** is installed and setup.

FLINT provides several analysis features:

- local and global analysis
- cross reference tables
- program statistics
- calling trees

FLINT reports the following error conditions:

- inappropriate arguments passed to functions or subroutines
- inappropriate library calls
- inconsistencies in common-block declarations
- non-portable code
- type usage conflicts across different modules
- unused functions, subroutines and variables
- variables which are referenced but not set

FLINT Options and Files

Use the **`-S`** (upper case) option to specify an output file name. The default extensions listed below are used. A separate file will be created for each component of the listing according to the additional option shown:

`lnt`	Analysis output
`xrf`	Xref table (**`-x`**)
`tre`	Call tree (**`-t`**)
`stt`	Statistics (**`-s`**)

Use the **`-L`** option to create a `*.lbt` **FLINT** library, containing all subroutine interface information needed for subsequent call usage tests. This library may be specified as **FLINT** input in place of the original source.

Use the **`-B`** option to create a `*.fdb` database file, containing all the symbol information needed for subsequent cross reference and call tree analysis. This database file may be specified as **FLINT** input in place of the original source.

Lowercase options may be combined. Use a double dash to disable an option (e.g., **`--w`**); this is useful for disabling a default option.

Uppercase options take parameters (with or without a space between the option and the parameter, depending on the option) and do not combine.

When original source is not available, `*.lsh` ASCII files may be created to describe calling sequences. The file `$FLINT_DIR/unixlib.lsh` describes the standard UNIX system calls.

FLINT Usage Examples

To see the standard analysis warnings for Isajet, enter:

```
% flint $ISAJET_DIR/isajet.f
```

To see a calling tree of the Isajet program, without warnings, enter:

```
% flint -t -Tcondensed --w $ISAJET_DIR/isajet.f
```

To see an options summary, enter:

```
% flint "-?"
```

16.11.2 dbx

dbx is a utility for source-level debugging and execution of programs written in C, C++, and FORTRAN**. dbx** allows you to trace the execution of a specified object file. You can step through a program on a line-by-line basis while you examine the state of the execution environment.

Programs compiled with the **`-g`** option of **cc** (and other compilers) produce an object file. This object file contains symbol table information, including the names of all source files that the compiler translated to create the object file.

dbx also allows you to examine *core files* via its **`where`** command. A core file contains the core image produced when the object file was executed, providing information about the state of the program and the system when the failure occurred. A core file named `core` is produced by default.

dbx commands can be stored in a start-up `.dbxinit` file that resides in the current directory or in your home directory. **dbx** executes these commands just before reading the symbol table.

There are some UNIX tools which provide a more sophisticated interface to **dbx**. See your local system documentation for information on GUI-based **dbx** tools. The product **ddd** (originally an interface for **gdb**) works as a front end for **dbx** in its more recent releases. It is currently available at Fermilab as part of the **gcc** product, but we expect to release it as a separate product soon. See the *DDD User's Guide* on the Web, document number DS0230.

Running dbx

To invoke **dbx**, enter the following command:

% dbx [options] [*object_file* [*corefile*]]

where ***object_file*** is the name of the file you want to debug.

Once **dbx** is running, you should see the **`(dbx)`** prompt. At this point you can start issuing **dbx** commands.

Commands

There are many **dbx** commands, all described in the man pages. Some of the basic commands are **`run`**, **`where`**, **`print`**, **`stop`**, **`list`**, **`cont`**, and **`quit`**:

run [*arguments*]	Begin executing the object file, passing optional command-line ***arguments***. The arguments can include input or output redirection to a named file.
where [*n*]	List all active functions on the stack, or only the top ***n***.
print [*expressions*]	Print the values of one or more comma-separated ***expressions***. To print values of two-dimensional FORTRAN array elements use the format: **print *array_name*[1,2]**
stop *restriction* [if *cond*]	Stop execution if specified ***restriction*** is true. Restrictions include (this is a partial list): **at** (source line) ***n*** **if** ***cond*** **in** (procedure or function) ***func*** ***cond*** (condition) is a Boolean expression; if it evaluates to true, then execution is stopped.
list [*n1* [,*n2*]] or **list *func***	List the source text between lines ***n1*** and ***n2***, or on lines surrounding the first statement of ***func***. With no arguments, list the next ten lines.
cont [at *n*] [sig *signal*]	Continue execution from the point at which it was stopped if no arguments. Resume from source line ***n*** or, if a ***signal*** name or number is specified, resume process as if it had received the signal.
status [> *file*]	Show active **trace**, **stop**, and **when** commands
delete [*n*]	Remove traces, stops, and whens corresponding to each command number ***n***, given by **status**. If ***n*** is **all** remove all.
quit	Exit **dbx**.

Example

You may want to start by using **dbx** to set some break points within your code. To step through your code at the very beginning, you need to stop in the `MAIN` routine if you are debugging an object file created from FORTRAN source code (stop in `main` if your source is in C language). For example, you would type:

```
(dbx) stop in MAIN
```

Now you can issue the **run** command to start execution of your object file. You will get the process id and the name of the object file being executed.

```
(dbx) run
```

At this point, you may use the **list** command for the first 10-line listing of the source code:

```
(dbx) list
```

Use the **stop** command to set break-points at various lines or procedures within the object-file:

```
(dbx) stop at 10
(dbx) stop in sub123
```

```
(dbx) stop in sub456 if i == 24
```

The execution will stop in the example above at line 10, or in subroutine `sub123`, or in subroutine `sub456` when `i` is equal to `24`. To continue execution at any point in your debugging, issue the **`cont`** command:

```
(dbx) cont
```

To restart your debugging session, issue the **`rerun`** command:

```
(dbx) rerun
```

To exit **dbx**, type **`quit`**:

```
(dbx) quit
```

Usage Note

A user reports that when using **`dbx`** ***object_file*** **`core`** he has found it useful to turn on all but one of the IEEE arithmetic traps, in order to stop execution when the arithmetic fault occurs (instead of continuing with some default action and then reporting that the following IEEE arithmetic flags had been set). He located a spurious division by zero in this manner. The **`f77`** man page for Solaris describes the necessary flag value on the **`f77`** command line: **`-ftrap=%all,no%inexact`**. We have not researched this for the other UNIX flavors.

16.11.3 gdb

gdb, a GNU product, can do four general types of things to help you debug your programs:

- Start your program, and indicate anything that might affect its behavior.
- Make your program stop on specified conditions.
- Examine what has happened when your program stops.
- Modify your program, allowing you to experiment with correcting one bug and go on to find another.

You can use **gdb** to debug programs written in C or C++.

You can also debug programs written in FORTRAN, although **gdb** does not yet support entering expressions, printing values, or similar features using FORTRAN syntax. Furthermore, it may be necessary to refer to some variables with a trailing underscore.

See the document *Debugging with GDB*, document number PU0172.

16.11.4 purify

purify is a commercial product that detects memory corruption and finds memory leaks in your executable programs. **purify** is currently available on FNALU for Solaris. The command to run it is **`purify`**.

See the man pages for information on its syntax, options and uses.

16.11.5 CASEVision

CASEVision is an advanced debugging environment for SGI. It is currently available on both of the SGI interactive nodes on FNALU. The command to run CASEVision is **`cvd`**.

See the man pages for more information. Additionally, documentation is available through the SGI on-line documentation tool **insight** (see section 3.1.3).

Chapter 17: The make Utility

The UNIX **make** utility is a tool for organizing and facilitating the update of executables or other files which are built from one or more constituent files. Although **make** can be used in a wide variety of applications, in this chapter we concentrate on its use in the area of software development. We describe how to define relationships between source, object, library and executable files for use by **make**, and how to invoke **make** in its simplest and slightly more complex forms.

17.1 An Overview of the make Utility

make is a command execution utility. You can use it to essentially automate any task in which one or more "target" file(s) requires updating via a shell command when changes have been made to any of its "required" files (files from which the targets are built). There is some preparation to do, but once that is complete, all you do is enter the `make` command!

make compares the modification dates of target files to those of their required files. For each file that needs updating, **make** issues the predefined update command(s) for the file. For example, if file A is a required file of file B (the target), and if file A has a more recent "last modified date" than file B, then **make** *re-makes* file B by issuing the specified update command(s). Target files that are found to be more recent than all their required files are skipped over.

make is especially useful in long-term software development projects that involve large numbers of source files, libraries, and executables connected by a complex set of relationships. You can use it with any programming language whose compiler can be run with a shell command.

make obtains information about the files, their relationships, and the specific update commands from one or both of the following:

- a specially formatted, user-supplied control file called the *Makefile* (see section 17.2)
- **make**'s set of built-in default rules (see section 17.6)

In preparing to use **make**, you generally need to write a Makefile. The Makefile defines the relationships among the constituent and target files of your project by listing the required files for each target file, and stating the shell commands that must operate on the required files to create or update the target(s). **make** implicitly treats all required files as targets, in an iterative manner. A file listed as a required file in one definition statement may also explicitly appear as a target in another statement. For example, in a program, typically the executable file (the final target) is created from object files, which are in turn created by compiling source files. Once you have a working Makefile, you just run the **make** program to perform all your updating tasks.

The **make** man pages give a full description of the command, its options and features, as well as the format and usage of the Makefile.

We have included two examples in Appendix I that illustrate many of the points discussed in this chapter.

17.2 The Makefile and its Components

The Makefile is a blueprint that you design and **make** uses to create or update one or more target files based on the most recent modify dates of the required files. The **`make`** command line syntax remains quite simple in the case where a Makefile is used:

`% make [-f` ***makefile_name*** **`]`** **`[`*****other options*****`]`** **`[`*****targets*****`]`**

If the **`-f`** option is not used, **make** checks for a Makefile of a particular name: it first looks in the current directory for a file named `makefile`, then for `Makefile`. If the Makefile is still not found, **make** looks in a couple of other places (see the man pages). In order to keep things simple and standard, we recommend that you always use the filename `Makefile` for your Makefile(s). Following this convention, you'll have only one Makefile in a directory. Your Makefile can contain instructions for building many different targets. When executing **make**, you can specify the desired target(s) on the **`make`** command line.

We can categorize the types of statements that can go in a Makefile as follows:

macro definition	A macro is a name that you define to represent a variable that may occur several times within the Makefile.
target definition	A target definition lists the target file, its required files, and the commands to execute on the required files in order to produce the target. (You can opt to specify the totality of this information in separate target definitions.)
suffix rules	Suffix rules indicate the relationship between target and source file suffixes (filename extensions). For example in FORTRAN, object files (`*.o`) are created from source files with a suffix of `f` (i.e. `*.f`). If no source file is explicitly given on a target definition line, **make** uses suffix rules to determine what source file to use to produce the target.
suffix declarations	Suffix declarations are lists of suffixes (file extensions) used in suffix rules.

Each new line in the Makefile starts a new definition, except that in target definitions:

- shell commands with leading tabs are part of the previous definition (a standard Makefile format that you need to recognize, but that we suggest you avoid using for reasons explained in section 17.2.2.)
- shell commands can be continued in standard UNIX format, using a trailing backslash (\)

Blank lines are permitted between definitions. Comments can be included after a pound sign (`#`). To use a literal pound sign, precede it with a backslash, i.e. `\#`.

17.2.1 Macros

A macro is a name that you define to represent a variable that may occur several times within the Makefile, or that needs to be updated frequently. Macros make maintenance of your Makefile much easier. They are commonly used to define settings, platform-specific commands, lists of required files for a target, lists of command options, and so on. **make** reads all the macro definitions before executing any commands. It is often convenient to put all the macro definitions at the head of the Makefile, but this is not necessary.

Format and Usage

Macro definitions are of the form:

macro_name* = *value

where ***macro_name*** is the name you want to use in place of the longer ***value.*** Everywhere in the Makefile that ***macro_name*** is found, **make** substitutes ***value***. For portability, if ***value*** contains any blank spaces that it is supposed to have, the ***value*** string must be enclosed in quotes in the definition statement.[1] Backslashes can be used to continue the same line; you cannot put a new line in a macro value.

Once a macro is defined, you refer to its value in the form `$(`***macro_name***`)`. If ***macro_name*** is a single character, you can omit the parentheses.

As an example, let's define a macro FFLAGS to set some FORTRAN default options to use with the **`f77`** command:

```
FFLAGS = "-O2  -w -Olimit 1500 -nocpp "
```

You can now refer to the value of FFLAGS within the Makefile in the form `$(FFLAGS)`. For example, you might include a target definition line like the following (the format is explained below in section 17.2.2):

```
foo : foo.f ; f77 -o foo $(FFLAGS) foo.f
```

Special Macros

To be sure that **make** uses the standard, portable Bourne shell, always include in your Makefile a macro of the form:

```
SHELL = /bin/sh
```

Some versions of **make** default to your current interactive shell if you don't include this explicit SHELL macro in your Makefile. The standard Makefile format that we describe in this chapter assumes **sh** as the command interpreter.

Macro Sources

Macro definitions are similar to and can take default values from environment variables. **make** gets additional macro definitions from the following sources, and applies them in the order shown:

1) currently defined environment variables

2) built-in **make** rules

3) definitions in the Makefile

4) definitions on the command line

In other words, this list is in order of reverse precedence; a value from a source later in the list overrides a value applied from an earlier one. Using the **`-e`** option on the **`make`** command line swaps the first two; see section 17.3.1.

---

1. Be aware that this quoted value is what **make** hands to the shell. The shell then hands the macro, e.g., `$(FFLAGS)` to a program to execute. If you want leading or trailing spaces in ***value*** to be included in the command, specify the macro in double quotes in the command statement, e.g., `"$(FFLAGS)"`.

Symbols Used in Macros

`$$` maps to a literal dollar sign

`$*` is used in suffix rules (see section 17.2.3); it refers to the filename without the suffix

`$@` is the current target make is processing

`$<` is the implied source in a suffix rule

17.2.2 Targets

Targets are the files that you want to update or create. A complete target definition includes the name of the target, the files needed to build it (its required files), and the commands that must be executed to recreate the target.

Format

The standard Makefile format using Bourne shell conventions calls for:

- a space between listed targets
- a colon (`:`) between the last target and the first required file
- a space between listed required files
- a semi-colon (`;`) after the last required file if commands follow on the same line
- a semi-colon (`;`) between the successive listed commands

A simple target definition with a single target, required file, and command can be written in the form:

target* : *required_file* ; *shell_command

or, using a more traditional format, listing the command on the next line (note that the semicolon (`;`) is omitted here):

target* : *required_file

`{tab}`*shell_command*

In this traditional Makefile format, the commands beyond the first line of a definition must have leading tabs, and there must be no intervening blank lines.

There are two reasons to avoid this second format: first, when working with the Makefile it is hard to see the difference between a tab and blanks, and secondly, some editors change tabs to blanks (or vice versa), which causes problems (neither **emacs** in default mode nor **vi** changes tabs to blanks). You can check your file to see if it contains tabs or blanks by running the command:

`cat -tev` *filename*

We propose as an alternative that you use the backslash character (****) to continue the single definition line down as many physical lines as you have to go. Also, note that under AIX the standard “tabbed” format can be unpredictable; it is therefore safer for *several* reasons to use our suggested format on this platform.

This “safer” format which we suggest if the entire definition doesn’t fit on the first line is:

***target* : *required_file*; **

shell_command

A target definition line can contain more than one of each element type. Or, it may contain only two of the three element types. A more complex definition in our suggested format looks like:

```
target_1 target_2 ... : required_file_1  required_file_2 ... required_file_n \
required_file_n+1 ...; \
shell_command_1 ; shell_command_2 ; shell_command_3 ; ... ; \
.
.
.
... ; shell_command_m
```

If there is more than one target (e.g. ***target_1 target_2***) in one definition, the commands are attempted separately for each target.

If you find it easier, you can list multiple required files for a single target in separate target statements. However only one statement for a given target can include commands, and therefore must include all the commands. Here is an example of this alternative format:

```
target : required_file_1
target : required_file_2
target : required_file_3
target : ; shell_command_1 ; shell_command_2 ; shell_command_3 ; ...
```

It can be confusing if you separate a series of statements like this from one another in the Makefile; if you use this format, keep the statements together!

Usage

The relative modification times of the ***target*** and the ***required_file(s)*** determine whether the listed commands will be executed. If the target file is found to be missing or to be older than any of its required files, **make** executes the commands to rebuild it. **make** treats the required files iteratively as targets, whether or not they are explicitly listed as targets in subsequent target definitions, and rebuilds them as necessary before rebuilding the final target.

17.2.3 Suffix Rules

A suffix is essentially a file extension. A suffix rule defines the relationship between target and required files by their file extensions. A suffix rule is much like a target definition except that it uses implied rather than explicit file names for target (output) and required (input) files.

A suffix rule is of the form:

.*insuffix* .*outsuffix* : ; *command*

The dots are really part of the suffixes themselves. As an example, assume you have a set of target files to rebuild whose filenames are all of the form `*.b`. The required files for these targets have filenames of the form `*.a`. Define the suffix rule:

`.a.b : ;` ***command(s)***

make expands this to the following target definition for all files ending in `.a` in the current directory:

filename **`.b : `** ***filename*** **`.a ; `** ***command(s)***

If you define suffix rules specifically for intermediate files in a process, you still need to include a rule for the final and original files, for example, if you define:

```
.tex.dvi: ; latex $*  #latex cmd makes .dvi files from .tex files
.dvi.ps: ; dvips $*   #dvips cmd makes .ps files from .dvi files
```

You still need to provide the rule:

```
.tex.ps: ; latex $*; dvips $*  #make .ps files from .tex files
```

make is not sophisticated enough to do the transitive closure on suffix rules.

☞ Note that suffixes don't have to start with a dot (`.`).

17.2.4 Suffix Declarations

Any suffix that you use in a suffix rule must be listed explicitly in a `.SUFFIXES` declaration in the same Makefile unless it is included in **make**'s built-in suffix declarations (see section 17.6). A suffix included in the built-ins can also be included in a suffix declaration in the Makefile.

Suffix declarations are really target definitions for a special target named `.SUFFIXES` and they contain no commands.[1] They are of the format:

`.SUFFIXES : .a .b`

where **`.a`** and **`.b`** are suffixes. This example suffix declaration would allow you to include the suffix rule from 17.2.3 in your Makefile:

`.a.b : ;` ***command(s)***

You may combine all the suffixes you use in the Makefile into one `.SUFFIXES` declaration, or group them into separate statements.

Suffixes that you declare add to the built-ins; they do not replace them.

17.2.5 Control Files within a Makefile

You may encounter situations where it is useful to pipe input or output of one command to another within a Makefile. You can echo a small control or data file within the Makefile, rather than maintaining a separate external file. In the following target definition example from an Isajet Makefile, several control statements are echoed into the **patchy** utility in order to extract `isaint.f` from the `isajet.car` patchy library:

```
isaint.f : isajet.car ; \
(echo "+USE,*ISAJET,$(MACHINE).              "; \
 echo "+USE,INTERACT.     INTERACTIVE PATCH  "; \
 echo "+EXE.                                 "; \
 echo "+PAM,T=C.                             "; \
 echo "+QUIT.                                ";)\
| ypatchy isajet.car  isaint.f  \& genint.lis .GO
```

1. Most versions of **make** have other special targets (sometimes called magic targets) besides `.SUFFIXES`. These special target names always start with a dot (`.`).

17.3 Running make

17.3.1 General Usage

The **make** utility is invoked with the `make` command. The command syntax is:

`% make [`*options*`] [`*targets*`]`

Several ***options*** are available, and are described in the man pages for **make**. We provide a list of some of the commonly used options:

-n	Preview the commands, don't execute. Very useful for testing.
-d	Debug; list the operations used and why ("read **make**'s mind"). Available on all platforms except OSF1.
-e	Environment variables override built-ins.
-f *makefile_name*	If your Makefile has a name other than `makefile` or `Makefile`, use this option followed by the file's name to identify it (leave a space between the option and the filename).
-p -f /dev/null [|less]	Print the built-in rules (see section 17.6).

The ***targets***, as mentioned earlier, are the files that you want to create or update. **make** searches the Makefile to find a target definition statement for each target listed on the command line.

You can include macro definitions on the command line which get applied after the assignments made in the Makefile. For example:

`% make "CC = gcc"` *target*

17.3.2 Usage without Specifying Target

When **make** is invoked without a specified target, the first non-suffix target in the Makefile is used. The command is simply:

`% make [`*options*`]`

For larger products, it is standard practice to name this target *all* in the Makefile, and in the list of required files to list the individual targets which together actually produce the full product. For example, the first Isajet target definition is:

```
all : isadecay.dat \
      isatext.doc  \
      isajet.a     \
      isaint       \
      isasusy
```

Notice that here no commands are listed. They would appear in the subsequent target definitions.

17.3.3 Usage without a Makefile

The extensive built-in rules let you use **make** quite effectively without having your own Makefile. Section 17.6 provides a brief explanation of the built-in rules. **make** will look for any file whose name matches that specified on the command line and which has a file extension that identifies it as a reasonable source. For example, to produce the executable `foo` (the target) from a `foo.c` or `foo.f` source that exists in the current directory, enter:

```
% make foo
```

make will look for the C or FORTRAN file as the source file for this target. Taking the FORTRAN program and no options as an example, this command is equivalent to (see section 16.2.3):

```
% f77 -o foo  foo.f
```

17.4 "Housekeeping" Targets

It is common practice to have a "housekeeping" target which removes stray files from the working directories. Typically you would run **make** on this target after you've completed the **make** operation on your principle target(s). It is conventional to call this target *clean*. Here is an example which removes unnecessary generated files from several different directories. The target definition has no required files:

```
clean: ; \
  rm  -f *.bak    ;\
  rm  -f *.lis    ;\
  rm -rf Maketemp ;\
  cd example/isaplt ; rm -f *.lis*   ;\
  cd  ../jet        ; rm -f jet.log*
```

You need to determine what stray files will be generated in your case, and define your commands accordingly. Run **make** on the *clean* target by entering:

```
% make clean
```

You may wish to define different levels of housekeeping targets. One that clears out everything, leaving only the original files you had before running **make**, is often named *clobber*.

17.5 Portability

It is desirable for your Makefile to be portable across different UNIX platforms. Why might this be a problem to implement? As mentioned earlier, **make** does all macro processing before any commands are executed. Therefore environment variables set in shell scripts executed by **make** have no effect on **make**'s macro definitions. And standard **make** doesn't support conditional macro definitions. So, how can you write a portable Makefile?

A Solution

1) Create a script for setting environment variables.

2) Have **make** run this script ("source" it; see section 4.4) from within the Makefile.

3) After it runs the script, have **make** rerun itself with a different target and the original command line options. A *$(MAKE)* macro which causes **make** to rerun itself is a standard feature.

An example of this technique follows.

Example

Create a portable shell script (named, for example, `Makeenv`) which sets appropriate environment variables for the Makefile. Here is a simple `Makeenv` script which defines the macro `F77` based on the current platform as determined by the command **`uname -s`**:

```
export F77 MACHINE

case `uname -s`  in
IRIX)
        F77="f77  -O2  -w -Olimit 1500 -nocpp "
;;
OSF1)
        F77="f77  -O1  -w -Olimit 1500 -nocpp -static "
;;
esac
```

The Makefile is below. Run **make** with the target `isaint`. The Makefile runs `Makeenv`, then **make** reruns itself with the target `do_isaint` and the correct `F77` value:

```
isaint    : isaint.f ;  . Makeenv; $(MAKE) do_isaint
do_isaint : isaint.f ; $(F77) isaint.f  -o isaint
```

Other Utilities

There are other utilities available for more complicated cases:

- **gmake** (Gnu make), part of the Fermilab **gtools** product, has some nice portability features, and supports other advanced features like parallel compilation on multiprocessor systems.
- There are preprocessors for building locally tailored Makefiles in very sophisticated ways, including **gnu configure**, **premake**, and **imake**.

17.6 make's Built-in Rules

make comes equipped with a long list of built-in defaults to make your job easier. You are free to override any of them in your Makefile. The defaults fall roughly into four categories:

1) macros that reflect your current environment variables

2) macros that define standard compilers and options

3) suffix rules for finding required files when building targets

4) a list of known suffixes

To get a listing of all the built-in macros and rules, enter the command:

`% make -p -f /dev/null [| less]`

Depending on your platform, there may be nearly a thousand lines of definitions, so you might want to pipe this to **less**, or redirect the output to a file.

17.7 A Few Caveats...

1) Recall that in the traditional Makefile target definition format successive commands are entered on successive lines, each starting with a tab. Be aware that each of these command lines runs in a different shell. Two important implications of this are:

   a) if you have issued a change directory (**`cd`**) command, it does not carry over to the following line(s)

   b) environment and shell variables do not carry over to the following line(s)

The format which uses a single logical line for the entire definition avoids this problem.

2) If you use non-shell commands (for example **`ls`**) in definition statements, be aware that the output may vary from platform to platform. For this reason it is best not to rely on the specific output format of these commands.

Chapter 18: Code Management

This chapter introduces the recommended code management solution for UNIX, **CVS** (Concurrent Versions System). We also introduce an alternative that is currently being used by a couple of Fermilab experiments, **UCM** (UNIX Code Management)[1]. Both packages use **RCS** (Revision Control System) as the underlying protocol. We provide basic information only, and refer you to the complete manuals for these utilities for detailed information.

RCS provides a version control system with which you can record the history of your source files. An **RCS** file contains multiple revisions of text, an access list, a change log, descriptive text, and some control attributes. Only the differences between versions are kept.

CVS and **UCM** implement the **RCS** features differently, but both assume availability of **RCS** commands. **CVS** allows concurrent development, whereas **UCM** was created to replace **CMS** (a VMS source code management system) and implements the same sequential development philosophy of that system. Both systems provide easy extraction of either a release version or the latest version, and they allow you to create tags for release versions of the software.

See *Code Management* under the **Software Development** heading on the *UNIX Resources* Web page for more information. Alternatively you can search the on-line product documentation database for **CVS**, product number PU0189, or **UCM**, product number PU0254.

18.1 CVS

The **CVS** product allows many programmers to work on the same code simultaneously, each in his or her own directory, and it merges the changes when they are finished. There is no built-in mechanism to prevent concurrent development.

CVS stores all files in a central repository, a directory populated with a hierarchy of files and directories. The files are organized in modules, where a module is made up of one or more files, and can include files from several directories. It is typical to define one module per project. Although the structure of the repository and modules file interact with your build system (e.g., Makefiles), they are essentially independent.

18.1.1 Accessing CVS and Obtaining the Manual

To set up the **CVS** product:

```
% setup cvs
```

1. **UCM** is currently supported only within the experiments using it; although it is provided as a **UPS** product, it is not centrally supported at Fermilab.

The **CVS** document (in postscript format) can now be found in the directory pointed to by `$CVS_DIR/doc`. You can also find documentation on the Web for **CVS**, product number PU0189.

CVS points to the editor defined in your *EDITOR* variable for entering log messages (see the **`cvs commit`** command below). If it is not set, the editor defaults to **vi**.

18.1.2 Basic CVS Commands

Normally you never access files in a repository directly; you use the **CVS** commands to get your own copy of files. The common commands and their functions are listed below (we do not provide information on options here; see the document referenced above):

`%` **`cvs import`**	installs new release of source in **CVS** repository the first time
`%` **`cvs co`** ***module***	creates new directory called ***module***, populates it with source files; this allows you to "checkout" your own working copy of the source ***module***
`%` **`cvs checkout`** ***module***	equivalent to **`cvs co`** ***module***
`%` **`cvs commit`**	commits changes you have made; opens editing session for entry of log message
`%` **`cvs rtag`** ***release_number module***	tags a release
`%` **`cvs export -r`** ***release_number module***	extracts the specified release without **CVS** administrative directories

18.2 UCM

UCM is a replacement for **CMS** which has been widely used at Fermilab on VMS systems. To enforce sequential development, it uses a reservation system to ensure that a second programmer cannot change a module that another programmer is currently editing. **RCS** (Revision Control System) is a readily available UNIX product which is very similar to **CMS** except that it operates on individual files rather than on libraries. The component of **UCM** which actually replaces **CMS** is called **UVM** (UNIX Version Management), and it is an interface to **RCS**. **UCM** also contains tools which extend its use beyond the scope of simple source code management.

18.2.1 Accessing UCM and Obtaining the Manual

To access **UCM**, you first need to set it up.[1] Include in your login files or enter:

`%` **`setup ucm`**

1. On some systems, you will have to setup **CVS** to access **RCS** before using **UCM**. This is not necessary where FUE is installed.

UCM, product number PU0254, is fully documented on the Web. Once the product is set up, you can also find the **UCM** manual in the directory pointed to by `$UCM_DIR/doc`. In addition to providing more in-depth information on the concepts and commands introduced here, it provides examples of updating include file directories, updating an object library, and maintaining release versions.

18.2.2 Basic UCM Commands

You use **UCM** by executing commands provided by two of its constituent utilities, **uvma** and **uvmi**. The **uvma** commands are used for creating and maintaining libraries, and the **uvmi** commands provide information to the user. These two command sets can be invoked directly through **`uvma`** and **`uvmi`**, or indirectly through the generic command **`uvm`**.

Typing **`uvm`**, **`uvma`**, or **`uvmi`** without any commands, arguments or options will display the list of commands available to the corresponding command set.

Some basic **uvma** commands are listed below with brief descriptions of their functions:

`%` **`uvm create library`** *library*

creates a **UVM** source code library

`%` **`uvm create element`** *library/file*

creates a **UVM** element in the library

`%` **`uvm fetch`** *library/file* — creates a copy of the specified **UVM** element(s) in the user's local area

`%` **`uvm reserve`** *library/file* — extracts a working version of the latest revision of the specified **UVM** element(s) into the user's local area and locks the RCS file[1]

`%` **`uvm replace`** *library/file* — creates a new revision of the specified **UVM** element(s) from working file(s) in the user's local area

`%` **`uvm tag`** *library/file* **`-n`** *tagname*

assigns a symbolic tag name, ***tagname***, to the specified revision of the **UVM** element(s)

`%` **`uvm group`** *library/file* **`-g`** *groupname*

adds a **UVM** element to a group specified by the **`-g`** option

---

1. Elements must be reserved before they can be changed.