DCD User Guide 3.4

# Fermi UNIX Environment DCD Products User Guide


Matthew J. Wicks


January 19, 1993

# 1 Intoduction

The Distributed Computing Department (DCD) provides and supports several products that are part of the core Fermi UNIX[1] environment. Some of these utilities have been written in-house, while others are public-domain software that has been tested on the four major UNIX platforms (i.e. IRIX[2]4.0.5, SunOS[3]4.1.2, ULTRIX[4]4.2, and AIX[5]3.2). Requests for porting to other platforms as needed at Fermi Lab will be considered.

The purpose of this Guide is to provide the necessary information for the end user to configure and use the utilities. Some of these utilities are system administration utilities and thus the end user is the local system manager, while other utilities are geared for the general user. In general, any configuration of the utilities must be performed by the local system manager, although some tools can be configured individually by the user.

The utilities provided by DCD that are documented in this guide currently are divided into three products, *funkern, futil,* and *systools* which are intended to be declared to the ups[6] database on the system. In addition, DCD provides the *fulib* product, which provides C-callable library routines. This product is required to build *systools, funkern* and *ups* and only needs to be installed on the system in order to re-build these products. DCD also supports a general purpose backup facilty, *fmb* (Fermi Modular Backup). This product is documented in DCD Design Note 42.0.

## 1.1 Retrieving and Installing these Products

All of these products are available using upd[7] from the central distribution machine FNSG01. The following indicates the latest versions of the products:

1. funkern - v4_0
2. futil - v4_0
3. systools - v4_0
4. fulib - v4_0

After retrieving the tarfile for the product (using upd copy) you may unwind it in any location. A standard location is `/usr/products/`*product_name/version_number*. However, you may place it in an alternative location.

[1]UNIX is a registered trademark of UNIX Systems Laboratories
[2]IRIX is a registered trademark of Silicon Graphics, Inc.
[3]SunOS is a registered trademark of Sun MicroSystems
[4]ULTRIX is a registered trademark of Digital Equipment Corporation
[5]AIX is a registered trademark of International Business Machines
[6]Documentation on ups is contained in pn426.
[7]Documentation on upd is contained in pn435.

Many of the files for the *futil, funkern* and, *systools* products eventually must be copied to the `/usr/local` area. (In the case of *funkern,* you may tailor the product to install the files in another location.) A script, named `current`, has been provided with the product that will perform the file copies as well as insure that the files have correct ownership and permissions. In addition, the script will save any critical configuration files that were used in a previous version of the product. This script is executed automatically when you make the product current.

**You must be logged in as root when the current script is executed (i.e. when declaring the product as current) since it installs some setuid root programs and changes the ownership of various files. In the case of *funkern,* you do not need to be root, but you do need to have write permission in `/usr/local/bin` (or the install area specified in the tailoring of *funkern*), in the area manual pages are installed specified during the tailoring of *ups,* and in the ups database area.**

## 1.2 Re-building the Products

As distributed, the products should not need to re-compiled. However, source code has been provided in the event that you need to re-build a product. **After re-building a product, you need to execute the current script in order to install the appropriate files.**

The following is an example of how to re-build the *funkern* product. Other products are re-built in a similar manner.

1. setup -b funkern
2. cd $FUNKERN_DIR/src
3. make clean
4. make uninstall
5. make install
6. $FUNKERN_DIR/ups/current or
   ups declare -c funkern v3_0

# 2 FUNKERN

The first DCD product *funkern,* (pronounced foo-kern with a silent N) is a small set of utilities that forms a kernel for other products. Normally, it is one of the first products installed on a system because other products require utilities in the *funkern* product. Before installing *funkern* you should make sure a `local` login and group exists on your system (UID=1532, GID=4526). Note: The requirement of a `local` account and group may not be required, based on the tailoring of the product.

When *funkern* is made current it copies its executables, by default, into `/usr/local/bin` and makes them owned and grouped to `local`. This may be changed by tailoring the *funkern* product before making it current. See the ups documentation for information on how to tailor a product.

Currently, *funkern* consists of two utilities `funame` and `dropit`. However, additional utilities may be added in the future.

## 2.1 funame

The `funame` program is a Fermi implementation and enhancement of the standard UNIX `uname` utility. It provides the standard `uname` options but guarantees that the `-s` option returns the name of the underlying operating system and not the node name. It also guarantees that the `-n` option (which is the default output) returns the short version of the node name (i.e `fnsg01` instead of `fnsg01.fnal.gov`).

In addition, `funame` provides the following options:

**-h** Print the nodename in *long* format. Long format is defined as the full machine name including Internet domain extensions (i.e. fnsg01.fnal.gov instead of fnsg01). The .fnal.gov domain extension is hard-coded into the source code for funame. If a different domain extension is required, a #define must be changed at the beginning of the source code and then funame must be re-compiled. Recompilation requires the presence of the *fulib* package.

**-i** Print the IP address of the machine.

## 2.2 dropit

The `dropit` program is a utility designed to make it easier to remove elements from your `$PATH` or other shell variables consisting of colon separated fields. You can also perform operations on shell variables consisting of fields separated by some other character by using the `-i` option in dropit. This utility is used by many products in the unsetup script.

The following is an example of how `dropit` could be used in a unsetup script. The complete details of using `dropit` are provided in the manual page.

This example will show how to remove all instances of the `cern` product from your $PATH variable in either a *csh* or *sh/ksh* script as well as provide an explanation of the script. (Note that currently all ups setup and unsetup scripts must be csh scripts.)

*Sample sh/ksh script*: export PATH=‘dropit /cern/v9‘

*Sample csh script*: set path=(‘dropit -d’ ’ /cern/v9‘)

The *sh/ksh* script uses `dropit` to return a colon separated list of elements from the original `$PATH` variable with all elements containing the string `/cern/v9` removed. Example `$PATH` elements that would be removed would include:

1. /usr/products/*cern/v9*2a/bin
2. /usr/products/*cern/v9*2a
3. /usr/products/*cern/v9*2b/bin
4. /usr/products/*cern/v9*2b/lib

Now in practice you would expect the `$PATH` variable to only contain elements such as `/usr/products/cern/v92a/bin`. However, any element that contains the string `/cern/v9` would not be returned from `dropit`.

In the *sh/ksh* script the string returned from `dropit` is set equal to the `PATH` variable. The contents of this variable are exported.

The *csh* script used the `-d` option of `dropit` in order to have a space separated list returned instead of the standard colon separated list. This option is desired because of the syntax differences in setting the path in csh. However, `dropit` still uses the value of `$PATH` as input. To use a string other than `$PATH` as input refer to the manual page. The alternative input string must still use the colon as the field separator, unless the `-i` option is also used.

# 3 FUTIL

The second DCD product, *futil* is a set of user level utility programs. Some of these programs were written in-house such as `flpr`, `Info` and `vt100` while others are public domain programs that have been tested on the various architectures. Before installing *futil* you should make sure a `local` login and group exists on your system (UID=1532, GID=4526).

## 3.1 flpr

The `flpr` program is a Fermi implementation of the standard UNIX lpr utility. It is used to submit jobs to VAX print queues. Its usage, along with the associated programs `flpq` and `flpk`, are documented fully in the `flpr` manual page and will not be repeated here.

The following precedence (from highest to lowest) is used to determine items such as printer, host, and nicknames:

1. command line arguments
2. environment variables
3. values in $HOME/.flprc
4. values in /usr/local/etc/flp.defaults file
5. values hard-coded into the program

Command line arguments allow the specification of the host with the `-h` option and the printer queue with the `-q` option. In addition, using the `-P` option you can specify a nickname that is a combined host and printer queue. Nicknames are specified in either the `$HOME/.flprc` or `/usr/local/etc/flp.defaults` files.

Environment variables allow the specification of the host via the `FLPHOST` variable and printer queues via the `FLPQUE` variable.

The `$HOME/.flprc` and `/usr/local/etc/flp.defaults` files have the same syntax. To specify a host, include a line with `host <hostname>`. To specify a printer queue, include a line with `queue <queuename>`. To specify a nickname (combined host and printer queue, include a line with `printer <nickname> <hostname> <queuename>`.

More detailed information can be found in the `flpr` manual page.

### 3.2 less

The `less` program is a sophisticated public domain paging program, that many people feel is superior to the `more` paging program. Its use is fully documented in the less manual page.

### 3.3 log/grp commands

There are a series of programs beginning with `log` that print out a specific field in the password file for the given user. In addition, there are a series of commands beginning with grp that print out information in the group file. The log commands take either the account name or uid as an argument. The grp commands take either the group name or gid as an argument. The full list of these commands are:

1. logdir - shows the login directory ($HOME directory)
2. logage - shows the password aging information (if any)
3. logdata - shows the *comment* or GECOS field
4. loggid - shows the GID
5. logshell - shows the login shell, if none is specified in the passwd file (defaulting to /bin/sh) then no value is returned
6. loguid - shows the uid
7. loguser - shows the account name
8. grpname - shows the group name
9. grpnumber - shows the gid
10. grpmembers - shows the members of the group

### 3.4 Info

The `Info` program allows posting of information to the general user community. The permissions of the `/usr/Info` directory determine which users can submit, delete, and purge articles.

`Info` provides a number of different command line arguments in order to add, display, delete, etc. articles. The following text is the help message from Info which documents all the different options

```
Info may take any one of the following arguments as requests:

  -help           requests this help text
  tag(s)          requests presentation of selected articles by tag
  -display tag(s) requests presentation of selected articles by tag
  -anynew         requests message about number of new articles
  -any            requests message about number of articles
  -anynewcode     requests exit status about number of new articles
  -anycode        requests exit status about number of articles
  -new            requests list of new articles
  -list           requests list of articles
  -add            requests prompting to add new article from text file
  -replace        requests prompting to replace existing article from text file
  -insert         requests prompting to insert article from text file
  -Add     tag headline expiredate textfile
  -Replace tag headline expiredate textfile
  -Insert  tag headline expiredate textfile
                  requests corresponding batch operation
  -del            requests the removal of a specific article by tag
  -delete         requests the removal of a specific article by tag
  -del tag        requests the removal of a specific article by tag
  -delete tag     requests the removal of a specific article by tag
  -purge          requests the removal of expired articles

  If none of the above are supplied, the default is -list.

  A tag is a short (1 to 20 character) article identifier.
  A headline is a short (1 to 32 character) description of the contents.
  A expire date is a date (see getdate), 'standard' (or 'std'), or 'never'.
  The expiration date can not be earlier than today.
  A text file is a existing readable readable file from which the text
  for an article is to be copied.

  A new article has a time stamp newer than your $HOME/.Info file.
  If you do not have a $HOME/.Info file then all articles are new.

  The article modification operations (-add, -replace, -insert, -Add, -Replace,
  -Replace, -Insert -delete and -purge) requires that you have permission to
  change the /usr/Info directory.
```

When adding an article (through `Info -add`) the user is prompted for the additional information required. This information includes article name, article headline, and expiration date. The expiration date can be added in a number of formats. For a complete description of valid formats, see the documentation for `getdate`.

In order to purge articles, it is suggested that a cron job be setup for an account that has write permission in the Info directory. The following cron job would purge articles once a week early Sunday morning

```
05 01 * * 0 /usr/local/bin/Info -purge
```

### 3.5 vt100

The `vt100` program currently only works on IRIX and AIX systems. The program creates a vt100 window, using xterm, on the display referenced by the $DISPLAY environment variable. On IRIX systems this variable is set for you by the `fermi.cshrc` when you log in to an SGI graphics terminal. If you give an argument to the vt100 program, after creating the window, it telnets to the specified host. When no argument is given, the window is treated as a login window, and thus the user's .login or .profile is executed.

### 3.6 findcpp

The `findcpp` utility provides all the preprocessor utilities automatically defined by the `cpp` program (C preprocessor) as well as by the `cc` command itself.

`findcpp` is a shell script that creates several C programs in the `/tmp` area, runs these programs, and then removes them. The output of the C programs it creates are the symbols that are automatically defined. (i.e. If you place "#ifdef *symbol_name*" in your C program the C preprocessor will include that portion of the code on this machine.)

### 3.7 getdate

The `getdate` utility is used to convert various time/date specifications into a variety of date formats including the format returned by the UNIX `date` command.

Input specifications can be `normal` date specifications such at 5/12/92 or English descriptions such as `first Monday in November plus 1 day`.

Full details of the utility can be found in its manual page.

### 3.8 cwi

`cwi` stands for copy with include and is a special purpose program to copy the contents of the file, with special provisions to include other files as needed. This program is used by `cmd rm, chmod, kill,` and `renice` to process the data files. It is fully documented in the manual page.

## 3.9 Programs Used by System Administration Utilities

There are several utilities that are part of the *futil* product that are used by the *systools* product. These programs include:

1. getlock - creates a lock file.
2. obtain - obtain a database from FNAL.
3. namefix - change format of the UID and GID databases for use by the system administration utilities.

These programs were placed in the *futil* product because they could be used in other applications and thus should be in an area accessible by the general user. If other products decide that some of these tools are valuable, perhaps they will be moved to the *funkern* product.

#### 3.9.1 getlock

The `getlock` program is a simple locking program that allows two or more programs to cooperate with each other and *lock-out* the other program from performing a certain function. For example, the various utilities in *systools* that modify the password and groups file use this program in order to insure that only one program that modifies these files is run at a time.

The usage of the program is:

```
getlock pid directory filename
```

The `getlock` program now attempts to create a file of the specified name in the specified directory. The contents of the file are the value of pid. When used from a shell script, the special variable $$ is used when calling the program since that is the value of the current process id.

If the file already exists, `getlock` reads its contents to determine the pid value from an earlier successful invocation. If this process still exists, then getlock fails. If the process no longer exists, then getlock succeeds and replaces the contents of the file with the new pid.

Using standard shell conventions, when `getlock` is successful, a value of 0 is returned. A non-negative value is returned for failure.

This locking mechanism doesn't require programs to remove the lock file since when the process no longer exists, the lock file is no longer considered valid.

### 3.9.2 obtain and namefix

The `obtain` and `namefix` programs are used to retrieve a copy of the FNAL global UID and GID databases. The `obtain` program is used to retrieve the program via anonymous ftp from fnal.fnal.gov, while `namefix` changes the output of the file into a format more easily parsed by various shell programs. `namefix` is called by the `obtain` program and thus is not normally used directly by a user.

The `obtain` program allows one of four arguments corresponding to two different formats of both the UID and GID database. The files are placed in the current working directory. If the current copy of the local file is less than a day old, the `obtain` program asks if you want to retrieve a new copy of the file.

The four arguments and corresponding local file names created are:

| | |
|---|---|
| **uid,uids** | unix.uid.list |
| **gid_id,gid_ids** | unix.gid_id.list |
| **gid,gids** | unix.gid.list |
| **lname,lnames** | unix.lname.list |

# 4 SYSTOOLS

The third DCD product, *systools* is a series of system administration utilities. To use the utilities both the *funkern* and *futil* products must be installed. If you are going to use the `addproduct,` `delproduct,` and `modproduct` utilities from this package, you must create an account with a login name of kits, with a UID of 1628, and a group of upd, with a GID of 3531. You must create this account prior to installing *systools.*

## 4.1 File Configuration

There are several configuration files that control the operation of the systools package. When you install `systools` default versions of these files are installed. Before using the utilities you must configure these files to reflect the specifics of your local system. In addition, you may need to change these files from time-to-time as part of the ongoing system management.

The following sections describe the format and purpose of each configuration file.

### 4.1.1 sys.env

The file `/usr/local/systools/library/sys.env` contains variable definitions that are used by each of the system administration utilities. The variable definitions are broken down into machine specific and general configuration areas.

The machine specific definitions primarily are variables that indicate the location of specific system programs. For example, the location of the `wc` program is different on the various UNIX platforms and so the `WC` variable is used to indicate the location of that program. The definitions of these programs should be correct upon installation.

The `YPDIR` variable is contained at the beginning of the machine specific definitions. This variable is used for two purposes. The first purpose is to indicate if the system is running NIS (Network Information Service, the new name for yellow pages). Uncommenting this variable indicates that NIS is being used on the system. The second purpose of the variable is to indicate the base directory for NIS since the location is machine dependent. It is not necessary to specify the directory location if NIS is not being used on the system.

The `PANDORA` variable is used to indicate that utilities that install or modify passwords should not age the password. The variable should be defined if for some reason your system can't handle password aging. One instance of a system that can't handle password aging is an SGI system running pandora (hence the name of the variable). Pandora is the program on the SGI console that displays login icons instead of the normal `login:` prompt.

In general, this file will be edited once upon initial configuration of the system administration utilities and then left unchanged. However, if you install NIS on your systems, you would need to edit this file and re-define a few variables. Once again, the comments in the file should indicate the correct value of each variable.

Two important variables are `PASSWD` and `GROUP`. These variables are used to indicate the names of the file that should be edited when accounts are added or modified. Normally, `PASSWD` is set to `/etc/passwd` and `GROUP` is set to `/etc/group`. However, if for some reason a different file is used for the password and group files then you should change these variables. One of the more common reason for a change occurs when running NIS. It is very common for the NIS master server to use a different file (i.e. /var/etc/passwd or /etc/passwd.yp) as the password file. In addition, if either of these variables is set to the value of `NULL` then the systools utilities will not allow the system to modify the password or group files. This configuration would be desirable if your system is running NIS, but is not the master server.

The end of the `sys.env` file contains definitions used by the `addproduct, delproduct,` and `modproduct` utilities. Any change of the variables must be coordinated with all machines using these utilities to provide products to the central UPD distribution machine.

### 4.1.2 Files in `/usr/local/systools/samples`

The directory `/usr/local/systools/samples` contains templates for system wide defaults for the following user files.

1. .login (named stdlogin)
2. .cshrc (named stdcshrc)
3. .mailrc (named stdmailrc)
4. .profile (named stdprofile)
5. .shrc (name stdshrc)
6. .Xdefaults (named stdXefaults.AIX and stdXdefaults.IRIX for the AIX and IRIX architectures respectively)

In addition, the directory contains files that are *sourced* from the std* files:

1. fermi.login
2. fermi.cshrc
3. setups.csh

4. fermi.profile
5. fermi.shrc
6. setups.sh

These files are a critical part of the Fermi UNIX environment. They provide a consistent login environment across the various UNIX platforms and it is hoped that the local system manager has no need to change these files. The first three files are for csh class shells (csh, tcsh) and the last three files are for sh class shells (sh, ksh, bash, zsh). The setups.* files are used to setup a ups environment. These files are *sourced* from the `fermi.cshrc/fermi.profile` to setup a fully functional ups environment. They can be *sourced* directly by scripts or the root account to obtain a ups environment without executing other aspects of the fermi files.

The files placed in the `samples` directory are placed there so the local system manager can inspect their contents before installing them to their *live* location. **After the system manager has made any local changes, the files must be copied to the /usr/local/etc directory. Utilities such as `adduser` and `clearaccounts` copy files from the /usr/local/etc directory in setting up the account, not the samples directory. In addition, the std* files *source* their files from /usr/local/etc not /usr/local/systools/samples.**

#### 4.1.3 greeting

The file `/usr/local/systools/library/greeting` is a greeting message mailed to users by the `adduser` program when a new account is added. Change this file as appropriate.

#### 4.1.4 functions

The file `/usr/local/systools/library/functions` determines which accounts will be allowed to run the various commands in `/usr/local/systools/library`. Furthermore, it allows you to indicate which UID and GID will execute the command. Thus, you can allow people other than the super-user (root) to run certain commands as root (or some other UID). The format of the lines are:

```
commandname:ruid,euid,rgid,egid,loginlist
```

The first field (commandname) is the name of the program in `/usr/local/systools/library`. The second field is four (4) comma separated numbers of the real and effective user id (UID) and group id (GID) that will be used to run the program. In almost all cases, these numbers will be zero (0). The third field is a command separated list of accounts that can execute the given command. The following is an example functions file along with an explanation.

```
addmember:0,0,0,0:root,*gm
adduser:0,0,0,0:root,xxx
chguser:0,0,0,0:root,xxx
deldata:0,0,0,0:root,xxx
delmember:0,0,0,0:root,*gm
disuser:0,0,0,0:root,xxx
moduser:0,0,0,0:root,xxx
resetuser:0,0,0,0:root,xxx
vigr:0,0,0,0:root
vipw:0,0,0,0:root
finduser:0,0,0,0:+*
clearaccounts:0,0,0,0:root,xxx,yyy
addlocal:0,0,0,0:-local
dellocal:0,0,0,0:-local
addproduct:1628,1628,3531,3531:-root,xxx
delproduct:1628,1628,3531,3531:-root,xxx
modproduct:1628,1628,3531,3531:-root,xxx
shell:0,0,0,0:+*
rm:0,0,0,0:root
chmod:0,0,0,0:root
kill:0,0,0,0:root
renice:0,0,0,0:root
```

In all cases except `addproduct`, `delproduct` and `modproduct` the commands will be executed with real and effective UID and GID of 0 (i.e. ruid,euid,rgid,egid is equal to zero). The `addproduct`, `delproduct`, and `modproduct` utilities will be executed with a UID of 1628 (the kits account) and a GID of 3531 (the upd group). In the case of `addmember`, both root and group managers can execute this command (denoted by *gm). A group manager account is defined as an account whose login name is the same as the group name. In the case of adduser, both root and the xxx account can execute this command. The `clearaccounts` command can be executed by root, xxx, and yyy and the finduser command can be executed by anyone (denoted by *).

Any item in the loginlist (including * and *gm) can be preceded with a + or - sign. The following describes the semantics:

1. No plus (+) or minus(-) sign indicates that the user must be logged in directly (i.e. he/she can not be su'ed to the account).
2. A plus (+) signs allows the user to be su'ed to the account but does not require it. In other words, he/she can login directly to the account **OR** be su'ed to it.
3. A minus (-) sign requires the user to be su'ed to the account.

Using the above rules, this indicates that `finduser` can be executed by anyone both when they are logged in directly as an account and when they are su'ed to another account. The `addlocal` and `dellocal` commands can only be executed when someone is su'ed to the `local` account. The `addproduct` and `delproduct` accounts can be executed by the `xxx` account (when directly logged in) and by someone su'ed to the `root` account.

The functions file should be mode 640 (read and write permission to owner, and read permission to group) and should have the same owner and group as `/usr/local/bin/cmd`.

In addition, you may create a file called *functions.hostname* (i.e. functions.fnsg01) in the `/usr/local/systools/library` directory. This file is identical to format, but takes precedence over the `functions` file. If no entry for a function is found in the `functions.hostname` file or it doesn't exist, the `functions` file is then searched.

#### 4.1.5 disks

The file `/usr/local/systools/library/adduser.work/disks` is used by the `adduser` program to determine where accounts should be placed. (Note: You may want to copy the version in `/systools/library/adduser.work` from a previous version of *systools*.) Each line in this file contains a directory name followed by a list of GIDS and UIDS. The following is a simple example of a disks file:

```
/usr/people,g=1234,u=456,
/usr1,g=1256,
/usr2,misc,g=1489,
```

This file indicates that if the GID is 1234 or the UID is 456 the account should be placed in /usr/people. If the GID is 1256 the account should be placed in /usr1. If the GID is 1489 the account is placed in /usr2. If no match is found the account is placed in /usr2 as indicated by the *misc* item. Note that this is a comma separated list that **must** end in a comma.

The first example shows how different disk partitions can be allocated to different experiments. A simpler example of the disks file would be to place **all** accounts in the same area. The following line would handle that case:

/usr/people,misc,

### 4.1.6 loginlist files

The `clearaccounts` program uses a data file to determine which logins may be cleared. These files are of the form `/usr/local/systools/library/loginlist.logname`. Thus if the `xxx` account had permission to use `clearaccounts`, a file should be created in `/usr/local/systools/library` with the name `loginlist.xxx` which contained a list of logins that `xxx` had permission to clear. This list should contain one login per line. These files should be mode 640 with group and ownership the same as `/usr/local/bin/cmd`.

### 4.1.7 groups files

The `addmember` and `delmember` programs look for a data file to determine which groups a given user may modify. These files are not required as documented in the section for `addmember` and `delmember`.

These files are similar in format to the loginlist files except they contain a list of group names instead of a list of logins.

### 4.1.8 shells

The `shells` file is a data file for the `cmd shell` program. It is a list of valid login shells.

### 4.1.9 user.* and group.* files

These files are data files for the `cmd rm, chmod, kill,` and `renice` programs. For `cmd rm` and `chmod` these files are located in `/usr/local/systools/library/rm.work` and for `cmd kill` and `renice` they are located in `/usr/local/systools/library/kill.work`.

These files contain as their extension an account name and use the `cwi` syntax to specify lists of users/groups. Thus the file `rm.work/user.wicks` would contain a list of users the `wicks` account would have the ability to delete (rm) or change permissions (chmod) their files. The files in `kill.work` are similar except they specify permissions that allow the designated user to kill processes (kill) or change the process priority (renice). The `group.*` files specify lists of groups instead of users.

#### 4.1.10 library.dirs

The `library.dirs` file contains a list of legal library directories other than `/usr/local/systools/library`. This file must be owned and writable only by root and is required to maintain security, but allow flexibility in providing alternate library directory locations.

An alternate library location is used by the `cmd` program when the environment variable $SYSTOOLS_LIB is defined to a value contained in the `library.dirs` file. It should be noted that when an alternate library directory is being used, all of the configuration files described in the above sections are now relative to the new library directory location.

### 4.2 Operation of Utilities

#### 4.2.1 The `cmd` program

None of the system administration utilities in `/usr/local/systools/library` are run directly. Instead, the `cmd` program is used. The primary purpose of `cmd` is to grant root permissions for specific commands. (Actually, it grants the permissions corresponding to the real and effective UID and GID listed in the `functions` file. In practice, this will almost always be root.) Since root privileges could allow the user to modify any file on the system, the `cmd` program conducts several checks before granting privileged access:

1. The `function` file has the same owner and group as the `cmd` program and has no world read or write permissions.
2. The command in `/usr/local/systools/library` that you are going to execute has the same owner and group as the UID and GID specified in the `functions` file. It must have no global read, write or execute permissions.
3. The directory containing the programs (i.e `/usr/local/systools/library`) has the same owner and group as the `cmd` program and has no world read or write permissions. If `cmd` is going to execute commands which will run as some UID/GID other than root, then the `/usr/local/systools/library` directory must include world execute permission.

Assuming all of the above checks have been met *and* the `functions` file authorizes you to execute the command, `cmd` executes the procedure with the appropriate privileges (normally root). Thus to execute the `adduser` program, which takes one argument (the UID), you would type:

```
cmd adduser wicks
```

Of course you would change `wicks` to the account name of the account you wish to add.

The `cmd` program may evaluate the above rules in a directory other than `/usr/local/systools/library` as specified by the $SYSTOOLS_LIB variable. However, the contents of this variable must be contained in the `/usr/local/systools/library/library.dirs` file.

When you setup systoools in the extended mode (setup -e systools) $SYSTOOLS_LIB is set to a directory within the current product structure. This is used mainly for testing purposes. It is not necessary to setup systools in normal operations.

### 4.2.2 UID and GID Databases

Many of the system administration utilities use the Global UID and GID databases which are maintained by the ACCESS Department. One of the programs in the *futil* product, `/usr/local/bin/obtain` is used to obtain these databases over the network. Both of these databases are stored in the `/usr/local/systools/library/adduser.work` directory. The UID database is in the `unix.uid.list` file while the GID database is in the `unix.gid_id.list` file. If a system administration utility needs access to one of the databases, it will first check to see if it already has been obtained and is stored locally, otherwise it will use the `obtain` program to get the database. In addition, if the current version of the database doesn't contain the data requested by the utility, `obtain` will be used to retrieve the latest version of the files, if the current version is more than a day old. If the current version is less than a day old, you will be asked if you want to retrieve a new copy.

### 4.2.3 deldata

Usage: `cmd deldata`

The `deldata` program is used to delete the UID and GID databases from the `/usr/local/systools/library/adduser.work` directory. You should use this command when you want one of the other system administration utilities which uses these databases to obtain a fresh copy from the network. This is needed from time to time as ACCESS updates the database.

### 4.2.4 File Locking

All of the system administration utilities that modify the passwd or group file first obtain a *moduser* lock by using the program `/usr/local/bin/getlock`. The lock is simply a file called `moduser` which is stored in the directory `/usr/local/systools/locks` and contains the PID of the process. If this lock cannot be obtained, the program exits. This simple locking mechanism, prohibits two system administration utilities from having write access to these files at the same time. **It doesn't prevent some other program from modifying these files.**

#### 4.2.5 finduser

Usage: cmd search_criteria

The finduser program searchs the UID and GID databases for the string specified as search criteria. A common use of this program is to search for someone's name prior to adding an account in order to have the correct UID. You may specify multiple strings as the search_criteria.

#### 4.2.6 adduser

Usage: cmd adduser uid|loginname

The adduser program adds an account for the specified UID or loginname. It consults the UID and GID databases to determine the correct loginname and GID. It consults the disks files to determine the correct location for the account. The program is interactive and will ask you to confirm these choices. It also installs a .profile, .login, .cshrc, and .mailrc file for the user. On AIX and IRIX systems, a .Xdefaults file is also installed. These files are copied from stdprofile, stdlogin, stdcshrc, stdmailrc, and either stdXdefaults.AIX or stdXdefaults.IRIX from the /usr/local/etc directory. It also creates a bin and rje directory in the users $HOME directory.

If the $HOME directory already exists, adduser will ask if the account should still be created. Adduser also prompts you for the login shell for the account. The default value is specified in the sys.env file.

If the GID doesn't exist in the group file, an entry is added. In addition, an entry is added in the passwd file. The location of the group and passwd files are set by the GROUP and PASSWD variables that are set in /usr/local/systools/library/sys.env. If either one of these variables has been set to the word NULL, then adduser will exit immediately, not allowing changes to these files. This configuration would be desirable if the machine is a NIS client. If the YPDIR directory variable is defined in sys.env file, then it is assumed that the system is the NIS master, and a yp make will be executed.

A password is automatically generated and displayed on the screen. This password is mailed to the user along with a greeting message specified in /usr/local/systools/library/greeting. The adduser program asks for an e-mail address (in typical UNIX form: accountname@machine) in order to send this information. The default machine for the e-mail address is specified in the sys.env file. The program allows an option to not send mail by entering three dashes for the e-mail address. When this option is chosen, the system manager must find another way to provide the initial password to the user.

Additional user specified tasks can be accomplished by creating files called adduser.local (always executed) and `adduser.hostname` (only executed for the specific host) in the library directory. These file **must** be Bourne shell scripts and must have the same permissions as the `adduser` script. The following shell variables are defined at the time of execution of the scripts:

| | |
|---|---|
| **$login** | account name |
| **$uid** | numeric uid |
| **$gid** | numeric gid |
| **$name** | full user name (value in comment field of passwd file) |
| **$home** | home directory of user |

#### 4.2.7 chguser/moduser

Usage: `cmd chguser uid|loginname`
Usage: `cmd moduser uid|loginname`

The `chguser` and `moduser` programs are identical and are used to change the UID, GID, or $HOME directory of the user specified by the uid or loginname. It still uses the information in the UID and GID databases, and `disks` files so the use of this program implies that one of these files has been changed.

The `chguser` and `moduser` programs use the same rules regarding NIS and the location of the group and passwd files as the `adduser` program.

#### 4.2.8 disuser

Usage: `cmd disuser loginname`

The `disuser` program disables the users account by inserting a dash (-) in the front of the encrypted password field of the password file. Re-enabling of the account with the same password can then be accomplished by removing the dash (by using the `vipw` program).

The `disuser` program use the same rules regarding NIS and the location of the passwd file as the `adduser` program.

#### 4.2.9 resetuser

Usage: `cmd resetuser loginname [password]`

The `resetuser` program assigns new automatically generated password to the specified account and mails this information in the same manner as the `adduser` program. If an optional second argument is given, the password is set to that argument.

The `resetuser` program use the same rules regarding NIS and the location of the passwd file as the `adduser` program.

### 4.2.10 addmember/delmember

Usage: `cmd addmember loginname`
Usage: `cmd delmember loginname`

The `addmember` and `delmember` programs allow the addition and deletion of login names from a specific group in the group file. If a `groups.loginname` file exists in the library directory for the user and the user is listed in the `functions` file, the user is prompted for the group to be modified, which must be listed in the `groups.loginname` file.

If there is no `groups.loginname` file for the user, are not a group manager account, and are listed in the functions file, any group can be modified.

Group manager accounts (assuming the default functions file) may only modify the group manager group.

The `addmember` and `delmember` programs use the same rules regarding NIS and the location of the group and passwd files as the `adduser` program.

### 4.2.11 vigr/vipw

Usage: `cmd vigr`
Usage: `cmd vipw`

The `vigr` and `vipw` programs invoke the vi editor to allow modification of the group and passwd files. As with all other system administration utilities it creates a lock file this preventing access from any other system administration utility.

The `vigr` and `vipw` programs use the same rules regarding NIS and the location of the group and passwd files as the `adduser` program.

#### 4.2.12 clearaccounts

Usage: `cmd clearaccounts`

The `clearaccounts` program is used to *clear-out* an account. It removes all files in the user's $HOME directory as well as their mail file. It then re-installs the initial files in the same manner as the `adduser` program. Finally, it eithers disables the password, or allows you to choose a new password for the account.

The `clearaccounts` program uses file of the form `loginlist.`*loginname* in the directory `/usr/local/systools/library`. This file should contain a list of accounts that can be cleared by the login name given in the *loginname* extension to the file. Each line in the file should contain a single login name.

You have the option of clearing all accounts, typing in specific accounts, are having `clearaccounts` prompt you for each account listed in the loginlist file. After you have specified which accounts will be cleared, you are asked, if you want to disable the password, or set a new one. Because of the destructive nature of this procedure, you are asked to confirm your desire to continue with the procedure before any file deletions occur.

The `clearaccounts` program use the same rules regarding NIS and the location of the passwd file as the `adduser` program.

#### 4.2.13 addlocal

Usage: `cmd addlocal relative_path full_path`

The `addlocal` program is used to add files to or update files in the `/usr/local` area. The argument `relative_path` refers to directory in which the file should be installed. This argument is given *relative* to the `/usr/local` directory. The argument `full_path` is the full pathname of the file to be installed. For example to install the file `/usr/people/newutil` in the `/usr/local/bin` directory you would execute:

```
cmd addlocal bin /usr/people/newutil
```

If the file you are trying to add already exists, you can only replace it if your effective uid (before executing `cmd`) is the same as the owner of the currently installed file. If you are the owner `addlocal` will copy the current version to the same directory with a *.O* extension.

Actions taken by `addlocal` are logged in `/usr/local/systools/logfile`.

#### 4.2.14 dellocal

Usage: `cmd dellocal relative_path`

The `dellocal` program is used to delete files from the `/usr/local` area. The argument `relative_path` is the path name of the file to be deleted from the `/usr/local` directory. You are only allowed to delete the file if your effective uid (before executing `cmd`) is the same as the owner of the currently installed file.

Actions taken by `dellocal` are logged in `/usr/local/systools/logfile`.

#### 4.2.15 addproduct/delproduct/modproduct

Usage: `cmd addproduct [-t tarfile] [-o os] [-p product] [-v version] [-c custom_info] [-C chain_name] [-y]`
Usage: `cmd delproduct [-o os] [-p product] [-v version] [-c custom_info] [-y]`
Usage: `cmd modproduct [-o os] [-p product] [-v version] [-c current_custom_info] [-C chain_name] [-m modified_custom_info] [-y]`

There are several requirements that must be met before a system can use the `addproduct`, `delproduct`, and `modproduct` utilities:

1. Have ups v1_8 or later installed
2. Have futil v1_2 or later installed
3. Have systools v1_3 or later installed
4. Install a kits account (UID=1628,GID=3531,group name=upd)
5. Configure the `addproduct/delproduct/modproduct` utilities
6. NFS mount the kits area from FNAL

Configuring the `addproduct/delproduct/modproduct` utilities involves modifying the `functions` file in the `/usr/local/systools/library` directory to indicate which accounts should be allowed to add and delete products. Refer to Section 4.1.4 for more details.

In order to mount the kits area from FNAL you must be authorized from the FNAL side. In order to have your system authorized, contact a system manger of FNAL.

The remote directory to be mounted is `fnal:kits$root`. The local mount point on the UNIX system should be `/usr/kits`. On ULTRIX systems, the local mount point should be `/usr/fnkits` since `/usr/kits` is used by the system.

The addproduct program is used by product maintainers to add products to the central UNIX product distribution machine, while the delproduct program is used to delete products. The modproduct program is used to change the customization information and/or chain instance of an existing product. All of these programs are interactive in nature and ask the following questions:

**Name of tarfile** A full or relative path name may be specified. The csh tilde syntax is valid even if you aren't running csh. The tarfile must be publicly readable. (This question is only asked by addproduct.) A default value may be passed with the -t option.
Example: `cmd addproduct -t /tmp/tarfile`

**Operating System** A numbered list is presented. You must choose one of the given choices. A default value may be passed with the -o option.
Example: `cmd addproduct -o IRIX`

**Product name** A numbered list of currently installed products for the operating system is presented. If you are adding a new product you may specify a product name not on the numbered list. A default value may be passed with the -p option.
Example: `cmd addproduct -p systools`

**Version number** Once again a numbered list is presented of currently installed version for that product. This question works the same as product name. A default value may be passed with the -v option.
Example: `cmd addproduct -v v1_3`

**Customization information** This allows you to specify restrictions on the product (i.e. only for E665). A help message is available with further details. The information specified here is combined with the operating system choice to determine the flavor of the product. A default value may be passed with the -c option. For modproduct, this refers to the customization information of the existing product.
Example: `cmd addproduct -c none`

**Chain Instance** The addproduct and modproduct utilities asks if this product should be a chain instance. Legal chain instances are: current, development, test, new, and old. A default value may be passed with the -C option.
Example: `cmd addproduct -C current`

**Modified Customization Information** This is only used by the modproduct utility to specify the customization information for the modified product. It may be the same as the customization information of the existing product in cases when only the chain instance is being changed. A default value may be passed with the -m option.
Example: `cmd modproduct -m none`

After receiving answers for all of these questions, you are asked to confirmed the answers. If you do not indicate all information is correct, all questions will be re-asked using the current information as the default answers. Once the answers have been confirm, the tarfile is copied or deleted as appropriate, the distribution database is updated, and the action is logged.

When the `-y` option is used, `addproduct`, `delproduct`, and `modproduct` operate in a batch mode. It assumes that values for all necessary information has been passed in via the command line. If necessary information is not provided or is invalid, an error message is printed out and the program exits.

### 4.2.16 shell

Usage: `cmd shell [new_login_shell]`

The `shell` program allows a user to change his/her login shell to a value specified in the `/usr/local/systools/library/shells` file. If the optional second argument is specified, the process is performed non-interactively. Otherwise, the user is prompted for the new login shell.

### 4.2.17 rm/chmod

Usage: `cmd rm files...`
Usage: `cmd chmod files...`

The `rm` and `chmod` commands allow designated users to remove and change permissions on the files specified on the command line. A designated user must have either a `user.account_name` or a `group.account_name` file in the `/usr/local/systools/library/rm.work` directory. These files contain list of users/groups using the syntax of `cwi`. In order to perform the action, the files must either be owned by an account specified in the user list or grouped to an group in the group list.

### 4.2.18 kill/renice

Usage: `cmd kill files...`
Usage: `cmd renice files...`

The `kill` and `renice` commands allow designated users to kill and change the priority of the process ids specified on the command line. A designated user must have either a `user.account_name` or a `group.account_name` file in the `/usr/local/systools/library/kill.work` directory. These files contain list of users/groups using the syntax of `cwi`. In order to perform the action, the process must either belong to an account specified in the user list or belong to an account with a primary group of a group specified in the group list.

# 5 FULIB

The *fulib* product provides library of C-callable functions that are meant to be used in other products. Several of the routines provide function versions of the utilities in the *funkern* product, but it is not a requirement of a function in *fulib* to have a utility counterpart in the *funkern* product.

To link with this library you must `setup fulib` and include the module $FULIB_DIR/lib/libfulib.a on the `ld` or `cc` line.

## 5.1 funame

The complete usage of the `funame` function is documented in the `funame.3` manual page. The following is taken from the SYNOPSIS section.

```
#include < sys/utsname.h>

int funame(struct utsname *baseuname, char *hostname, char *ipaddress)
```

The first argument is the structure used for the `uname(2)` system call, and the remaining arguments are character strings for additional information provided by `funame()` that isn't provided by `uname()`.

The return value is -1 for failure and non-negative for success.

The function version is meant to be used from with-in a C program, as it is much more efficient than forking a shell to call the shell utility version of `funame`.

To link with this library you must `setup fulib` and include the module $FULIB_DIR/lib/libfulib.a on the `ld` or `cc` line.

## 5.2 dropit

The complete usage of the `dropit` function is documented in the `dropit.3` manual page. The following from the SYNOPSIS section.

```
char *dropit(char *path, char idel, char *odel, char **substring)
```

The first argument is the input string to be manipulated, which is assumed to contain fields separated by the character specified in the second argument. The third argument is the field separator of the string returned by the function. The fourth argument is an array of strings. Any field within the first string that contains a substring of any of the element of this array is removed and returned as the output of the function.

The function version is meant to be used from with-in a C program, as it is much more efficient than forking a shell to call the shell utility version of `dropit`.

## 5.3 tilde_expand

The complete usage of the `tilde_expand` function is documented in the `tilde_expand.3` manual page. The following provides a brief synopsis.

```
int tile_expand(char *input, char *output)
```

The `tilde_expand` routine performs $HOME directory expansion as in csh and ksh. If the *input* character string begins with a tilde character and the remainder of the string refers to a valid account, then the *output* character string will contain the $HOME directory for that account. Otherwise, the *output* character string will be identical to the *input* character string. The routine returns 0 for success and -1 for failures with `errno` set appropriately. When a tilde expansion can't be performed errno is set to `EINVAL`.

## 5.4 Shell Wildcards

There are a series of routines that functions similar to shell wildcarding. These routines are used to allow wildcarding with ups list and may be useful for other products. The routines are known as gobi which is a take-off on the term globbing which is often used to describe shell wildcarding. The routines are fully documented in DCD Release Note 17.0.