Acknowledgments

This document represents a joint effort on the part of several members of the Fermilab Computing Division.

Contributors and reviewers include: Eileen Berman, Etta Burns, Lauri Loebel Carpenter, Chuck DeBaun, Mike Diesburg, Lynn Garren, Lisa Giacchetti, Anne Heavey, Randy Herber, Randy Hilgers, Marc Mengel, Frank Nagy, Judith Nicholls, Ruth Pordes, Dane Skow and Margaret Votava.

We'd like to thank the readers who have taken the time to send us suggestions about the manual.

Table of Contents

Chapter 1: Introduction

This chapter provides an introduction to the Fermi UNIX Environment, known affectionately as *FUE* (pronounced *foo-ey*), as well as an introduction to the document itself. In particular you will find:

- an introduction to the Fermi UNIX Environment
- brief introductions to **UPS** and **UPD** which provide the methodology, infrastructure and user interfaces for product support and distribution under the Fermi UNIX Environment
- the intended audience and purpose of the *Guide to the Fermi UNIX Environment*
- the typeface conventions and symbols used throughout the document
- a summary of the contents of the chapters and appendices

> We plan a second release of this document by July 1997 which will include the much-needed documentation on **Product Development under FUE**. Check the WWW page listed in section 1.2.3 for updates.

1.1 What is the Fermi UNIX Environment (FUE)?

The on-site UNIX[1] computer systems at Fermilab are used for a wide range of tasks, including among other things general purpose interactive use, batch jobs, farms[2], and data acquisition systems. In order to most efficiently accommodate the needs of the Fermilab user community, the Computing Division provides a single, operating system-independent UNIX environment known as the *Fermi UNIX Environment*, or *FUE*, which we document in this guide.

FUE started as an initiative within the Fermilab Computing Division. The FUE project team is composed of people from several different departments within the Computing Division, and some researchers from other divisions. The team's activities are described in Appendix A.

The mission of the FUE project team, in brief, is to provide a cross-department and cross-division structure for the proposal, discussion, design and implementation of all things that affect the user when operating in a UNIX environment at Fermilab. An auxiliary goal is to provide a standard product support methodology. Although many activities in this area are the mandate of a single group within the Computing Division, the FUE project team provides a forum in which to coordinate and organize this work.

---

1. UNIX has been customized to several different hardware platforms, and each of these adaptations (SunOS, IRIX, etc.) is considered a separate operating system (also called OS type in this document).
2. Farms are clusters of RISC-based workstations for which parallel processing software has been developed to support activities such as event reconstruction and Monte Carlo.

1.1.1 Elements of FUE

FUE can be divided into three main areas:

- a set of login scripts, called the *Fermi files*, that provide a common environment; these are described in *UNIX at Fermilab*, document number GU0001
- an infrastructure to allow uniform access to software products: **UPS** (UNIX Product Support) and **UPD** (UNIX Product Distribution).
- general utilities, system administration tools, and software products, all in **UPS** format

1.1.2 Product Support and Distribution under FUE (UPS and UPD)

The methodology and infrastructure for product support and distribution under FUE is provided via a software support toolkit called UNIX Product Support (**UPS**). The **UPS** system was developed with the goal of providing a uniform and consistent interface for the management, distribution, installation and use of all the UNIX software it makes available.

UPS consists of two parts:

- one or more databases
- a set of procedures/programs to manipulate database

A **UPS** database is a directory containing an ASCII file for each individual product. This file contains information about and pointers to all of the installed copies of the product. Product maintainers use **UPS** commands to add, delete and modify product information in these files. Several environment variables are defined within the context of **UPS**. In particular, the variable *PRODUCTS*, set to the directory specification of the **UPS** database, is used internally by **UPS** commands to find the database. *PRODUCTS* can in fact be set to point to multiple directories, thus giving support for multiple databases.

UNIX Product Distribution (**UPD**) is a companion product to **UPS**, and provides the functionality for distributing products, generally stored as tar files, from a remote host to a local system. A convenient menu interface is provided that includes the **UPD** and **UPS** functionality necessary to complete an entire process of listing available product instances on the distribution node, copying a product instance from the distribution node, installing it on the local node, and declaring it to the local **UPS** database. The interface also can be used to list the files in or extract a file from a remote tar file.

1.1.3 FUE Compliance

Recommendation

The FUE conformance levels are described in Appendix A. They are included in this document because they are the currently defined and accepted standards which determine (1) the levels of support that the Computing Division provides, and to some extent (2) what software products you can have installed. However, throughout this document the term “FUE compliant” refers to a conformance level of Full FUE or FUE Plus. The software at these conformance levels works properly and predictably. We recommend against choosing a level of conformance lower than Full FUE because you may experience problems with a small but not easily identifiable subset of software products.

Advantages

Together FUE and **UPS** provide a couple of very important advantages for real-time data acquisition and other production systems at Fermilab:

- Multiple versions of a software product can be made available concurrently.

  Each release of a product is self-contained and can be installed and accessed independently of other releases. This feature makes possible the next advantage:
- You can back out of a new software release **completely** and **assuredly** if something goes wrong, and immediately start up a previous tried-and-true release.

All users of UNIX utilities and software products on a FUE-compliant system will appreciate these additional features:

- high degree of uniformity in appearance and functionality of different UNIX platforms
- consistent and reliable support from Computing Division
- capability for supporting a cluster of UNIX systems from different vendors fitted with a common environment and common software products (and optionally a common products disk)
- an easy and consistent method of accessing different software products
- wider availability of supported software
- linking of dependent products in such a way that when a product is made accessible for use, any products that it relies upon for proper functioning are also automatically made accessible

Further advantages that product installers, system administrators and product providers will encounter include:

- a rich set of commands for installing and maintaining products as well as for connecting between systems
- a single product development/distribution methodology and set of standards
- a wider audience available to test new products

1.2 The *Guide to the Fermi UNIX Environment*

1.2.1 Intended Audience

Now that you understand the general purpose of FUE and its related tools, we can identify the intended reader groups very clearly. This guide is intended mainly for the following types of users:

- maintainers of software products on local FUE-compliant systems
- installers of software products on local FUE-compliant systems
- administrators of systems on which software products are provided in accordance with FUE standards
- administrators of systems which need to be made FUE-compliant
- developers of software products that are intended to be distributed in accordance with FUE standards
- providers of pre-existing software that must be made to conform to FUE standards

End users of software products should find most if not all the information they need in *UNIX at Fermilab*, document GU0001. However, this information is also contained here, therefore for completeness we include:

- end users of FermiTools[1] software products (on systems that are not necessarily FUE-compliant)
- end users of **UPS** products on FUE-compliant systems

1.2.2 Purpose

The different chapters and appendices in this guide cover a variety of material as suggested by the intended audience identified in the previous section. In brief, we attempt to explain all the concepts related to product support and distribution here at Fermilab, and to illustrate the associated functions. The material is organized with the goal that the different user groups find all the information they need, *but only as much as they need.* The range of system activities addressed includes:

Add phrase for product development/provision, above in rev 2.

- obtaining information about **UPS** products from a product distribution database
- accessing **UPS** products that have been installed on a FUE-compliant node or cluster
- installing **UPS** products onto a FUE-compliant node or cluster
- maintaining a **UPS** environment on a node or cluster
- configuring a system to be FUE-compliant
- making a pre-existing product **UPS**-compliant
- developing a product in accordance with **UPS** standards

☞ It is *not* the purpose of this document to provide information on:

- general UNIX system administration
- general UNIX or Fermilab information (see *UNIX at Fermilab*, document number GU0001)
- the use of any specific **UPS** product

1.2.3 Availability of this Guide

Copies of the *Guide to the Fermi UNIX Environment* can be obtained from the following sources:

WWW	Don't know yet -- probably not right off UNIX Resources this time...
Paper Copies	Wilson Hall, 8th floor, NE; document number ?? (just across from what used to be the Computing Division library)
AFS	PostScript files of the individual chapters and appendices are available at `/afs/fnal/files/docs/UNIX/fue_guide/ps`

1.2.4 Conventions Used in this Guide

The following notational conventions are used in this document:

bold	Used for product names (e.g., **UPS**) and shells (e.g., **csh**).

1. Fermilab-written software products that are made publicly available.

italic
: Used to emphasize a word or concept in the text; many terms found in the index are italicized in the text. Also used to indicate variables and reference book titles.

`typewriter`
: Used for filenames, path names, contents of files, output of commands.

`typewriter-italic`
: In filenames, pathnames and environment variables, indicates variables for which the user must make context-specific substitutions (e.g., `$`*`PRODUCT`*`_DIR/ups` is `$WWW_DIR/ups` for the product **www**)

`typewriter-bold`
: In text, used to indicate commands and prompts. In command formats, indicates what the user types “as is”.

bold-italic
: In command formats, indicates variables for which the user must make context-specific substitutions.

<CTRL-*char*>
: Indicates a control character. To enter a control character, hold down the control key (labeled Ctrl, probably) while pressing the key specified by ***char***.

[ ]
: In command formats, indicates **optional** command arguments and options.

|
: When shown in in a command line (**x|y|z**), separates a series of options from which one may or must be chosen (depends if enclosed in square brackets). In UNIX command formats, used to pipe output of preceding command to the following one.

' ... '
: Single vertical quotes indicate apostrophes in commands.

` ... `
: Single backquotes in UNIX commands have a special meaning (use output of string in place of string itself). Don’t confuse them with apostrophes.

...
: Means that a repetition of the preceding parameter or argument is allowed.

All command examples are followed by an implicit carriage return key.

Some of the files discussed in this document are shell family-specific, and thus come in pairs. Their filenames carry the extensions `.sh` and `.csh`. We often refer to a pair of files as `filename.`*`$UPS_SHELL`*, where *`$UPS_SHELL`* is either `sh` or `csh` (for example, see section 2.1.1).

The following symbols are used throughout this document to draw your attention to specific items in the text:

A “bomb”; this refers to something important you need to know in order to avoid a pitfall.

This symbol is intended to draw your attention to a useful hint.

This is a reminder.

1.2.5 Summary of Chapters and Appendices

<u>include once they’re nailed down</u>

1.2.6 Reader Comments

The *Guide to the Fermi UNIX Environment* is bound to contain some errors, however we endeavor to minimize the error count! We encourage all the readers of this guide to report back to us:

- errors or inconsistencies that we have overlooked
- any parts of the manual that are confusing or unhelpful -- please offer *constructive* suggestions!
- other topics to include (keeping in mind the purpose of the manual)
- tricks that other users might find helpful

Send your comments via email to *cdlibrary@fnal.gov.*

Chapter 2: Understanding UPS and Using it to Manipulate Products on your System

In this chapter we explain in detail the concepts used in **UPS**, and describe how to perform common operations using **UPS** commands. A short section on troubleshooting is also provided.

The concepts covered in section 2.1 are also fundamental to **UPD**, so we recommend that you read this section prior to launching into Chapter 3 which covers product installation using **UPD**.

2.1 UNIX Product Support (UPS)

UPS (UNIX Product Support) is a software support toolkit developed here at Fermilab for the management and access of software products on local systems by the system administrators and users. It was also designed to facilitate the product distribution and configuration management tasks of the product providers. The two principal user benefits are:

- a uniform interface for accessing all products on a UNIX system via the **`setup`** command
- unified and coordinated support of in-house and vendor-supplied software across all the supported UNIX operating systems

A **UPS** installation has two parts:

- one or more databases; a **UPS** database is a directory which functions as a central repository of information about products and contains pointers to products
- a set of procedures/programs to manipulate and view the database(s)

2.1.1 The UPS Environment

On a FUE-compliant system the standard default login files (also known as *Fermi files*[1]) set your environment to facilitate **UPS** operations, among other things. Regarding shells, **UPS** supports both UNIX shell families, C (Berkeley) and Bourne[2]. At login time, the environment variable *UPS_SHELL* gets set to indicate the family of the shell in use. Any shell within that family can then be used for **UPS** operations.

The Fermi files are designed for interactive use only and will break if they are called from a non-interactive script. You need to explicitly include the following actions in the order shown in any script that you will use to setup products (i.e. in which you use the **`setup`** command to access one or more products; **`setup`** is described in section 2.2.1):

For the C shell family:

```
source /usr/local/etc/setpath.csh
```

1. See *UNIX at Fermilab* (GU0001) for descriptions and listings of the Fermi files.
2. The C shell family includes **csh** and **tcsh**; the Bourne shell family includes **sh**, **ksh**, **bash** and **zsh**. **zsh** is not supported at Fermilab.

```
source /usr/local/etc/setups.csh
```

For the Bourne shell family:

```
. /usr/local/etc/setpath.sh
. /usr/local/etc/setups.sh
```

The `setpath.$UPS_SHELL` script sets a reasonable default starting path. The `setups.$UPS_SHELL` script (read "set **UPS**") sets the **UPS** environment variables described above[1].

The login files also set the alias for **`setup`**. **Note that if you change shells or processes, this alias is no longer available to you.** In this case you need to manually re-source the script `setups.$UPS_SHELL` in your new shell.

If you have made any changes to your login files, make sure that the following lines are still included before using **UPS**:

For the C shell family:

`.cshrc` includes	`source /usr/local/etc/fermi.cshrc`
`.login` includes	`source /usr/local/etc/fermi.login`

For the Bourne shell family:

`.profile` includes	`. /usr/local/etc/fermi.profile`
`.shrc` includes	`. /usr/local/etc/fermi.shrc`

Three environment variables get set by `setups.$UPS_SHELL` at login:

PRODUCTS	If only one **UPS** database is defined, this points to it; if two or more are defined, this variable gets set as the space-separated list of **UPS** databases. The order of the databases in this list reflects the order of precedence for accessing products.
UPS_DIR	This points to the top level directory (called the product root directory) of the current instance of **UPS** (see section 2.1.9 for an explanation of "current instance").
UPS_SHELL	As mentioned above, this indicates the shell family in use; it is set to either `sh` or `csh`.

You can use the environment variable *PRODUCTS* to list all the available products. For example, if it is set to the following list of **UPS** databases:

```
/home/dcdsv0_lv0/aheavey/ups_play/ups_database/declared
/usr/products/ups_database/declared
/afs/fnal/products/ups_database/declared
```

then the command:

`% ls $PRODUCTS`

1. These scripts are included in the **systools** package (see section 4.4).

will yield information similar to the following (edited for brevity):

```
/afs/fnal/products/ups_database/declared:
acrobat          frame           lund            tcl
admintools       fslib           mad             tclx
afsemu           ftcl            mailtools       tex
...text removed...
flint            ktevana         softwindows     xpdf
fmb              lapack          stdhep          zephyr
fmss             libgpp.obs      systools

/home/dcdsv0_lv0/aheavey/ups_play/ups_database/declared:
cedit     exmh      fermitpu  glimpse   juke      www

/usr/products/ups_database/declared:
altuascvs    fulib        lund         qq           tk           www
chekker      funkern      mbone        req          tpu          xemacs
conv2html    futil        mgui         sdsscvs      uas_build    ximagetools
...text removed...
flint        html         ntp          tcl          upp
fmb          ispell       ocs          tclx         ups
fonts        juke         olscvs       teleserver   webservers
```

Notice that the **ls** command does not necessarily list the databases in the order of precedence according to the *PRODUCTS* variable.

When you setup a product, the environment variable ***PRODUCT**_DIR* gets defined, where ***PRODUCT*** is the name of the product in upper case, for example *WWW_DIR*. This variable points to the root directory of the particular copy of the product that you setup; for example after running:

% setup www v2_7

the variable *WWW_DIR* is defined as something like `/`*`ups_dirs`*`/products/SunOS+5/www/v2_7`. You can then run a command like the following:

% ls -al $WWW_DIR

which returns a list of the files and subdirectories in the product root directory for **www** version v2_7.

```
total 40
drwxr-xr-x   7 products g150        2048 Jun 19 18:54 .
drwxrwxrwx   5 root     root        2048 Sep  9 23:16 ..
-rw-r--r--   1 products g150         165 Jun 18 11:31 BUILD_INFO
-rw-r--r--   1 products g150        4703 Oct 16  1995 Makefile
drwxr-xr-x   2 products g150        2048 Aug  9 18:19 bin
drwxr-xr-x   2 products g150        2048 May  5  1996 etc
drwxr-xr-x   2 products g150        2048 May  5  1996 include
drwxr-xr-x   4 products g150        2048 May  5  1996 lib
drwxr-xr-x   2 products g150        2048 May  5  1996 ups
```

2.1.2 UPS Products

Products distributed and managed by the **UPS** system on a distribution or user node are often called **UPS** products. **UPS** products can be maintained on different disks, and/or in different directories. To be included under the **UPS** umbrella on a node or cluster, products must be installed (preferably in an area specially designated for products, although **UPS** does not require this) and declared to a **UPS** database. Product declaration can be done in **UPD** (see section 3.3) or via the **UPS** command **`ups declare`** (see section 2.2.2).

On the distribution node, all the constituent programs and/or data files of a particular copy of a **UPS** product (called an *instance*, see section 2.1.7) are grouped into a directory tree which has been packed up into a single binary file for distribution, generally a tar file. During product installation on a user node, this file can be unwound using **tar** (optionally with **UPD** or **UPS** as an interface) and installed into its product root directory.

2.1.3 UPS Databases

A **UPS** database is, simply put, a directory. Within the directory, there is a *product file* for each individual product that is accessible via **UPS** on the system. A product file, described in section 2.1.4, contains identifying information for, and pointers to, each of the installed copies of a product. **UPS** commands refer to the database directory via the environment variable *PRODUCTS*, described in section 2.1.1.

The environment variable *PRODUCTS* can be set to point to one or to several directories, thus allowing support for multiple databases. This allows users to maintain one or more private databases in addition to or instead of the common one(s). Creating additional databases is discussed in Appendix B. In **UPS** operations, the databases are searched in the order they appear in *PRODUCTS*.

In a multiple database environment, commands that modify any element in a database must specify which database the element is in. A **`-z`** option (described in section 2.2) is provided for several of the **UPS** commands. It takes the database specification as its argument, as shown:

`%` ***ups_command*** **`-z /usr/mydatabase [`*other options*`]`** ***product***

The database specified in a command in this way does not have to be included in the *PRODUCTS* environment variable.

2.1.4 UPS Product Files

The information that **UPS** needs to identify, locate and retrieve a product and its requirements resides in an ASCII *product file* specific to that product in the **UPS** database. The name of each product file is simply the product's name (see the list of product files obtained by the command **`ls $PRODUCTS`** in section 2.1.1). The file's contents can be summarized as:

- a product identifier line showing the product name
- a line that says `DO NOT EDIT THIS FILE DIRECTLY.` Nothing prevents you from editing the file, but you run two risks if you do. First, if you enter something that **UPS** doesn't recognize you may corrupt the database so that commands cannot work on this file. Secondly many editors leave journal and/or backup files that you need to remember to delete. These extra files could cause unexpected results if you ever use wildcards in commands.
- a **UPS** database version identifier line
- zero or more chain clauses (described in section 2.1.9)

• one or more instance clauses (described in section 2.1.7)

In order to fully interpret the contents of this file, several concepts presented later in this chapter must be understood. The following is an example of a product file for **www**:

```
Product:        www

# This file is maintained by Unix Product Support.

# DO NOT EDIT THIS FILE DIRECTLY.



db_version:     "1.2"


Chain:
        current OSF1+V3 v2_7    9-26-96 "mengel"
end.

Instance:       "v2_7"  OSF1+V3 /afs/fnal.gov/products/OSF1/www/v2_7
                9-26-96         "mengel"


                Authorized_Nodes: [*]

                Requirements:    {(< ghostview,current,OSF1+V3),(< ximagetools,c
urrent,OSF1+V3),(< xpdf,current,OSF1+V3)}
                                 {}
                                 {}
                                 {}


end.
```

2.1.5 Versions

UPS supports multiple versions of software products. Maintaining multiple versions on a given system is often necessary and/or desirable for several reasons, for example the overlap of development and production use of a product or the sharing of workstation resources by many different users. Therefore, when a new version of a product becomes available, an existing directory tree cannot simply be replaced with another. Rather, a branch must be added for the new version.

2.1.6 Flavors

What is "Flavor"?

Many programs require separate compilation for the different UNIX operating systems. In **UPS**, we maintain different directory trees for the separate compilations (and related files) of the same product. We therefore need to distinguish between OS-dependent compilations. To indicate the OS dependency, we use the term *flavor*. This additional term allows us to maintain the same product name and version across the different operating systems and releases, which is desirable since the same program source files are used in the separate compilations.

On a given system, several different copies of a product may exist. When you use **UPS** commands to manipulate or use a product, the system needs to have enough information to select the appropriate one. Many **UPS** commands support a **-f** option allowing you to specify flavor-related information.

Simple Flavors

Simple flavors (as opposed to extended flavors) are the UNIX operating system names, as returned by the command **funame -s** (for example IRIX or SunOS). This is sufficient for products which can run on multiple releases of a single OS. For products which have no compiled programs, and are thus operating system-independent, a special flavor of NULL is used.

Extended Flavors

In addition to differentiation by OS, some products require separate compilations for different releases of the same operating system (e.g., IRIX+5 and IRIX+6). For this purpose, **UPS** supports *extended flavors*, which are designations that include both OS (simple flavor, e.g., IRIX) and OS release (e.g., +5). This allows a mixed cluster (e.g., IRIX+5 and IRIX+6) to share a single **UPS** database.

Extended flavors may further include information about options used by the product developer at compilation time (e.g., **+debug**). Keep in mind that there are many more extended flavors than simple flavors, and most products require differentiation by extended flavor.

Extended flavor is designed to accomplish two goals:

- identify the most appropriate compilation of a product for the given operating system and release
- allow the product developer to specify additional compilation options such as **debug**, **optimized**, etc.; the user can then select and run one of these compiled instances of a product for special purposes

2.1.7 Instances

Each installed, declared copy of a product in **UPS** is called an *instance* of the product. Each instance has a unique combination of product name, version and (extended) flavor. The concept of *chains*, discussed in section 2.1.9, allows users to easily access the appropriate instance, without needing to remember its version number and (extended) flavor.

In the product file one instance clause appears for each installed, declared instance of the product. This is the actual product declaration to **UPS**.

The instance clause starts with `Instance:` and ends with `end.` It includes:

- product version, in quotes
- flavor (simple or extended)
- product root directory
- date of installation
- installer's name in quotes
- authorized nodes (usually set to "*" indicating that all nodes known by the active instance of **UPS** are authorized)
- use and build requirements, if any (see section 2.1.10)[1]

Below is a sample instance clause showing a set of use requirements:

```
   Instance:       "v2_7"  OSF1+V3 /afs/fnal.gov/products/OSF1/www/v2_7
                   9-26-96         "mengel"


                   Authorized_Nodes: [*]

                   Requirements:    {(< ghostview,current,OSF1+V3),(<
ximagetools,current,OSF1+V3),(< xpdf,current,OSF1+V3)}
                                    {}
                                    {}
                                    {}


   end.
```

2.1.8 Flavor Specification and Instance Selection

When a **UPS** command is issued, the system must determine which product instance or instances to act upon. For some **UPS** commands multiple instances can be retrieved, as is often the case for **`ups list`**, for example. The product name and version are generally specified unambiguously, but the extended flavor can be specified such that it requires some interpretation by the system. This allows **UPS** to do some of the instance selection work for you. In fact for some operations you can choose not to specify flavor at all and let **UPS** determine it completely.

Flavor Specification Vehicles

1) **UPS** will determine the flavor based on the rules described under *Instance Selection Algorithm* below, if you do not specify one. In most cases this is sufficient.

2) Flavor can appear on a **UPS** command line as the argument to the **`-f`** option:

   `%` ***ups_command*** **`-f`** ***flavor*** **`[`*****other options*****`]`** ***product***

   When specified this way, the system will only retrieve an instance exactly matching ***flavor***.

3) **UPS** provides an environment variable, *UPS_EXTENDED_FLAVOR*, that you can use to specify compilation options. If you set this variable, **UPS** internally appends its value to the flavor specification (if any) used in subsequent **UPS** commands. In the absence of a command line flavor specification, **UPS** internally appends the value of this variable to the OS type of the machine on which the command is run.

4) In addition, for the **UPS** commands **`setup`** and **`ups tailor`**, the **`-x`** and **`-X`** options are provided. They take arguments in the same format as extended flavors and override the value of *UPS_EXTENDED_FLAVOR* in individual commands as follows:

   `-x` ***argument*** append ***argument*** to *$UPS_EXTENDED_FLAVOR*

   `-X` ***argument*** replace *$UPS_EXTENDED_FLAVOR* by ***argument***

---

1. The top set of curly brackets under `Requirements` is for use requirements, the middle two were supposed to be for start and stop requirements but were never implemented, and the last one is for build requirements.

Note that whereas a flavor specified on the command line as the argument to the **-f** option must be matched exactly, portions of the flavor specified by *UPS_EXTENDED_FLAVOR* or the **-x** or **-X** options do not need to be matched in order for an instance to be retrieved. If an exact match is found, it is retrieved; if not, the system uses the algorithm described below in *Instance Selection Algorithm* to retrieve the most suitable instance.

Flavor Specification Syntax

To specify a flavor in a **UPS** command, use the **-f** option in the following format (note that other command line elements are suppressed in this example):

`%` ***ups_command*** **-f** ***OS_type+OS_release+token_1+token_2 . . .***

where the elements of the flavor specification are:

OS_type The simple flavor; a string equal to **\`funame -s\`**. If the **-f** option is used, this argument is required and must appear as the first element of the argument.

OS_release The operating system release; this is not a required element of the argument, but if included, it must directly follow ***OS_type***. The syntax of ***OS_release*** is given in the box below.

token_n These fields are optional and allow the specification of particular compile/build-time options, such as **+debug** or **+nosrc** or **+debug_machine_code_only**. The available tokens on a product are set by the product maintainer and therefore vary from product to product; there are no standards for them.

The ***OS_release*** field can be specified in any of the following ways:

+OS_release	specifies exactly the release to match; only an instance with an exactly matching ***OS_release*** can be retrieved for use. ***OS_release*** is a string equal to **\`funame -r\`**, or to the left-most characters of **\`funame -r\`**. Usually it is entered up to the first period; e.g., V3.2 is entered as **V3** (on OSF1), or 5.5.1 is entered as **5** (SunOS). Sometimes the vendor makes so many OS changes in so-called *minor* releases that more information must be kept (e.g., **3.2.1** and **3.2.5** on some AIX products).
+ or **++**	tells the system to use the best match to the current operating system release. For example if you specify **-f IRIX+** on an IRIX v5_2 node, an instance with the value `IRIX+5.2` would be selected in preference to `IRIX+5` which would in turn be selected before `IRIX`. Use the single **+** if no optional token fields follow; use the double **++** if one or more does. For example, the flavor specifications **AIX+** and **AIX++debug** are acceptable, but **AIX++** and **AIX+debug** are not.
unspecified	only an instance with neither ***OS_release*** nor extra token specifications in its flavor can be retrieved.

The environment variable *UPS_EXTENDED_FLAVOR* takes a value of the format ***token_1+token_2 . . .***, where all elements are optional. Here are a couple of examples of possible values for *UPS_EXTENDED_FLAVOR:*

- `debug`

- `debug+nosrc`

The **`-x`** and **`-X`** command line options take arguments of the same format.

Instance Selection Algorithm

The instance selection is made based on matching the greatest number of elements of the extended flavor, provided all other criteria are met. A flavor specification on the command line takes precedence, the environment variable *UPS_EXTENDED_FLAVOR* is used as a secondary weighting.

UPS uses the following built-in rules to determine which instance to retrieve:

1) That instance with a flavor exactly matching the ***OS_type+OS_release*** (***+extra tokens***).

2) That instance with a flavor exactly matching the ***OS_type*** and the release specification which is the longest first substring of ***OS_release***.

3) That instance with a flavor exactly matching the ***OS_type*** and having no release specification or extra tokens.

4) That instance with a flavor of NULL.

5) The selection fails if none of the above can be met within the databases supplied.

☞ The system will never select an instance with ***token*** fields in its flavor designation beyond those specified on the command line or in *UPS_EXTENDED_FLAVOR* (and **`-X`** or **`-x`**).

Examples

The following table gives examples of instance selection:

Current Operating System	Available Instances	**`-f`** argument	*$UPS_ EXTENDED_ FLAVOR*	Selected Instance
IRIX 4	IRIX+5.3 IRIX NULL	none		IRIX
IRIX 4	IRIX+5 IRIX+4 IRIX NULL	none		IRIX+4
IRIX 4	IRIX+5 NULL	none		NULL (sometimes not compatible)
IRIX 5.3	IRIX+5.3 IRIX+5.1 IRIX	**`IRIX+5.1`**		IRIX+5.1

Current Operating System	Available Instances	**-f** argument	*$UPS_ EXTENDED_ FLAVOR*	Selected Instance
IRIX 5.3	IRIX+5.3 IRIX+5.1 IRIX	**IRIX+**		IRIX+5.3
IRIX 5.3	IRIX+5.3 IRIX+5.1 IRIX	**IRIX+**	debug	IRIX+5.3
IRIX 5.3	IRIX+5.3 IRIX+5.1 IRIX	**IRIX++debug**		none
IRIX 5.3	IRIX+5.3 IRIX+5 IRIX	**IRIX+5.1**		none
IRIX 5.3	IRIX+5.3 IRIX NULL	**IRIX+5.1**		none
IRIX 5.3	IRIX+5+dbg IRIX+5	**IRIX+5+dbg**		IRIX+5+dbg
IRIX 5.3	IRIX+5+dbg IRIX+5	**IRIX++dbg**		IRIX+5+dbg

2.1.9 Chains

We mentioned earlier that **UPS** supports multiple versions of software products on a machine. End users do not find it convenient to specify product version numbers each time they setup a product. This is especially true if product setups are needed at login. Most users want to run the latest, tested, approved version of products without having to keep track of the version numbers.

To allow users to specify the version of a product according to its *status* rather than by its version number, **UPS** supports *chains* to product versions. A chain is a **UPS** database entry (i.e. a clause in a product file) that points to a particular instance declaration in the database (an instance clause in the same product file). It must match the (extended) flavor and version of the instance declaration exactly. It “attaches” a chain name to a product instance, thereby tagging the product instance according to its status.

Five statuses, or chains, have been defined for use: current, new, test, old, and development.

Chain Names, Options, and Usage

Chain	Option	Usage
current	**-c**	default version recommended for general use
new	**-n**	tested version that is not yet current
test	**-t**	version installed for testing
development	**-d**	version under development
old	**-o**	older version that was previously current

In **UPS** commands, the command line option associated with the desired chain (status) is used to specify the product instance to retrieve. Using chains is optional, but recommended. Both chained and unchained instances of a product may be declared to **UPS**; the user can still retrieve any instance, chained or not, by using its product version number.

Chain Definitions in Product Files

Chain definitions for a product are stored in its product file in the **UPS** database. Since there is one product file per product, there may be multiple chain definitions in a product file. Note that instances do not need to be chained, thus you can have more instance clauses (described in section 2.1.7) in a product file than chain clauses. On the other hand, you can set multiple chains to a single instance and thus have more chain clauses than instance clauses. We recommend that for each node or flavor, you attach only one chain to a given instance to simplify maintenance.

A chain definition starts with `Chain:` and ends with `end.` It includes:

- chain name
- flavor (simple or extended)
- version (also confusingly called the "instance" internally to **UPS**)
- date of installation
- installer's name in quotes

Here is an example of two chain clauses in a product file. The **UPS** database in this example resides on a UNIX cluster with multiple operating systems, thereby allowing an instance chained to current for each operating system:

```
Chain:
       current OSF1+V3 v2_7    9-26-96 "mengel"
end.

Chain:
       current IRIX+5  v2_7    9-26-96 "mengel"
end.
```

Looking at the first chain clause, if your operating system is OSF1, and **`funame -r`** matches V3, then when you run[1]:

% setup [-c] ***product***

1. For **setup**, the default chain option is **-c**.

you will get the current instance of the product installed on your OSF1 system, which is version 2_7 as built for extended flavor (OS type) OSF1+V3.

All other information about a chained instance (e.g., product root directory, use requirements, etc.), is stored in the instance clause of the product file.

2.1.10 Product Dependencies (Use and Build Requirements)

Many **UPS** products rely on other products being installed, declared and setup for proper functioning or for use of special features. These are generally referred to as *product dependencies* or *use requirements*[1]. The coupling of products with their dependencies facilitates product setup. Users need only setup a single **UPS** product to access any and all of the products listed as its dependencies. Use requirements are setup recursively by default. Each of the dependencies can be setup in its own right, as well.

Requirements can exist across databases. The databases must be included in *$PRODUCTS* for this to work. When you declare a dependency, **UPS** searches the databases in *$PRODUCTS* and uses the first instance it finds; you cannot specify which database a dependency resides in.

Multiple levels of requirements are possible. As an example, the mail product **exmh** has several use requirements, one of which is **www**. **www** in turn has use requirements.

General Requirements Syntax

Use and build requirements are specified in a particular format in **UPS** product files, on the command line, and in **UPD**. The syntax is:

`[+|-] <|=|> [-jeb]` ***required_product*** **`[`*version*`]`** **`[`*chain*`]`** **`[`*flavor*`]`**

The elements of this format are:

`+|-` (used on the command line for **`ups modify`** only) means *either* **`+`** or **`-`**:

- **`+`** add this requirement to the existing list of requirements; used with **`<`** or **`=`** (e.g., **`+<`**)
- **`-`** remove this requirement from the existing list of requirements; used with **`<`** or **`=`** or **`>`** (e.g., **`-<`**)

`<|=|>` means **`<`** *or* **`=`** *or* **`>`**:

- **`<`** the requirement needs to be setup before the main product
- **`=`** the requirement can be setup before, at the same time as, or after the main product
- **`>`** the requirement needs to be setup after the main product

`-j` means "this required product only, and none of its dependencies"; recursive behavior disabled (use requirements are setup recursively by default)

`-e` means "set the environment variable *UPS_EXTENDED* "on" for this requirement". *UPS_EXTENDED* is an on/off type variable. The requirement's file `setup.`*`$UPS_SHELL`* may contain dependent functionality that is triggered by this option. In order to enable the dependent functionality (i.e. to set the variable "on"), first this flag (**`-e`**) must be set at the time that the parent product is declared with this build requirement, and secondly the **`-e`** option must be specified in the **`setup`** command for the main product.

---

1. There are also such things as build requirements. These need to be in place only for building the main product, and are usually omitted on end-user systems.

A note must be made to address the case of *recursive dependencies*[1] (i.e. sublevels of dependencies that are automatically setup along with their parent product): Once *UPS_EXTENDED* is turned off for the main product or a dependent product, it is turned off for all of that product's subsequent dependencies. For example, if A's dependent product B was declared *without* **-e**, but B's dependent product C was declared *with* **-e**, then even when A is setup with the **-e** flag, *UPS_EXTENDED* is *not* set for C, and its dependent functionality is suppressed.

It can remain on for parallel dependency branches (recall that the main product can have multiple dependencies, as can one or more of its dependent products; each product will have been declared individually with or without the **-e** flag).

-b used in this position means "set the environment variable *UPS_BUILD* "on" for this requirement". Like the **-e** option, this is an on/off type variable, which enables functionality that is tied to it in `setup.`*`$UPS_SHELL`*. In order to enable the dependent functionality (i.e. to set the variable "on"), first this flag (**-b**) must be set at the time that the parent product is declared with this build requirement, and secondly the **-b** option must be specified in the **setup** command for the main product.

The ***version***, ***chain*** and ***flavor*** specifications are used in the requirement specification only as necessary to uniquely identify the instance of ***required_product*** to be used. If no version or chain is specified, the current instance is assumed by default. If no flavor is specified, **UPS** will make a "best guess" according to the following rules:

1) If both the requirement and the main product are entered on the command line without a flavor, then the system uses the rules described under *Instance Selection Algorithm* in section 2.1.8.

2) If a flavor is specified for the main product and the requirement has an entry in the database with the same flavor, this flavor is used.

3) If a flavor is specified for the main product and the requirement does not have an entry with this flavor, NULL is used.

Requirements Syntax on a Command Line

When specified on a command line (for **ups declare** or **ups modify**), requirements are entered as the argument to either the **-u** or **-b** option (for *use* or *build* requirement), in a double-quoted string, using the syntax described above. The command line thus has the syntax:

% *ups_command* [*options*] -u|-b "[+|-] <|= [-jeb] *required_product* [*version*] [*chain*] [*flavor*]" *product version*

where:

-u|-b means *either* **-u** or **-b**:

-u the accompanying argument specifies one or more use requirements

-b the accompanying argument specifies one or more build requirements[2]

Command Line Examples

In all the following examples, we have left off the **-r *product_root_directory*** option. This is fine only if the product instance has previously been declared to the database.

1. Use requirements are always used recursively; build requirements never are.
2. When used in this position, **-b** does *not* set the environment variable *UPS_BUILD*.

% ups declare -c -u "< -j *req_prod* v*x_y*" prod1 v3_8

Declares **prod1** v3_8 as current with the use requirement ***req_prod*** v***x_y***. No dependencies of ***req_prod*** are to be considered dependencies of **prod1** (**-j**).

% ups declare -u "< -e *req_prod* v*x_y*" prod1 v3_8

Adds version v3_8 of **prod1** to the database. The use requirement ***req_prod*** version v***x_y*** must be setup before **prod1**. If **prod1** is setup with the **-e** option, the environment variable *UPS_EXTENDED* will be set for ***req_prod*** too.

% ups declare -u "< *req_prod* v*x_y* SunOS,= *req_prod2*" prod1 v3_8

Adds **prod1** v3_8 to the database with two use requirements: ***req_prod*** v***x_y*** of flavor SunOS must be setup before **prod1**, and the current instance of ***req_prod2*** must be setup before, at the same time as, or after **prod1**.

% ups modify -u "+< *req_prod*" prod1 v3_8

Modifies the use requirements of **prod1**; in addition to any previously defined use requirements, the current instance of ***req_prod*** must be setup before **prod1**.

2.2 Basic Product Maintenance Operations: A How-To Guide

Several of the **UPS** operations are typically done through the **UPD** menu while installing a product instance. The **UPD** menu operations are covered in Chapter 3. These **UPS** operations can also be done independently of **UPD** via the **UPS** command line interface. Additional **UPS** shell commands are provided to perform actions that are not appropriate to include in the **UPD** menu. In this section, you will learn how to run the **UPS** shell commands geared towards end users, product installers and product maintainers.

Man pages are available for each of the **UPS** commands, and there you will find an exhaustive list of options and other information. Here we attempt to explain how and why you would use the commands. All the **UPS** commands (except **ups initd**) are equipped with the **-v** (verbose) option so that you can print informational messages as the command executes.

For all of the **UPS** commands, you must be aware that the options you'll need to specify depend upon your environment. The set of available options varies slightly from command to command, however in general, these rules apply:

- If your *PRODUCTS* environment variable is set to multiple databases, you may need to specify which database to modify. The **-z** option, available for many of the commands, takes the database specification as its argument.

  The environment variable *DPRODUCTS* can be used to point to a particular database for use by the **ups declare** and **ups modify** commands. The **-z** option on any given command overrides *$DPRODUCTS*.

  In addition, there is a **-Z** option available for the **ups declare** command. It is described in section 2.2.2.

- If your database is on a multi-flavor cluster you usually need to specify the flavor via the **-f** option to fully identify the instance. If yours is a mixed cluster (includes two or more nodes of the same operating system with different releases) you must specify the extended flavor. (If a product's version number is specific to your platform, you could use that instead.) In general, we recommend that you specify the extended flavor to be safe.

For commands that modify the database, failing to specify the flavor may result in your product instance being declared with a flavor of NULL.

- If there is more than one instance of a particular product installed on your machine, or in the case of a cluster, more than one instance for a single flavor (extended or not), you need to specify the desired instance uniquely by including its version or chain. If a current instance exists, it will be used as the default. No option is associated with the version; it must appear immediately after the product name on the command line. The commands **`ups declare`**, **`ups modify`** and **`ups tailor`** always require specification of version.

Many of the examples in this section are shown on two lines for easy readability. We've used the UNIX standard backslash (****) when a second line follows.

2.2.1 Setup and Unsetup a Product Instance

Setup

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]. **`setup`** makes the appropriate changes to your software environment in order to make the product available for use. At a minimum it sets the environment variable *$PRODUCT_DIR* to point to the root directory of the product instance and sources the file `$PRODUCT_DIR/ups/setup.$UPS_SHELL` if it exists. By default, the use requirements are setup recursively along with the main product unless the **`-j`** option is used to restrict the setup to *just* the specified product and none of its dependencies[2].

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 www
```

The current instance is, in other words, the default. To setup any other chained instance, include the chain flag 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
```

As simple as it is to use in everyday situations, **`setup`** contains some subtleties. It is equipped with a host of options that allow you to specify various parameters. We refer you to the man pages for specifics; here we attempt to provide a summary/clarification of the available options. You can specify:

- which chained instance of the product to setup (used in example above)
- which use or build dependencies to setup along with main product (you cannot setup both)
- an option string to pass to the setup script via the environment variable *UPS_OPTIONS*
- enable any optional functionality in `setup.$UPS_SHELL`

1. There are exceptions to this rule; in particular, you should never setup the core FUE products **futil**, **fulib**, **funkern** and **shells**.
2. **`setup`** also comes equipped with a **`-b`** option that allows you to setup the build requirements *instead of* the use requirements. Build requirements are not setup recursively; in fact they are not generally distributed with the products.

☞ 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 -e prod1`

Sets up the instance of product **prod1** chained to development, and all of **prod1**'s use requirements. Sets *UPS_EXTENDED* on for **prod1** and for any of its use requirements that were declared with the **`-e`** option.

`% setup -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.

`% setup -v prod1, -e prod2 v1_0, -b prod3`

Sets up the current instance of **prod1** (and **prod1**'s use requirements) with verbose on; sets up v1_0 of **prod2** with *UPS_EXTENDED* defined (and **prod2**'s use requirements; also sets *UPS_EXTENDED* for any of its use requirements that were declared with the **`-e`** option), and sets up the current instance of **prod3** along with its build requirements.

Unsetup

`unsetup` sources the script `$PRODUCT_DIR/ups/unsetup.$UPS_SHELL` which in general 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 (e.g., **www**), simply type:

`% unsetup www`

Any use-dependent products get deactivated (unsetup) automatically at the same time unless the **`-j`** or the **`-b`** option is used:

- **`-j`** unsetup *just* the specified product and none of its dependencies
- **`-b`** unsetup the build requirements of the product (useful only when the product was setup with the **`-b`** option)

The man pages list other options, but in practice, unsetup scripts aren't coded to exploit them.

2.2.2 Declare a Product Instance to the UPS Database

A product instance must be installed on the system before it can be declared to the **UPS** database. Declaring a product instance makes it known to **UPS**, and therefore retrievable within the **UPS** framework. If a product file already exists for the product (i.e. other instances of the product are already declared), **UPS** simply adds an instance clause (and possibly a chain clause[1]) to the existing file. If not, it creates the product file with the clause(s).

This operation is most easily accomplished via the **UPD** menu interface (see "UPS DECLARE" in section 3.3). If you choose to use the command line interface, see the man pages (under **`man ups`**) for more information. The command syntax is:

`% ups declare [chain] -r` ***product_root_dir*** **`[`*****other options*****`]`** ***product version***

At a minimum, the product name, product version, and the product root directory (argument to the **`-r`** option) must be specified. You may also need to include the **`-z`** and/or **`-f`** options as described at the beginning of section 2.2.

The **`ups declare`** command is also uniquely equipped with a **`-Z`** option for database selection in the case of multiple databases. It takes no argument but does request confirmation from you. **`-Z`** selects the first valid database it finds according to the following rules:

1) the first database listed in *$PRODUCTS* in which the instance being declared already exists

2) the first database listed in *$PRODUCTS* in which the product is declared as current

3) the first database listed in *$PRODUCTS* in which any instance of the product exists

4) the first database listed in *$PRODUCTS*

If you want to declare a chain to the product instance, include the appropriate option in the command (see section 2.1.9 for a listing). You can also include use and/or build requirements (see section 2.1.10).

In executing this command, you will be warned if dependent products do not exist in the database. The declaration will still succeed (assuming everything else is correct), however you will need to individually install and declare all dependent products prior to running a successful setup on the product.

Examples

`% ups declare -f IRIX -r /usr1/prod1/v1_1 -z \`
`/usr/products/ups_database/declared prod1 v1_1`

Adds instance of **prod1**, of version v1_1 and flavor IRIX, to the database specified by `/usr/products/ups_database/declared`. The root directory of this instance is `/usr1/prod1/v1_1`. Use of the **`-z`** option indicates that *PRODUCTS* points to multiple databases, one of which must be explicitly selected.

`% ups declare -t -d -r /usr1/prod1/v1_1 prod1 v1_1`

Adds version v1_1 of **prod1** to the database. The root directory for this instance is `/usr1/prod1/v1_1`. This instance is also chained to both test and development.

`% ups declare -r /usr1/prod1/v1_1 -u "< -e` ***req_prod*** **`v`*****x_y*****`" prod1 v3_8`**

Adds version v3_8 of **prod1** to the database. The use requirement ***req_prod*** version v***x_y*** must be setup before **prod1**. If **prod1** is setup with the **`-e`** option, the environment variable *UPS_EXTENDED* will be set for ***req_prod*** too.

`% ups declare -r '$PRODUCTS_ROOT/prod1/v1_1' prod1 v1_1`

1. You can declare a product instance to the database and declare a chain to it in one step. We recommend using either *test* or no chain, in general, for the initial declaration of a product instance. It is best to declare an instance current in a subsequent step (see section 5.4.5).

Adds version v1_1 of **prod1** to the database. Its root directory is `$PRODUCTS_ROOT/prod1/v1_1`, where *$PRODUCTS_ROOT* will get translated at setup time (due to the apostrophes; use no quotes or double quotes to have the shell resolve it).

```
% ups declare -r '$PRODUCTS_ROOT/prod1/v1_1' -U \
  /usr/products/prod1/ups prod1 v1_1
```

Adds version v1_1 of **prod1** to the database. Its root directory is `$PRODUCTS_ROOT/prod1/v1_1`, where *$PRODUCTS_ROOT* will get translated at setup time (due to the apostrophes; use no quotes or double quotes to have the shell resolve it). The `ups` directory for this product will be located at `/usr/products/prod1/ups`. If the **-U** switch had not been specified, then the `ups` directory would have defaulted to `$PRODUCTS_ROOT/prod1/v1_1/ups`.

2.2.3 Declare a Chain to a Product Instance

To add a chain to a previously installed product instance, again we recommend use of the UPS DECLARE function in the **UPD** menu interface for simplicity. Otherwise use the **ups declare** command. In general, this does not change any existing chain, it adds a new one.

☞ However, if you have an instance already chained, and you wish to declare a new instance of a different version but the same (extended) flavor to the same chain, the pre-existing chain will be removed automatically.

When using the command line interface, you need to specify at a minimum the product name, its version and the chain option. For example, to declare a product as current, enter:

% ups declare -c ***product version***

You may also need to include the **-z** and/or **-f** options as described at the beginning of section 2.2.

A couple of examples will help to clarify how this works. In these examples we assume that the product instance has already been declared to the database either with no chain or with a different chain. Some of these commands will also work for declaring an instance initially to the database with a chain, however we refer you to section 2.2.2 for examples specific to that operation.

Example 1: Move product instance from test to current

In a typical situation, a product instance is initially declared as test to the database. Its product file may look similar to this:

```
Chain:
        test     OSF1+V3    v1_0   date  installer
end.

Intance:          "v1_0"   OSF1+V3   product_root_directory
                   date               installer
...
end.
```

At some point you want to take the product instance out of test and declare it as current. After issuing the command:

% ups declare -c -f OSF1+V3 ***product*** **v1_0**

you will end up with two chains (test and current), both pointing to the same instance. Note that the test chain is not removed.

Example 2: Change current chain to point to a new instance

Another frequently encountered situation is that in which you already have a version chained to current (v1_0, below), as shown:

```
Chain:
        current    OSF1+V3   v1_0   date  installer
end.

Intance:        "v1_0"  OSF1+V3  product_root_directory
                 date            installer
...
end.

Intance:        "v2_0"  OSF1+V3  product_root_directory
                 date            installer
...
    end.
```

and you want to declare as current a different version of the product (v2_0) for the same (extended) flavor. In order to change the existing current chain pointing to v1_0 to point instead to v2_0, issue the command:

% ups declare -c -f OSF1+V3 ***product*** **v2_0**

The previously current instance now has no chain (unless it also had other chains pointing to it, in which case they remain unchanged).

If there had been no instance previously chained to current, this command would simply create the current chain clause and point it to this instance of the product.

2.2.4 Remove a Chain from a Product Instance

To remove a chain from a product instance, you must use the **ups undeclare** command; this function is not available in the **UPD** menu interface. One common situation in which this is useful is when there is a version in test, and a newer version surpasses it and goes directly to current. The test chain should be removed in this case since it points to an earlier version than the current one.

The command has a simple syntax.

% ups undeclare ***chain product***

As an example, let's remove the *test* chain from the instance of **www** of flavor IRIX:

% ups undeclare -f IRIX -t www

You may also need to include the **-z** and/or **-f** options and/or the version as described previously.

2.2.5 Change the Chain to a Product Instance

In general, changing the chain to a product instance requires removing the pre-existing chain and adding a new one.

☞ When a current instance of a product already exists, if you declare a new instance of a different version but the same (extended) flavor as current, the current chain on the previously current instance gets removed automatically.

2.2.6 Undeclare and Remove a Product Instance

Getting Ready

Before removing anything, you should find out if any other products have the product instance in question declared as a use requirement. The following command will provide this information:

% ups depend -u parent ***product***

If the product instance is required, you may want to reconsider since the product's removal may affect the operation of these other products. The following output for the product **ghostview** lists several parent products (edited for brevity):

% ups depend -u parent ghostview

```
SYMBOLS
-------
C = current chain, D = devel chain
1 = /afs/fnal/products/ups_database/declared

           PARENT PRODUCT                            DEPENDENT PRODUCT
-----------------------------------            -------------------------------
C 1 conv2html     v2_0d     SunOS    ( Use )   ghostview     current
C 1 conv2html     v2_0c     IRIX     ( Use )   ghostview     current
  1 conv2html     v1_0      AIX      ( Use )   ghostview     current   AIX
...text removed...
  1 lyx           v0_10_7   AIX      ( Use )   ghostview     current   AIX
  1 mbone         v2_0c     SunOS    ( Use )   ghostview     current
...text removed...
C 1 www           v2_7      SunOS    ( Use )   ghostview     current
...text removed...
```

In this case it would probably be unwise to remove any of the listed instances of **ghostview**.

Once You're Ready...

To remove a product instance (i.e. an instance clause in a product file), you must use the **ups undeclare** command; this function is not available in the **UPD** menu interface. Fortunately, it has a simple syntax. You must specify the version of the instance, not its chain, in the command. **Specifying the chain removes only that chain, not the instance itself.** When an instance is removed, all chains that point to it are also removed.

In order to delete the product directory tree starting from the product root directory, use one of the options **-r** or **-R** with **ups undeclare** (**-r** queries you for confirmation, **-R** does not). You may also need to include the **-z** and/or **-f** options as described above.

This is the recommended procedure for removing product instances. Removing them manually does not ensure that all the files get deleted or that chains get updated properly, eventually leading to a fragmented products area.

Examples

`% ups undeclare -c -r -f IRIX+5 pine v3_91`

Removes the instance of the product **pine** version v3_91 for flavor IRIX+5, and any chains that point to it. Also removes the product root directory for this instance of **pine**; it queries for confirmation before doing so. The **-c** option is ignored.

`% ups undeclare -R pine v3_91`

Finds the best flavor match using the standard instance selection algorithm (section 2.1.8) and removes that instance of the product **pine** version v3_91 and any chains that point to it. Also removes the product root directory for this instance of **pine**; it does not query for confirmation before doing so.

This is a dangerous example! You may end up removing the instance for, say, OSF1+V3 when you really wanted to remove the one for OSF1!

2.2.7 Configure and Tailor a Product Instance

Some products require that you perform supplementary procedures during the installation process, for example copying files to other locations or creating needed files or directories. Procedures that can be completely automated are collected in a file called `configure`. Procedures that require input from the installer are collected in a file called `tailor`.

Configuring

For any product that requires it, the `configure` executable (usually a script, although it can be a compiled binary) is provided in the directory *$PRODUCT*`_DIR/ups`. In **UPS** *configuring a product instance* means sourcing this executable, if it exists, automatically when the product is declared (or if you run **`ups configure`** manually, which is not typically done).

When the product is undeclared, the associated executable *$PRODUCT*`_DIR/ups/unconfigure` is run automatically to undo what was done in `configure`.

The `configure` script may need to create links to the product root directory from other areas, typically from the `/usr/local/products` area. If this area is not identical for each node accessing the **UPS** database in which the product instance has been declared, then you will need to run the **`ups configure`** command manually on each node that mounts a unique `/usr/local/products` area. If you are not sure whether you need to configure a product instance on each node, read the `configure` file to see what it does.

Tailoring

Tailoring is the aspect of the product implementation that requires input from the product installer (e.g., specifying the location of hardware devices for a software driver package). If the product requires tailoring, the file *$PRODUCT*`_DIR/ups/tailor` is usually supplied in the format of an interactive executable (script or compiled binary). To *tailor a product instance* means to source this executable. If this executable exists, and if you use the **UPD** menu interface, then during the product declaration, a message is displayed indicating that tailoring is required. You must explicitly tailor the product instance using either the UPS TAILOR screen in the **UPD** menu interface or the **UPS** command **`ups tailor`**; tailoring is *not* performed automatically.

☞ Tailoring is generally allowed on any node of a cluster, however we recommend that you perform any node-specific tailoring from that node, or flavor-specific tailoring from a node of that flavor.

Beyond the usual options used to uniquely identify the product instance, the **`-O`** flag (upper case "o") is provided to specify an option string to pass to the tailor script if needed, as in the following example:

% **ups tailor -f IRIX+5.1+debug -d -O "*pass this string*" prod1**

In this example we are tailoring the version of **prod1** chained to *development* (**`-d`**) for the flavor IRIX+5.1+debug, and we're passing an option string to the tailor script.

If the product you're tailoring requires daemon processes to start and stop automatically, `start` and `stop` executables should be provided for you. The tailor process ensures that they get copied to *$PRODUCT_*`DIR/ups` where the **`ups start`** and **`ups stop`** commands find and execute them. Daemon process are further discussed in section 2.3.2 and Appendix C.

2.3 Advanced Product Maintenance Operations: A How-To Guide

The functionality discussed in this section is not available from the **UPD** menu interface. Man pages are available for each of the **UPS** commands, and there you will find an exhaustive list of options and other information.

Recall that for all of the **UPS** commands, you must be aware that some of the options you'll need to specify depend upon your environment. You may want to review section 2.2 where this was discussed.

2.3.1 Change an Instance Clause in the Product File

The command **`ups modify`** is provided to allow you to make changes to instance clauses in product files. You can use it to change, add, or delete items in an instance clause. This command string can become long and complicated since it requires that you specify the product instance as well as the elements in the instance clause to modify. Usually you only need to change a single element at a time.

Again, the minimum specification for the product instance depends on whether you've got multiple **UPS** databases and/or multiple product instances. The **`-z`** option is available for specifying the database, if necessary. Given a single product instance in a single database system, the product name will suffice.

An individual option (or flag) is provided for each of the possible elements in the instance clause, in order to identify the one to change. The option must be followed by the new value (or a blank space if you're removing the associated element).

Examples

% **ups modify -f SunOS -r /usr1/prod1/v1_1 prod1 v1_1**

Changes the root directory of **prod1** version v1_1, flavor SunOS to `/usr/prod1/v1_1`.

% ups modify -a none prod1 v1_1

Modifies the authorized nodes of **prod1** v1_1 to be *none*.

% ups modify -u "< b, = c" prod1 v1_1

Modifies the existing use requirements of **prod1** v1_1 such that product **b** must be setup before **prod1**, and product **c** can be setup before, concurrently with, or after **prod1**.

% ups modify -u "+< d" prod1 v1_1

Adds the current instance of product **d** to the existing list of use requirements for **prod1** v1_1; **d** must be setup before **prod1**.

% ups modify -f SunOS+5 -r /usr1/prod1/v2_0 -u " " prod1 v1_1

Changes the root directory of **prod1** v1_1 flavor SunOS+5 to `/usr1/prod1/v2_0`, and removes all its use requirements.

% ups modify -b "-< d v1_0 IRIX" prod1 v1_1

Removes product **d** v1_0, flavor IRIX from the existing build requirement list of **prod1** v1_1.

% ups modify -U /usr1/ups/prod1/ups prod1 v1_1

Modifies the `ups` directory of **prod1** v1_1 to be `/usr/ups/prod1/ups` instead of the default `ups` directory which would lie directly under the product root directory for **prod1** v1_1.

2.3.2 Install a Product as a UPS Daemon Process

Some products are appropriate to run as background processes (daemons) or servers. In this section we describe implementing a product as a daemon under **UPS** control.

Does Your UPS Implementation Support It?

When **UPS** was installed, it may or may not have been tailored to control daemon processes (see Appendix C). If it was, you have the option of using **UPS** for daemon process control. If not, you will need to use the standard facilities on your operating system (refer to your OS-specific manuals for instructions).

If you can use **UPS**, we recommend it for the following reason. When a new instance of a product is declared as current[1], **UPS** runs the new one automatically upon system reboot. There is no need to involve the system administrator.

You need to determine whether **UPS** was tailored to control daemon processes. If the files `$PRODUCTS/../startup/ups_startup.`*`$UPS_SHELL`* and `$PRODUCTS/../shutdown/ups_shutdown.`*`$UPS_SHELL`* exist, then **UPS** has been tailored to support daemon process control.

1. Well, usually current. Continue reading under *UPS Daemon Process Control Files* for more information.

Product start and stop Files

Products that are appropriate to run as daemons or servers (should) come equipped with two executables, `start` and `stop`, in their `ups` directory. Often these products also come with a `tailor` script (see *Tailoring* under section 2.2.7). Once the product has been configured and tailored properly, `start` is run at boot time to start the product and `stop` is run at system shutdown to stop it.

UPS Daemon Process Control Files

In order to make known to the system that your product is to be started at boot time, you will need to add a specific line of text to the appropriate control file. This line provides the actual start command for the product. Go to the `$PRODUCTS/../startup` directory[1]. There you may find one or more files with the name *node*`.products` (where *node* is one of the nodes in your cluster, e.g., `fsgi02.products`) and/or *flavor*`.products` (where *flavor* is one of the flavors in your cluster, e.g., `IRIX+5.products`). If you want your product to run as a daemon on a single node in your cluster, edit the file appropriate to that node, or create the file if it doesn't exist. If you want it to run on all nodes of a particular flavor, edit or create the file appropriate to that flavor. A line for a particular product can exist in more than one file, i.e. both the node-specific and the flavor-specific files. The system runs all these files at boot time and uses only the first start command it finds for a product.

Add a line of the following format to the appropriate file(s):

/bin/su –*login_id* –c "ups start [*options*] *product* [*version*]"

The ***login_id*** is that under which the program must be run. The login id used must be set either to source the file `fermi.cshrc` (`.shrc`) or to run **`setup ups`** so that other products can be successfully setup. The **`–c`** shown here reflects general usage; it can in fact be any chain flag. The chain flag must match the product instance chain. If the ***login_id*** is *root* and the product instance is declared as current, you can leave out the portion **/bin/su –*login_id* –c**, and the line can start from **`ups start`** (no quotes needed in this case). The options are detailed in the man pages for **`ups start`**.

Similarly, to tell your system to stop the daemon at shutdown, go to the directory `$PRODUCTS/../shutdown`. There you will find similarly named files, one or more of which you will need to edit. Add a line of the following format to the appropriate file(s):

/bin/su –*login_id* –c "ups stop [*options*] *product* [*version*]"

The same notes regarding the ***login_id*** and chain flag apply.

Make sure the mode of all the *node*`.products` and *flavor*`.products` files is set to 744. This ensures that the files will be executable by *root* and that they have appropriate permissions for avoidance of security holes.

Summary for Implementing a Daemon under UPS Control

1) Declare the product to the **UPS** database (if configuration is required it gets done in this step; see section 2.2.7).

2) Tailor the product (usually required).

3) Add a control line to the appropriate file in the `$PRODUCTS/../startup` directory.

1. If *$PRODUCTS* includes more than one database, use the `startup` directory in the database in which the currently running instance of **UPS** resides.

4) Add a control line to the appropriate file in the `$PRODUCTS/../shutdown` directory.

When the system is restarted, your daemon process should start running on the nodes you've designated.

2.3.3 Disable Automatic Start/Stop of Daemon Processes

To disable automatic start and stop of **UPS** products at boot time and shutdown, remove the file `ups_startup.sh` from the `$PRODUCTS/../startup` directory and the file `ups_shutdown.sh` from the `$PRODUCTS/../shutdown` directory.

2.4 Troubleshooting

Here are a few hints if things don't seem to work after installing a product or modifying the database.

- The command:

  `% setup -j` *product*

  will just setup the main product and none of its dependencies. This should help determine which file has the problem, the main one or a dependency.

- Using the **`-v`** option with **`setup`**:

  `% setup -v` *product*

  will print out verbose information.

- The command **`setup`** is actually an alias in the C shell family and a function in the Bourne shell family. It executes the binary `$UPS_DIR/bin/ups_setup`. This binary writes a temporary file which **`setup`** sources and deletes. If you just run the binary, by entering:

  `% $UPS_DIR/bin/ups_setup` *product*

  the temporary file is written to `stdout` (the file is not sourced in this case). You might want to redirect it to a file to examine this output. Expect it to be lengthy.

- Check file permissions. The scripts in the `$`*PRODUCT*`_DIR/ups` directory must be both readable and executable. The `$PRODUCTS/product` file must be readable.

Chapter 3: Installing Software Products on FUE-Compliant Systems

This chapter describes how to obtain **UPS** products from a remote distribution node and install them onto a local user node. All the available methods for accomplishing these tasks are covered, namely:

- the **UPD** menu interface
- the command line (shell) interface
- anonymous **ftp**

The information presented here assumes that the user node is FUE compliant at least to the level of Full FUE and that the utilities UNIX Product Distribution (**UPD**) and UNIX Product Support (**UPS**) are installed.

Because of the number of different workstations and software packages and the frequency of new versions of the software, it is the policy of Fermilab to provide software distribution in binary form, as opposed to source, wherever possible. This way, software versions are consistent and reproducible. **UPS** products are generally packaged as tar files.

A central UNIX product distribution node, upd.fnal.gov (usually refered to as simply upd), has been established on which the UNIX software products supplied and supported by the Fermilab Computing Division are stored as tar files. Throughout this chapter we've used upd as the assumed distribution node, however you can use the tools described in this chapter to get products from any source, as long as the products are in the correct format.

3.1 Getting Ready

3.1.1 Verify FUE Conformance Level

As mentioned in the introduction above, we've written this documentation assuming that your system already conforms to at least the Full FUE standard (see section A.1) and has **UPS** and **UPD** installed on it. In addition to **UPD** and **UPS**, Full FUE compliance assures that the Fermilab products **funkern**, **futil**, **systools** and **shells** are installed. To determine if these products are installed, use the **UPS** command:

`% ups list` ***product***

for each product. For example, if you enter:

```
% ups list systools
```

the system should return information similar to the following:

```
systools                          v4_3: current version
                                  Database: /usr/products/ups_database/declared
```

To system administrators:

Instructions for installing **UPD** and **UPS** and these additional Full FUE products can be found in Chapter 4.

If you are using an offsite machine and you're unsure if it is registered, try the following command and see if it works. If it works, the machine registered.

```
% rsh -l upd upd.fnal.gov ls
```

On some machines the syntax may be different, for example HP-UX uses **`remsh`** instead of **`rsh`**.

3.1.2 Select Appropriate Operating System

If you're installing a product for a single operating system in a mixed OS cluster, perform the installation from a node of the operating system for which the product instance is compiled. In other words, install the SGI instance from an SGI platform, then log in to the OSF1 platform to do the OSF1 installation, and so on. For some products it won't make a difference, but you can't know which ones a priori; therefore, always follow this prescription.

Why is this necessary? Products often have operating system-dependent requirements. For example some products need to have their root directory path hard-coded, and in a multi-OS cluster, this path contains an OS level. The convention we've adopted to address this requirement causes links to be created in `/usr/local/products` that point to the product root directory. The important issue is that typically there is a distinct copy of `/usr/local/products` for each operating system type in a cluster. Therefore, if you install from a node of the wrong operating system type, you may (1) overwrite the binaries for the current node's OS, and (2) make the binaries for the other OS inaccessible.

3.1.3 Determine Installation Account and Verify Directory Permissions

Check with your system administrator to find out what account(s) can be used for product installation on your node or cluster[1]. Your personal account may not have the appropriate permissions.

Which Directories are Involved?

You need write access to the directory specific to the flavor under which the product directories go, for example:

`/usr/products/AIX`

You also need write access to the directory above the product root directory[2], for example:

`/usr/products/AIX/emacs`

You will, of course, need write and execute permission for the product root directory, for example:

1. If you are the system administrator, see section 4.1.2 for a discussion of install accounts and file system semantics.
2. Note that the product root directory itself should be specific to the product version.

`/usr/products/AIX/emacs/v19_34`

which should happen automatically if the parent directory is set up properly.

You also need write access to the **UPS** database in which you intend to declare the product instance (this is where the product file goes), and to the man page and Info directories as set during the installation of **UPS** (see sections 4.1.3 and 4.2.2).

Depending on the product, write access to a few additional directories is sometimes required when the product is declared current. For example, occasionally a link gets made in `/usr/local/products` to the product root directory, which implies you need write access to `/usr/local/products`. This is true in particular for third party products such as **tcl** and **emacs**. Write access to directories under `/usr/local/` (for example `bin`, `etc`, `systools`) is required for the FUE packages **futil**, **funkern**, **systools**, and **shells**.

How to Test for Write Permission

To test for write permission to the directories, change to each directory in turn and run the command:

`%ls -ald`

The returned record will look similar to this:

```
drwxrwxr-x  22 root     oss          512 Nov  6 10:57 .
```

Is it group-writable? In this example it is (`drwxr`**`w`**`xr-x`). Next look for the group id (fourth space-separated field). The group id is `oss`. You need to belong to this group to write to the directory. To see the list of groups that you belong to, run the command **`groups`**. If you are not in the appropriate group, you need to login as *root*, as the owner (if other than *root*), or as a member of the group, to write to the directory.

3.1.4 Choose the Product to Install

Probably by this point you have determined which product you want to install. If not, and you need information on the available products in order to find one that fits your need, try the *Computing Division home page* on **www**, and select *Documentation and Software* or one of its subheadings. If there is not enough information there for you to make a decision, ask around or contact the helpdesk at *helpdesk@fnal.gov.*

Some products may already be on your system. To list all the products already installed in the **UPS** database on your local system, enter at the shell prompt:

`% ls $PRODUCTS`

This command only returns the product names (as listed in the product files).

Beware that this may also list products that are out-of-date; use **`ups list`** or check the contents of the product files.

3.1.5 Create Parent Directory for Product Root Directory

You first need to know where on your system **UPS** products are installed, if a particular area or areas have been designated. Your system administrator should be able to tell you. For purposes of this discussion, we will assume they reside in the directory `/usr/products`. If you work on a multi-node cluster consisting of dissimilar hardware platforms (and thus OS types), then underneath `/usr/products` we will assume there is a subdirectory for each OS type, for example:

`/usr/products/SunOS`

`/usr/products/IRIX`

`/usr/products/OSF1`

and if your cluster includes machines of the same OS type but different releases, you may find directories like:

`/usr/products/IRIX+4` and `/usr/products/IRIX+5`

where the +***number*** indicates a release of the OS beginning with that number, e.g., `IRIX+5` might be IRIX release 5.0 or 5.5.1. The **`funame`** command and its options can help you determine which subdirectory is appropriate for installation of a product on your machine. The command:

```
% funame -s
```

returns your OS type (e.g., `SunOS` or `IRIX`), and the command:

```
% funame -r
```

returns your OS release (e.g., `5.5.1`).

Before installing a product instance, you need to make sure a subdirectory exists for the product you're going to install. For example:

`/usr/products/SunOS+5/ghostview`

If other instances of the same product have been installed previously, this directory will exist, and the product instances will have been installed underneath it in separate version directories. These are the individual product root directories. For example, under `/usr/products/SunOS+5/ghostview` you might have a series of directories like:

`v1_5c  v2_0   v4_0`

When you install an additional product instance, a new product root directory specific to that instance will be created for you during the unwind process; you can create it ahead of time, but you do not need to.

3.2 Product Installation Overview

Generally **UPS** products are stored as tar files on the distribution node. The available software falls into three categories:

- software written in-house, and supported by the provider or by the Computing Division
- public-domain software provided (and usually supported) by the Computing Division
- third party proprietary products with licensing restrictions

You are not expected to know a priori into which category your selected product falls. All products can be installed using the same tools. The various installation methods all involve some parts of **UPS** and/or **UPD**.

- The easiest way to install a product is to use the **UPD** menu interface. It accesses the complete set of commands required for installation and for file extraction. This includes a combination of **UPS** and **UPD** commands.

- The other easy way is to use anonymous **ftp** for copying product tar files over from the distribution machine, and to finish installing a product instance locally via the **UPD** menu interface.
- **UPD** and **UPS** commands can be run from the shell prompt to copy tar files over from a distribution node and then to fully install the products on the local node. The disadvantage is that there are lots of commands and command options to remember. This feature is useful for inclusion of commands in scripts.

Whichever method you use, products are installed in a maximum of six steps:

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

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

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

4) Declare the product to the **UPS** database.

5) Tailor the product.

6) Declare the product as "current" to **UPS**.

Very few products require tailoring (see section 2.2.7); if you use the **UPD** menu interface, it notifies you if tailoring is needed. If tailoring is not required, you have the option of declaring the product to the **UPS** database and declaring it as current in the same step[1], although we recommend initially declaring products as test.

If the product you select is proprietary software, its tar file on the upd distribution node will not contain the product, but rather a set of instructions for obtaining authorization to use the product. Once the authorization is granted, you will be given instructions for obtaining the product. See section 3.7 for more information.

3.3 Using the UPD Menu Interface to Install a Product

The **UPD** 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. **UPD** provides functionality to list products, list contents of a tar file, copy a product instance or copy an individual file from a product tar file. Once on the local node, **UPD** can also be used to unwind the tar file and declare the product instance to the **UPS** database.

As mentioned above, it takes four to six steps to install a product. This section first walks you through the steps for the simplest type of installation using the **UPD** menu interface, and then discusses the additional steps required for more complicated installations and other operations.

☞ We are documenting the most recent version of **UPD** available as of this writing, v2_8d (for SunOS+5). Significant improvements have been made with respect to older versions in the areas of wording and appearance, default values provided, and message handling. If your system is running an older version of **UPD**, we strongly recommend upgrading.

1. The only cases in which you could not do this are those for which the `current` script doesn't redo the necessary configuration in the `configure` script. This is explained in section 5.4.5.

3.3.1 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 products and directories 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**.

To access the **UPD** menu, first setup **UPD**:[1]

`% setup upd`

To run it, enter:

`% upd`

The following main menu will appear:

```
 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 :
```

1. If the version of **UPD** that you want has not yet been declared as current on your system, include the appropriate chain flag or version in the **`setup`** command.

3.3.2 UPD Menu Parameters

The following is a comprehensive list of the parameters you will find on the **UPD** menu interface screens. They appear in roughly the order you'll encounter them. Many of these correspond to command line options for the set of **UPS** and **UPD** commands. Further information can be found in the man pages for **`ups`** and **`upd`**.

☞ A note on the default values: As you progress through an installation session, several of the parameter values that you enter get passed to subsequent screens so that you don't need to retype them each time.

Parameter and Default Value	Screen(s)	Description
remote node name Default: `upd.fnal.gov`	UPD LIST, UPD COPY, UPD TAR CONTENTS, UPD EXTRACT	The name of the remote distribution node from which you want to obtain product information or a tar file. Other nodes may be set up for product distribution in addition to the Fermilab central one (upd.fnal.gov), so you may need to enter a different node. Check with your system administrator.
product name Default: none	UPD LIST, UPD COPY, UPD TAR CONTENTS, UPD EXTRACT, UPS DECLARE, UPS TAILOR	The product name must be exactly the **UPS** product name known to the database. The asterisk (`*`) can be used as a wildcard to replace one or more characters in the name. It does not need to be quoted. Example format: `g*`
version or chain (e.g., v1_0 or current) Default: any previously entered version or chain; otherwise `current`	UPD LIST,UPD COPY, UPD TAR CONTENTS, UPD EXTRACT,	The version or chain of the product instance tar file(s) on the remote node that you want to work with.
version (e.g., v1_0) Default: any previously entered version; otherwise none	UPS DECLARE, UPS TAILOR	The version of the product now on your local node that you want to declare to the **UPS** database, or to tailor.
flavor Default: (extended) flavor of your system	UPD LIST,UPD COPY, UPD TAR CONTENTS, UPD EXTRACT, UPD GET REQUIREMENTS, UPS DECLARE, UPS TAILOR	Your system's extended flavor up to the first period. Examples: `SunOS+5` for SunOS v5.5.1 or `OSF1+V3` for OSF1 V3.2

Parameter and Default Value	Screen(s)	Description
temporary local tar file name Default: its name on the distribution node, preceded by the directory defined in *$TMPDIR* (or `/tmp`, if not set)	UPD COPY, UPD UNWIND	The name of the tar file that you copy into a temporary directory on your local node, prior to unwinding it. The filename on the distribution node is of the form: `product_x_flavor.tar` where `x` is either the chain or the version. On the local node, it puts the file by default in *$TMPDIR* (if not set, it uses `/tmp`) and retains this filename.
local product root directory Default: none	UPS DECLARE, UPD UNWIND	The permanent directory on your local node in which to unwind the tar file and maintain the product instance.
ups sub directory Default: none	UPS DECLARE	This specifies where the `ups` subdirectory for the requested product is located. By default, it is located directly under the product root directory (`$PRODUCT_DIR/ups/`); if a different location is specified, this parameter overrides the default.
use requirements Default: any use requirements found either on the remote node or in the unwound tar file.	UPS DECLARE	Additional products that must be installed, declared and setup before using the specified product instance. Example format: `(< ximagetools,current,SunOS+5)`
build requirements Default: any build requirements found either on the remote node or in the unwound tar file.	UPS DECLARE	Products required for building (compiling and linking) the selected product instance. Example format: `(< ximagetools,current,SunOS+5)`
ups declare chain flag Default: `-t`	UPS DECLARE	If this field is left at the default value of `-t` or changed to a different chain flag, the product instance is automatically declared as test (or other chain) to the local **UPS** database. If blank, the instance is declared with no chain.
copy files toInfo Default: `-I`	UPS DECLARE	This copies all the files in `$PRODUCT_DIR/ups/toInfo` into the area specified by the **UPS** node tailoring data file `$UPS_DIR/ups/default.dat`.
verbose Default: blank	UPS DECLARE	If set to `-v`, informational messages are displayed as the declaration proceeds

Parameter and Default Value	Screen(s)	Description
local **UPS** database Default: first listed directory in *$PRODUCTS*	UPS DECLARE, UPS TAILOR	If multiple databases are used on the local system, this parameter specifies which local database to operate on.
tailor current Default: `-c`	UPS TAILOR	Normally you don't need this field. If it is filled in, it must point to the same instance as the specified version.
filename within remote tar file Default: none	UPD EXTRACT	The name of an individual file that you want to extract from a remote tar file.
local file specification Default: none	UPD EXTRACT	The file specification on your local node given to a single file that is to be extracted from a tar file on the remote node.
remote database, if different from default Default: none	UPD LIST, UPD COPY, UPD TAR CONTENTS, UPD EXTRACT	If you are using a different distribution node, enter the specification for the directory on that node containing the tar file(s) you want. This is not needed for the default distribution node.

3.3.3 Simple Installation Example

To illustrate the simplest install procedure, we will install the instance of the product **pine** that has been declared as current for SunOS+5 in the distribution database. If you just want the product instance for the right flavor that is declared as current in `KITS`, you don't need to know the product's version number, and therefore you don't really need to run option 1 from the main menu, `upd list`. We'll run it here so you can see what the output looks like.

UPD LIST

From the main menu, type **`1`** at the `Enter choice :` prompt to get to the `UPD LIST` menu printed below. Here you can set parameters to narrow the list of available product instances to be returned by **UPD**, and determine the instance of the product that you want to install.

```
UPD LIST:   * implies value required
        *1      [upd.fnal.gov] remote node name
         2      [pine] 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 :
```

To list the current version of **pine**, you only need to fill in the product name before executing the list; the other values are defaulted. You should get a response similar to the following[1]:

```
 UPD sending request to remote node...
No remote authentication
                Distributable Versions of pine
 current     --> v3_95             Flavor: SunOS+5
No remote authentication UPD sending request to remote node...
```

and it returns you to the UPD LIST screen with the parameters you used still set.

☞ Just a note: If you want to see instances appropriate for all releases of your operating system, enter the simple flavor followed by an asterisk, e.g., `SunOS*`, in the flavor field,

Next, copy **pine** version v3_95 (the current instance for this flavor) to your local database.

UPD COPY

From the main menu, enter **2** to arrive at the UPD COPY menu, shown below. This function copies the tar file associated with the selected product instance on the remote node to the "temporary local tar file name" specified here. **UPD** defaults the tar file specification to include the product name, the chain or version and the flavor, and assumes it should be copied into the directory defined by the environment variable *TMPDIR* (**UPD** uses `/tmp` if this is not set).[1]

☞ Note that *TMPDIR* can be set to a location that has a greater capacity than `/tmp`.

All the information you need is already filled in from the UPD LIST step, so you can just execute the function unless you wish to change any parameters.

```
UPD COPY: * implies value required
        *1       [upd.fnal.gov] remote node name
        *2       [pine] product name
        *3       [v3_95] version or chain (e.g., v1_0 or current)
         4       [SunOS+5] flavor
        *5       [/tmp/pine_v3_95_SunOS+5.tar] temporary local tar file name
         6       [] remote database, if different from default

         e       execute upd copy
         q       quit and return to main menu
```

If you didn't run UPD LIST you need to enter the parameters. If you want to copy over the current version of a product for your OS and you don't know the product version number, just enter `current` for option 3.

If the directory specified in the path for `temporary local tar file name` does not exist, **UPD** attempts to create the directory and will only succeed if the parent directory already exists (the default `/tmp` doesn't have one) and the account from which you're working has write access to that directory.

Upon execution, **UPD** provides some minimal run-time information and then returns to the main menu:

```
Enquiring tar file name from remote node ...
No remote authentication
Copying tar directory ...
No remote authentication
No remote authentication

UPD COPY SUCCEEDED: continue with unwind
```

Among these messages you may also see:

---

1. Ignore the "No remote authentication" messages. The default remote distribution node currently uses the AFS file system which is responsible for these messages.
1. Make sure you can write to the temporary area, or change the tar file specification.

```
WARNING: Product not being distributed from Valid node
```

This warning can be ignored. It just means that the entry for `remote node name` is not the same as the name stored in the tar file. The upd node can also be called KITS or the actual node name (which we do not quote here since it can change), and it may appear as one of these in the tar file or the database on upd.

From the main menu you can fork a shell (option 9) and verify that the tar file you copied is there:

```
$ cd /tmp
$ ls -l pine*
-rw-r--r--   1 aheavey  g020     10076160 Apr 11 11:37 pine_v3_95_SunOS+5.tar
$ exit

Welcome back to upd
```

Next, you need to unwind the tar file into its permanent directory, called the "product root directory".

UPD UNWIND

From the main menu, enter **3** to arrive at the UPD UNWIND menu, shown below. Sometimes specific information regarding the installation procedure has been provided by the product developer. If so, this information resides in the file on the distribution database `$`*PRODUCT*`_DIR/ups/INSTALL_NOTE`, and will be automatically typed to the screen when the unwind is executed (see sections 3.3.4 and 5.4.5)[1]. The file `$`*PRODUCT*`_DIR/ups/declare.dat` is also searched for the product's use and build requirements, if any. You are notified of any use requirements that are missing in your database.

```
UPD UNWIND: * implies value required
        *1       [/tmp/pine_v3_95_SunOS+5.tar] temporary local tar file name
        *2       [/home/dcdsv0_lv0/aheavey/ups_play/products/SunOS+5/pine/v3_95
] local product root directory

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

This function unwinds the local tar file (which was copied over in the previous step) into the local product root directory specified in this step. If the product root directory does not exist, **UPD** attempts to create the directory. This will only succeed if the *parent to the parent* directory already exists (in this example `/home/dcdsv0_lv0/aheavey/ups_play/products/SunOS+5`), and the account from which you're working has write access to it.

Once you execute the function, **UPD** returns many lines of run-time information (the majority of file records have been deleted for brevity in the sample output shown below). The first line appears only if the product root directory has not previously been created.

```
UPD creating root directory /home/dcdsv0_lv0/aheavey/ups_play/products/SunOS+5/
pine/v3_95...
UPD unwinding tar file ...
x pine, 0 bytes, 0 tape blocks
x pine/CPYRIGHT, 2674 bytes, 6 tape blocks
x pine/README, 1263 bytes, 3 tape blocks
```

1. If you don't use the **UPD** menu to unwind the tar file, the `INSTALL_NOTE` will not be typed to the screen automatically.

```
x pine/bin, 0 bytes, 0 tape blocks
...text removed...
x bin/pine, 2693552 bytes, 5261 tape blocks
x bin/pilot, 221344 bytes, 433 tape blocks
x bin/pico, 222624 bytes, 435 tape blocks
x etc, 0 bytes, 0 tape blocks
x include, 0 bytes, 0 tape blocks
 UPD UNWIND succeeded   The temporary tarfile "/tmp/pine_v3_95_SunOS+5.tar" sti
ll exists
   Do you wish to delete /tmp/pine_v3_95_SunOS+5.tar(y or n)[y]:
```

Press return (for `y`) to delete the temporary file unless there is some particular reason not to in your case. **UPD** returns:

```
  /tmp/pine_v3_95_SunOS+5.tar deleted


 UPD UNWIND SUCCEEDED: continue with get requirements
```

and brings you back to the main menu. Next, find out if the product has any use or build requirements.

UPD GET REQUIREMENTS

From the main menu, enter **`6`** to arrive at the UPD GET REQUIREMENTS menu, shown below. This function is designed to furnish information only; it performs no aspect of installation for any requirements it finds. Again, the parameters are filled in, just enter **`e`** to execute.

```
UPD GET REQUIREMENTS:   * implies value required
        *1       [upd.fnal.gov] remote node name
        *2       [pine] product name
        *3       [v3_95] version or chain (e.g., v1_0 or current)
         4       [SunOS+5] flavor
         5       [] remote database, if different from default

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

For **pine**, **UPD** returns the information:

```
 UPD sending request to remote node...
No remote authentication
 pine v3_95 has no use requirements on upd.fnal.gov
 pine v3_95 has no build requirements on upd.fnal.gov
No remote authentication
```

and returns you to the UPD GET REQUIREMENTS menu. We see that there are no use or build requirements for this product. We can continue on to UPS DECLARE.

UPS DECLARE

From the main menu, enter **`7`** to arrive at the UPS DECLARE menu, shown below with the defaults that we expect by now, plus a couple of new ones. This function calls **`ups declare`**. See the man pages for **`ups declare`** (under **`man ups`**) for detailed information on the processes that occur when this is run. Also see sections 2.2.2 and 2.2.3.

```
UPS DECLARE: * implies value required
        *1      [pine] product name
        *2      [v3_95] version (e.g., v1_0)
         3      [SunOS+5] flavor
        *4      [/home/dcdsv0_lv0/aheavey/ups_play/products/SunOS+5/pine/v3_95
] local product root directory
         5      [] UPS sub directory
         6      [/home/dcdsv0_lv0/aheavey/ups_play/ups_database/declared] loca
l UPS database
         7      [] use requirements
         8      [] build requirements
         9      [-t] ups declare chain flag
        10      [-I] copy files toInfo
        11      [] verbose (-v)

         e      execute ups declare
         q      quit and return to main menu
 Enter choice :
```

If the default in parameter 9 is left as `-t`, **UPS** chains the specified instance to test. If you want the product instance to go directly to current[1], change this to `-c`. Leaving `-I` in 10 will copy all the files in `$`*`PRODUCT`*`_DIR/ups/toInfo` into the area specified by the file `$UPS_DIR/ups/default.dat` if you're declaring the product as current. You must have write access to this area in order for this to work. If the product instance is not being declared as current, leaving `-I` in 10 has no effect.

For the present example, we leave the default `-t` in 9 and `-I` in 10. Upon executing, **UPD** returns:

```
UPS DECLARE SUCCEEDED: Continue with another product, or quit
```

and then returns to the main menu.

1. We discourage declaring a product instance as current when it is first being declared. The reasons for this are discussed under *The current and uncurrent Executables* in section 5.4.5.

If tailoring were required, you would be informed now. To check that the declare worked, verify the existence and contents of the file `pine` in your **UPS** database. The file should contain information similar to the following:

```
Product:        pine

# This file is maintained by Unix Product Support.

# DO NOT EDIT THIS FILE DIRECTLY.



db_version:     "1.2"


Chain:
        test    SunOS+5 v3_95   4-11-97 "aheavey"
end.

Instance:       "v3_95" SunOS+5 /home/dcdsv0_lv0/aheavey/ups_play/products/SunO
S+5/pine/v3_95
                4-11-97         "aheavey"


                Authorized_Nodes: [*]
end.
```

This installation is now complete. To make this test instance of **pine** available, exit out of **UPD** and run:

`% setup -t pine`

3.3.4 Additional Steps Required for Some Products

Install Note

If a product instance comes with an install note, it will be displayed to the screen during the unwind process. For example, upon unwinding the tar file for the product **www** v2_7, the following text displays (edited for brevity):

```
UPD creating root directory /usr/products/SunOS+5/www/v2_7...
UPD unwinding tar file ...
x ups, 0 bytes, 0 tape blocks
x ups/INSTALL_NOTE, 1089 bytes, 3 tape blocks
x ups/setup.common, 326 bytes, 1 tape blocks
...text removed...
x etc, 0 bytes, 0 tape blocks
x include, 0 bytes, 0 tape blocks
******************************************
***       INSTALL_NOTE                 ***
******************************************
This is www v2_7.

It includes the following browsers:
        Name    Description
        ----    -----------
        www     Cern Line mode browser
        lynx    Full screen ASCII browser
```

```
        Mosaic  NCSA Motif X Window System browser

The Netscape liscence prevents us from distributing
Netscape as part of the www product, however if you
```
. . . *text removed* . . .
```
*******   END OF INSTALL_NOTE  ***********
****Press enter/return when done reading****
 UPD UNWIND succeeded   The temporary tarfile "/tmp/www_v2_7_SunOS+5.tar" still exists
   Do you wish to delete /tmp/www_v2_7_SunOS+5.tar(y or n)[y]:
  /tmp/www_v2_7_SunOS+5.tar deleted
  UPD UNWIND SUCCEEDED: continue with declare
```

Requirements

The UPD GET REQUIREMENTS function is used to display a list of the use and build requirements of a product. Note that product providers can choose to supply the requirements either in a `declare.dat` file (see sections 5.2.5 and 5.4.2) or in the product's product file in the `KITS` database (or other remote database if different).

This function first looks for a listing of requirements in the appropriate product file in the `KITS` database. If they are not listed there (they are often not listed there; the `KITS` product files generally point to tar files and thus are different from product files on user nodes), it looks in the product tar file for a `declare.dat` file, extracts it, and displays the requirements information from it in the standard **UPS** format.

```
UPD GET REQUIREMENTS:   * implies value required
        *1       [upd.fnal.gov] remote node name
        *2       [www] product name
        *3       [v2_7] version or chain (e.g., v1_0 or current)
         4       [SunOS+5] flavor
         5       [] remote database, if different from default

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

If a product has use or build requirements, upon execution of the UPD GET REQUIREMENTS screen, you'll see output similar to the following (this example shows use requirements only):

```
 UPD sending request to remote node...
No remote authentication
 www v2_7 has no build requirements on upd.fnal.gov
No remote authentication


UPD GET REQUIREMENTS:   * implies value required
        *1       [upd.fnal.gov] remote node name
        *2       [www] product name
        *3       [v2_7] version or chain (e.g., v1_0 or current)
         4       [SunOS+5] flavor
         5       [] remote database, if different from default

         e       execute upd getreq
         q       quit and return to main menu

           use requirement : < ximagetools current  SunOS+5,< ghostview current
 SunOS+5,< xpdf current  SunOS+5
          build requirement : none
 Enter choice :
```

Each of the use requirements listed here calls for the instance chained to current for SunOS+5, and must be setup before **www** (the main product).

Tailoring

Tailoring is the aspect of the product implementation that requires input from the product installer. If tailoring is required, the file `$PRODUCT_DIR/ups/tailor` is supplied in the format of an interactive executable (script or compiled binary). If this executable exists, then when the declaration is complete, a message displays indicating that tailoring is required.

```
UPS DECLARE SUCCEEDED: Continue with UPS TAILOR
```

You must explicitly tailor the product instance using either the UPS TAILOR screen in the **UPD** menu interface or the **UPS** command **`ups tailor`**; tailoring is not performed automatically.

For a sample tailor session for the **UPS** product, see *Tailoring UPS* under section 4.2.2.

3.4 Using the UPD/UPS Shell Interface to Install a Product

You need to use a combination of **UPD** and **UPS** shell commands to install a product from the command line. There is actually only one **UPD** command for a series of functions; the initial option that you must specify on the command line indicates which function to perform. For interactive installations, there is little advantage to typing each command in by hand, however if you want to write scripts to perform **UPD** operations, the commands can be used in these scripts.

You can get limited help on the **`upd`** command online by typing **`upd -h`**. In this section we provide more explanation on the options and give examples to clarify usage.

The **`upd`** command is of the format:

`% upd` ***initial_option*** **`[-n`** ***node*****`]`** **`[-p`** ***product*****`]`** **`[-i`** ***instance*****`]`** **`[-f`** ***flavor*****`]`** **`[-T`** ***local_tar_file*****`]`** **`[-z`** ***remote_database*****`]`** **`[-r`** ***product_root_directory*****`]`**

Some general notes apply to the command:

- The ***initial option*** determines the function to perform: list, copy, unwind, etc.
- The argument to **`-n`** is the remote distribution node.
- You can use the asterisk (**`*`**) in ***product*** if you quote it (e.g., **`-p g\*`**).
- ***instance*** here requires chain or version (e.g., **`-i v1_0`** or **`-i current`**).
- The ***flavor*** must be specified exactly (e.g., **`-f SunOS+5`**).
- ☞ Note that in the **`upd`** command the **`-z`** option refers to the *remote* database, whereas in the **UPS** commands, it refers to the *local* database.

3.4.1 Run a Product Listing (upd -l) on Distribution Node

Command Syntax

`% upd -l -n` ***node*** **`[-p`** ***product*****`]`** **`[-i`** ***instance*****`]`** **`[-f`** ***flavor*****`]`** **`[-z`** ***remote_database*****`]`**

Example

`% upd -l -n upd.fnal.gov -p g\* -i current -f SunOS+5`

lists all current instances of products on upd.fnal.gov (in `KITS`) whose names begin with *g* and which are compiled for SunOS+5; it produces the following output:

```
 UPD sending request to remote node...
No remote authentication
               Distributable Versions of gcc
 current     --> v2_7_2             Flavor: SunOS+5
               Distributable Versions of geant
 current     --> v3_21_05           Flavor: SunOS+5
               Distributable Versions of ghostview
 current     --> v4_0b              Flavor: SunOS+5
               Distributable Versions of glimpse
 current     --> v3_0a              Flavor: SunOS+5
               Distributable Versions of groff
 current     --> v1_09              Flavor: SunOS+5
               Distributable Versions of gtools
 current     --> v1_1               Flavor: SunOS+5
No remote authentication
No match.
```

3.4.2 Copy the Tar File (upd -c)

Command Syntax

% upd -c -n ***node*** **-p** ***product*** **-i** ***instance*** **[-f** ***flavor*****] -T** ***local_tar_file*** **[-z** ***remote_database*****]**

Example

% upd -c -n upd.fnal.gov -p ghostview -i v4_0b -f SunOS+5 -T /tmp/ghostview_v4_0b_SunOS+5.tar

selects the remote tar file in `KITS` that best matches the given product, instance and flavor specifications; produces the following output as it copies this tar file to the designated local area:

```
  Enquiring tar file name from remote node ...
No remote authentication
 Copying tar directory ...
No remote authentication
No remote authentication
```

You can verify that the tar file was copied to your local `/tmp` directory:

% cd /tmp

% ls -l ghostview*

```
-rw-r--r--   1 aheavey  g020     7280640 Feb  5 13:44 ghostview_v4_0b_SunOS+5.tar
```

3.4.3 Unwind the File (upd -u)

First create the parent directory to (what will be) the product root directory if it does not yet exist.

Command Syntax

% upd -u -T ***source_tar*** **-r** ***product_root_directory***

Example

% upd -u -T /tmp/ghostview_v4_0b_SunOS+5.tar -r /home/dcdsv0_lv0/\ aheavey/ups_play/products/SunOS+5/ghostview/v4_0b

creates the designated product root directory if needed, and unwinds the previously copied tar file into it; produces the following output:

```
ups_play/products/SunOS+5/ghostview/v4_0b
UPD creating root directory /home/dcdsv0_lv0/aheavey/ups_play/products/SunOS+5/g
hostview/v4_0b...
UPD unwinding tar file ...
x ups, 0 bytes, 0 tape blocks
x ups/declare.dat, 122 bytes, 1 tape blocks
x ups/setup.csh, 113 bytes, 1 tape blocks
...text removed...
x bin/wftopfa, 52 bytes, 1 tape blocks
x bin/ghostscript symbolic link to gs
x bin/gv symbolic link to ghostview
```

Remove the temporary tar file manually.

3.4.4 Declare the Product Instance (ups declare)

In order to declare the product instance to the **UPS** database, the **`ups declare`** command is required. You can do this within the **UPD** menu interface (see UPS DECLARE under section 3.3.3) or at the command line (see section 2.2.2). If tailoring is required, see *Tailoring* under section 3.3.4.

Examples using the shell interface are provided in section 2.2.2.

3.5 Using Anonymous ftp to Obtain 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 as described in section 3.3.3 (menu interface) or 3.4 (shell interface).

3.5.1 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. Standard **ftp** commands are covered in *UNIX at Fermilab* and in the man pages.

In order to use anonymous **ftp** to obtain tar files[2] from the `KITS` area, your remote node must be registered with Fermilab. If you need to do this, 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.

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

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

3.5.2 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):

```
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.
```

Get, and then type, the index file (you can't list a remote file from **ftp**):

```
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.

3.5.3 Copy the Tar File

We'll use **pine** again as an example. To copy pinev3_95SunOS+5.tar over to *$TMPDIR* (or /tmp, if the variable is not set) on your local machine, first change to the right directory and list the tar files to make sure it is there:

```
ftp> cd KITS/SunOS/pine/v3_95
250 CWD command successful.
ftp> pwd
257 "/KITS/SunOS/pine/v3_95" is current directory.
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
total 19700
```

```
drwxr-xr-x   2 kits     upd          512 Aug 20  1996 .
drwxr-xr-x   4 kits     upd          512 Mar  4 08:12 ..
-rw-r-----   1 kits     upd      10076160 Aug 20  1996 pinev3_95SunOS+5.tar
226 Transfer complete.
205 bytes received in 1.5 seconds (0.13 Kbytes/s)
```

Use the standard **ftp** **`get`** function to copy the file:

```
ftp> get
(remote-file) pinev3_95SunOS+5.tar
(local-file) /tmp/pinev3_95SunOS+5.tar
200 PORT command successful.
150 Opening ASCII mode data connection for pinev3_95SunOS+5.tar (10076160 bytes)
.
226 Transfer complete.
local: /tmp/pinev3_95SunOS+5.tar remote: pinev3_95SunOS+5.tar
10298474 bytes received in 17 seconds (6e+02 Kbytes/s)
```

Verify the transfer after exiting from **ftp**:

```
ftp> bye
221 Goodbye.
<fsui02> cd /tmp
/tmp
<fsui02> ls -l pinev3_95SunOS+5.tar
-rw-r--r--   1 aheavey  g020     10076160 Apr 16 12:09 pinev3_95SunOS+5.tar
```

Now use **UPD** to unwind the tar file into the appropriate location on your system and declare it.

3.6 Extract a File from a Remote Tar File

3.6.1 Using the UPD Menu Interface

UPD TAR CONTENTS is used in conjunction with UPD EXTRACT to pull a single file from a product tar file across the network.

List Tar File Contents

From the **UPD** main menu, enter **`4`** to arrive at the UPD TAR CONTENTS menu, shown below. This option provides a list of files in the tar file at the remote node. We use the product **www** as an example:

```
UPD TAR CONTENTS: * implies value required
        *1       [upd.fnal.gov] remote node name
        *2       [www] 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 tarcont
         q       quit and return to main menu
 Enter choice :
```

Upon executing, the system returns the list of files (edited for brevity):

```
 Enquiring tar file name from remote node ...
No remote authentication
 Enquiring list of contents...
No remote authentication
 ups/
```

```
 ups/INSTALL_NOTE
 ups/setup.common
 ups/setup.csh
 ups/setup.sh
...text removed...
 bin/arena.real
 bin/Netscape.fake
 bin/Arena symbolic link to start
 etc/
 include/
No remote authentication


 UPD TARC SUCCEEDED: continue with extract
```

Then it returns you to the main menu. Determine which file you want to extract.

Extract a File

From the main menu, enter **5** to arrive at the UPD EXTRACT menu, shown below. This function pulls a single file across the network from the product's tar file on the remote host to your local machine.

```
UPD EXTRACT: * 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
        *6       [] filename within remote tar file
        *7       [] local file specification

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

The filename within the remote tar file must match exactly the name as shown in the UPD TAR CONTENTS list. For the `local file specification`, be sure that the directory into which the file goes has been created before executing this function.

3.6.2 Using the UPD Shell Interface

Refer to section 3.4 for a description of the **upd** command.

List the Contents of the Tar File (upd -t)

Command Syntax:

% upd -t -n ***node*** **-p** ***product*** **-i** ***instance*** **[-f** ***flavor*****] [-z** ***remote_database*****]**

Example:

% upd -t -n upd -p ghostview -i current -f SunOS+5

lists the files in the tar file in `KITS` that best matches the given specifications, producing the following output:

```
 Enquiring tar file name from remote node ...
No remote authentication
 Enquiring list of contents...
No remote authentication
 ups/
 ups/declare.dat
 ups/setup.csh
 ups/setup.sh
 ups/unsetup.csh
 ups/unsetup.sh
...text removed...
 bin/ghostscript symbolic link to gs
 bin/gv symbolic link to ghostview
No remote authentication
```

Extract a File (upd -x)

Command Syntax:

% upd -x -n *node* -p *product* -i *instance* [-f *flavor*] -L *local_tar_file* -R *remote_filename* [-z *remote_database*]

The ***local_tar_file*** must be a UNIX standard file specification unless you want the file to be copied to the current working directory. The ***remote_filename*** should be specified *exactly* as returned by **upd tarcont**.

Example:

**% upd -x -n upd -p ghostview -i current -f SunOS+5 -L\
ghostsetup.csh -R ups/setup.csh**

extracts the specified file, producing the following output:

```
 Enquiring tar file name from remote node ...
No remote authentication
 Extracting the file on the remote node
No remote authentication
 Copying file upd:ups/setup.csh to ghostsetup.csh
No remote authentication
No remote authentication
```

You can verify that the file exists:

% ls -l ghostsetup.csh

```
-rwxr-xr-x   1 aheavey  g020         113 Feb  5 14:12 ghostsetup.csh
```

3.7 Proprietary Product Installation Information

For proprietary software with licensing restrictions, a dummy tar file containing the necessary application forms and instructions is put in place of the actual tar file on the distribution node. During the unwind step of product installation, a boilerplate install note displays to the screen. At the time of this writing, it contains the following text:[1]

```
******************************************
***       INSTALL_NOTE                 ***
******************************************

This is a proprietary product.  To obtain this product, you must complete the
enclosed request form and forward the completed form to the designated account.
This form is contained in a file called std.form that was placed in the
directory in which you unwound the product.  Upon receipt and approval of your
request, information will be sent to you on how to obtain the requested
proprietary software.


Should you have any further questions, please contact the consulting office,
Monday-Friday 8am-5pm, at 708-840-2345.

*******   END OF INSTALL_NOTE  ***********
```

1. The dummy tar file for some proprietary products may be out of date; use the information presented here if there is a question.

It refers you to `std.form` in the product root directory, which is the following file:

```
                        FERMILAB COMPUTING DIVISION

                          UNIX Proprietary Product
                                Request Form


To request UNIX proprietary  software,  complete  the  following  form,
entirely, and return it to compdiv@fnal.gov.  To request updated versions
of UNIX proprietary software, complete the items beginning with an
asterisk. See notes at the end of this form regarding proprietary software.
You will be given instruction on how to retrieve the software when you are
authorized to receive it.

*NAME:

*E-MAIL ADDRESS:

*PHONE NUMBER:

*PRODUCT NAME:

*FLAVOR DESIRED:
(IRIX, DIGITAL/UNIX, SunOS, AIX etc.)

MACHINE TYPE, MODEL NAME, AND NUMBER:

NODE NAME:
(fully-qualified internet name)

OWNER OF COMPUTER:
(e.g., Fermilab, university)

*DATE:

COMMENTS:


===========================
Our contract with the vendor determines how  and where the  products can be
made available.  In some cases,  we have a  "site license" which allows the
product to be used on any machine on site.  In other cases, we have a "site
license" that applies only to Fermilab-owned machines.  In other cases,  we
have  purchased a specific  number of  licenses,  one of which  may be made
available  to you  depending on the  arrangement your  group/experiment has
made with the Computing Division.

Licensed (proprietary) software is obtained by the laboratory for your use.
Most of this  software carries with it explicit  (sometimes implicit) legal
restrictions as to who,  what,  where,  when,  and how the  software may be
used.  When you  obtain such  software from  the Computing  Division for a
particular platform, that software should be used on that platform only and
not copied.  It should be assumed that you may not make or take a copy away
with you unless you have permission to do so.   Most software now has legal
protection under patent or copyright law.
```

After filling it out and sending it to *compdiv@fnal.gov*, you will receive a reply of the form:

```
You have requested access to the following product:
        Product: product
        Version: vx_y
        Flavor: flavor
You may access this file via anonymous ftp at the machine ftp.fnal.gov.
It is found at /path/tarfile

In order to access this file, after logging into ftp, you will have to
issue the following commands at the prompt:
        quote site group group_name
        quote site gpass password

If you have any problems, please send mail to compdiv@fnal.gov.
```

You may be given a time limit for copying the real tar file over, after which your access will be cut off. Once you have copied the tar file, use the **UPD** menu (or **UPS** commands) to complete the installation.

Chapter 4: Making your system FUE-Compliant: How to Bootstrap FUE

The recommended method for installing Full FUE on a system is to install first **UPS**, then **UPD**, and finally to use **UPD** to install the FUE packages. This chapter discusses the preparation necessary for the installation of these products and then walks you through the installation procedures for each of them.

4.1 Getting Ready

As we mentioned above, this chapter describes the installation of a Full FUE environment, which works properly and predictably. We recommend that you select at least this level of FUE conformance. If you are not interested in FUE at this level, we recommend that you limit yourself to the products in FermiTools, which are accessible to the general public and don't require **UPS** or any level of FUE. As an aside, **UPS** is one of the products in FermiTools.

If you install FUE at a level lower than Full FUE, you may run into problems with certain products. These problems and their solutions are not documented in this manual, or anywhere else, in a complete fashion.

If you plan to install **UPD** you need to have **NFS** on your system so that the commands **`rshell`** and **`rcp`** work. Also, the node must have an internet address.

If you need to have your (on-site) system configured properly prior to beginning the installation processs, contact your Computing Division liaison for assistance.

Before continuing with this section, make sure you know whether **UPS** is going to be used system-wide on your node or cluster, or used by a single individual. You will need to make decisions about directories, login ids and permissions accordingly.

4.1.1 Organize Your Products Area

For **UPS** to work reliably, make sure you keep the same name for each product on your local system as it has on the distribution system. This facilitates support and upgrades, besides making products as easily recognizable to your users as possible. Also, many setup scripts expect the name to be as it appears on the distribution node. This is especially important for the **UPS** product!

UPS is not dependent upon where products are installed; they can even be installed in completely different areas from each other. However, your system will be easier to maintain if you build some order into your installation. You want a directory structure that neatly allows multiple versions of products to exist without confusion. First, a few points to consider:

- The product's install directory (root directory) in your local **UPS** database should reflect its version.
- You may or may not want to keep any extra labelling information that describes the compile options used. In case you ever want to trace back exactly which instance you copied over, this is useful information, but it may be cumbersome in your local **UPS** database.

- If you're working on a single node that is not expected to become part of a cluster, there is no advantage to including the ***OS_type+release*** directory level.

If you have a cluster with separate product servers for each flavor, organizing your products in this way keeps them grouped cleanly:

/your_system_directories*/`products`*/OS_type+release/product/version+extra

for example,

`/ups_dirs/products/IRIX+5/ghostview/v2_0+debug`

Notice that the flavor is split into two pieces, so that the operating system and OS release go together (e.g., `IRIX+5`), and the compiler options, if you decide to retain them, get lumped with the version.

On the other hand, if a single product server is used for all flavors, an organization like the following that moves the OS and release down to the end of the hierarchy is cleaner:

/your_system_directories*/`products`*/product/version+extra/OS_type+release

for example,

`/ups_dirs/products/ghostview/v2_0+debug/IRIX+5`

4.1.2 Determine Install Accounts and File System Semantics

About Install Accounts

The local system manager and product installer(s) must determine who needs to be able to install products on their system. They must set the various directory permissions and ownerships appropriately. Making this decision now will save a lot of headaches later on.

There are two approaches:

- Designate a single user account to perform all product installs, and give appropriate access to this account only.
- Allow multiple users to perform product installs, and either set up a common account to which all the appropriate users know the password, or allow installs from individual accounts.

Each of these approaches has advantages and disadvantages.

Single Install Account	Multiple Install Accounts
Setting up the file ownership and protections is much more straightforward. The files just need to be writable by this owner.	This approach works best when all installer accounts belong to the same group. This allows you to track the installer for each product instance.
If several people use this account to install products, you lose the ability to track the individual responsible for each installation.	All installers have write access to the files and directories in question, so more care must be taken.

About File System Semantics

An important issue to understand at the outset is file system semantics. Individual file systems can be mounted either with Berkeley or System V semantics.

- Under Berkeley semantics, the group ownership of a newly created file is inherited from that file's parent directory. If you choose to have multiple install accounts but for some reason they cannot share a group, this is more straightforward.
- With System V semantics, the group ownership of a newly created file is the creator's current group id. If you choose to have multiple install accounts with a shared group, all files have the same group, which facilitates management.

How to Implement your Selected Semantics

IRIX, SunOS, AIX and OSF1 support Berkeley file semantics at the directory level by setting the `setgid` bit on the directory in question. For each operating system, the command:

`% `**`chmod g+s`** *directory*

or an OS-dependent variant of it, will work (check the man pages, especially for IRIX).

At the file system level, there is significantly greater variance, as follows:

IRIX	Default is System V; to get Berkeley set the `/etc/fstab grpid` option to `enable`. This is available for all file system types supported.
SunOS	Default is System V; to get Berkeley set the `/etc/vfstab grpid` option to `enable`. This is available for NFS file systems only.
AIX	Default is System V; no option available to change, **`chown`** and **`chgrp`** (by mortals!) is not permitted (`/etc/filesystems` is the AIX `/etc/fstab` equivalent).
OSF1	Default is Berkeley; to get System V set `/etc/fstab nogrpid` option to `disable`. This is available for UFS and NFS only. AdvFS apparently is Berkeley semantics only.

You will probably need to reference the man pages for the **`mount`** command and for the file (e.g.,. `fstab` or `vfstab`) to find out how to set the option.

Regardless of the semantics chosen, perform the entire installation from the *root* account to ensure that you have all the necessary privileges. We merely recommend this for **UPS** and **UPD**, however it is mandatory for installation of the FUE packages.

Be aware that at the end of the FUE installations, you will need to make ownership changes for the areas into which future product installers will need to write. We will point these out as we come to them.

4.1.3 Create Required Groups, Accounts and Directories

First create the necessary groups, accounts and directories.

The UIDs and GIDs (user and group ID numbers) used in this documentation are appropriate for all machines in the fnal.gov domain, since we have established a site-wide UID-GID database. If your machine is not part of this domain, select appropriate values for your site.

Groups

You need the following group records in the file `/etc/group`:

- `products::4525:`
- `local::4526:`

Accounts

Use the following OS-dependent commands to create the accounts (see the man pages for details):

IRIX	`cpeople`
AIX	`smit` (under Security & Users/Users/Add a User)
OSF1	`adduser`
SunOS	`useradd`

The required accounts are *products* and *local* with the following UIDs (first number) and GIDs (second number):

- `products 1342 4525`
- `local    1532 4526`

For example, on a SunOS system, you would create the accounts as follows:

```
% useradd -c "products account" -d /usr/products -e ""  -g 4525 -u
  1342 products
```

```
% useradd -c "local account" -d /usr/products -e ""  -g 4526 -u
  1532 local
```

Notice that we've made `/usr/products` the home directory for both accounts.

Directories

Some of the directories listed below require specific ownership and write permissions. Therefore, after you create them you will need to change permissions since you are logged on as *root*. The directories you need are:

- `/usr/local`
- `/usr/local/bin`
- `/usr/local/etc` (owner *products*, writable by owner)
- `/usr/local/products` (owner *products*, writable by owner)
- `/usr/products`
- `/usr/products/ups_database/declared`[1] (owner *root*, group *products*, writable by group)

After you install **UPS**, **UPD** and the FUE packages, make sure that all the additional directories to which future product install accounts will need to write are given the appropriate permissions.

To set permissions for owner and group in the same step, you can use the command:

% chown ***owner*:*group* *directory***

For example:

```
% chown root:products declared
```

If you want to create any private databases, see Appendix B.

1. Creating `ups_database` and `ups_database/declared` is optional. The `ups_install` script will create these directories for you. See section 4.2.2.

4.1.4 Select Instance of UPS to Install

☞ Note that starting with v3_7, **UPS** is equipped with a convenient install script called `ups_install` which is packaged in the `ups` directory under the product root directory. We recommend, therefore, that you choose an instance of version v3_7 or later.

If you don't have access to **UPS** and **UPD** on another system, use anonymous **ftp** to choose an instance (see section 3.5).

If you do have access to a system that is already equipped with **UPS** and **UPD**, and that has access to the distribution node that you are using, you can use it to determine which instance of **UPS** to install. Use the **UPD** menu, or run the **`upd -l`** command:

% **upd -l -n** ***node*** **[-i** ***instance*****] [-f** ***flavor*****] [-z** ***database*****] [-p** ***product*****]**

As an example, the command:

% **upd -l -f SunOS+5 -p ups**

will return information for the instances of **UPS** for SunOS+5 on the upd distribution node (which is the default), in the format:

```
test        --> v3_8              Flavor: SunOS+5
current     --> v3_4b             Flavor: SunOS+5
no_chain:       v3_7              Flavor: SunOS+5
no_chain:       v3_6              Flavor: SunOS+5
```

4.2 Installing UPS

Before beginning the installation, create a product root directory according to the version of **UPS** that you have chosen. Assuming the product area is `/usr/products`, the root directory may be:

`/usr/products/`*`flavor`*`/ups/`*`version`* (owner *products*; writeable by owner)

If you're installing onto a single node that is not part of a cluster, you can leave off the level associated with flavor in the directory hierarchy:

`/usr/products/ups/`*`version`* (owner *products*; writeable by owner)

Throughout this section we provide sample user sessions in which we use the products area `/usr/products`, the flavor SunOS+5 and the **UPS** version v3_8c. Our product root directory for **UPS** is:

`/usr/products/SunOS+5/ups/v3_8c`

4.2.1 Retrieve and Unwind the UPS tar file

Use anonymous **ftp** to copy over the appropriate instance of **UPS** from the `KITS` area into a temporary area, e.g., `/usr/tmp`:

% **cd /usr/tmp**

% **ftp ftp.fnal.gov**

Login as user *anonymous*, use your own full login id as your password (e.g., *freddy@fnal.gov*). Then change to the appropriate directory and retrieve the file:

> **cd /KITS/SunOS/ups/v3_8c**

> get upsv3_8cSunOS+5.tar

Quit out of **ftp**[1]. Change to the product root directory for **UPS**, and use **tar** to unwind the tar file in this directory. Then remove the old tar file.

% cd /usr/products/SunOS+5/ups/v3_8c

% tar -xvf /usr/tmp/upsv3_8cSunOS+5.tar

% rm /usr/tmp/upsv3_8cSunOS+5.tar

4.2.2 Install, Declare, Tailor and Setup UPS

As mentioned above, recent versions of **UPS** are equipped with a convenient install script called `ups_install` which is packaged in the `ups` directory.

The README File

☞ Read section A of the `README` before running `ups_install` in case any changes have been made to the installation procedure. We reprint it here for reference (**UPS** version v3_8c):

```
SECTION A

      Installing Ups on a System

It is assumed, if ups exists on the system, that it has been set up
and the the PRODUCTS environment variable has been defined.  If this
is not the case ups_install will define the PRODUCTS environment
variable[2] and create the ups database there and declare ups in it.
Ups_install will also ensure that the /usr/local/etc/setups.* scripts
exist if permitted to write in /usr/local/etc.  If necessary,
ups_install will attempt to create /usr/local/etc.

If a current version of ups exists, ups_install will leave the
installed and tailored version of ups installed as "new" unless
the "current" operand is used.

% /usr/products/ups/X.X/ups/ups_install or
% /usr/products/ups/X.X/ups/ups_install current # To force the "current"
# declaration

The setups.* scripts provide the mechanism to select
the appropriate ups for the platform upon which the script
is run.  However, these scripts are meant to be *sourced*  `
at login and shell start times.

To automatically provide ups services for users it is necessary
to include a

     source /usr/local/etc/setups.csh
                   or
              . /usr/local/etc/setups.sh
in the appropriate files which will also be sourced at the
appropriate times.
```

1. Before quitting, you can also get the tar file for **UPD** to save yourself that step later on.
2. This can be confusing. The `ups_install` script defines environment variables for internal use only. *$PRODUCTS* gets defined afterwards when you source the `set-ups.$UPS_SHELL` file.

```
These files are often refered to as .cshrc, .login, .profile,
and .shrc files
```

Description of ups_install

For your reference, we include the description from the script of the tasks it performs.

Note that all environment variables that get set in this script are for internal use by the script only. Since the script is not sourced, they do not propagate back up a process level. The variables get defined for general use in a subsequent procedure.

```
# Ups_install
#
# As the name implies this script attempts to perform the necessary
# gyrations to install a version of ups.  It will attempt to do
# this whether there is a version of ups available on the platform,
# even from a platform with another OS.
#
# SYNTAX
#
# $UPS_DIR/ups/ups_install <current>
#
# ARGUMENTS
#
#“current”
if you wish this version installed as
#the “current” version.  This is the
#default if there is no “current” ups
#available in the “normal” databases
#with the same flavor(determined from
#$UPS_DIR/ups/Version)
#
# DESCRIPTION
#
# Ups_install performs the following tasks:
#
#Determine the ups home directory from either the current
#working directory or the contents of $0 if executed
#with a full pathname.
#
#If the PRODUCTS environment variable is not defined
#set PRODUCTS(and the ups database name) to
#/usr/products/ups_database/declared.
#
#Create, if necessary, the ups database.
#
#Set up this version of ups(using ups_init and setup -r)
#If platform OS is sufficiently similar to target.
#Otherwise, use ups from the resident platform, if
#available.
#
#Thereafter, use ups declare and ups tailor to declare and
#tailor this version of ups as “new”.  If desired, or
#defaulted, also declare this version “current”.
```

Running the ups_install Script

Our sample install procedure shows `ups_install` called with the optional argument *current*. This new instance of **UPS** will therefore get chained automatically to current as well as to new. This argument is not strictly necessary for an initial installation, since chaining to current is the default in this case. Note that in a subsequent installation this would override any previous current instance.

Start the installation script[1]:

```
% cd /usr/products/SunOS+5/ups/v3_8c/ups
% ups_install current
```

The script performs several actions, as described above. The first thing that you will see on the screen is the tailor session. You will need to complete it as shown below.

Tailoring UPS

In a cluster, **UPS** needs to be tailored once for every extended flavor of which one or more nodes will be used to declare products. Actually, it's once per instance installed on the cluster; in most cases this is the same. When an instance is tailored, a `default.dat` file is created that can be accessed by other nodes of the same flavor. If you select node-specific tailoring, a `*node*.dat` file, where *`node`* is the node name, will get created.

Several aspects of the implementation are included in the **UPS** tailor executable. You need to be prepared to provide the appropriate answers. Be aware that it can be retailored later if your needs change.

- Set the current node as a list server. Respond `y` or `n`.

  Running **UPS** as a list server allows users on remote machines to list the product database without having an account. Normally this is neither necessary nor desirable. For many of the experiment machines on site, however, the Computing Division specifically requests that **UPS** be implemented this way. Contact your Computing Division liaison if you have questions.

- Select the directory in which to put man pages for all **UPS** products that provide them. A default is provided. Man pages get copied to appropriate subdirectories under this one when a product is declared as current. Two caveats:

  a) Depending on your file system semantics, if a product is declared current under the *root* login id, you may need to reset write permissions on the man page areas for your install accounts.

  b) If you are planning to run **UPS** by itself without FUE, include this directory in *$MANPATH*.

- Select the directory in which to put Info files for all **UPS** products that provide them. A default is provided. Info files get copied to appropriate subdirectories under this one when a product is declared as current. You may need to reset write permissions on the Info areas for your install accounts

- Enable automatic start/stop capability. Respond `y` or `n`.

  Some products need to run as background daemon or server processes. You have the option of configuring **UPS** to automatically start and stop these products at system startup and shutdown. If you choose *not* to enable this capability in **UPS**, but you still want to run some of these products, then you will need to edit your standard OS-specific startup files to control the automatic start/stop. The advantage to using **UPS** is that when a new instance of a product is declared as current, this instance automatically takes the place of the older instance when the system is restarted; no files need to be updated.

  Note that the **UPS** product itself can be run as a daemon process. This allows **UPS** to run as a list server.

---

1. Remember, if "dot" is not in your *PATH*, preface the **`ups_install`** command with **`./`**. You could just as well supply the full pathname for the `ups_install` script and run it from any directory.

To implement the start/stop capability in **UPS**, you will need to respond `y` and then complete some additional configuration after the `ups_install` script finishes, as described in Appendix C. This needs to be done only once per cluster.

The following is a sample transcript of a tailor session for **UPS** v3_8c on a SunOS+5 machine.

```
 The default action is to tailor ups for
flavor/cluster wide action, not node specific actions

 If you wish to tailor ups for a specific node,
   enter the unqualified node name here: webtest


 TAILORING FOR NODE: webtest

 List server node tailoring supported

 Do you wish to continue(y/n)[n]?y


TAILORING OF MAN PAGES

Please enter the directory in which the man directores are to be created.
    `./catman' and `./man' directories will be created to hold the
    formatted and unformatted man pages respectively.
    Appropriate sectional subdirectories will be created
    as needed.
    A default has been chosen for you. To accept this press
    enter/return key.

    If you wish to use another please
    consider a directory already in MANPATH
    These are: /usr/openwin/share/man:/usr/man

Remember to leave off the final `/catman' and  `/man'.

Top level man page directory (must begin with a /)?

 Mandir[/usr/products]:
Remember to leave off the final `/catman' and  `/man'.

Top level man page directory (must begin with a /)?

 Mandir[/usr/products]:


TAILORING OF INFO FILES

Formatted Info files should be copied into which directory?

   A default has been chosen for you.  To accept this press
   enter/return key.  If you wish another, it must begin with a /

 Infodir[/usr/products/Info]:

TAILORING OF A LIST SERVER NODE

ups has a list server which allows a remote node to querry theups database on this node.
Only one node should be tailored as a list server  in a cluster

Do you want this node to be the list server (y/n)[n] ?n


TAILORING FOR STARTING/STOPPING PRODUCTS
```

```
Ups supports starting products at boottime and stopping them at
shutdown. Special files need to be created if your system
contains products that need to be started/stopped. These files
usually need only be created once per cluster, not per node.
Do you want to create the start/stop files (y/n)[n] ? y

You have chosen the following values:

   ManHome: /usr/products
   InfoHome: /usr/products/Info
   ServerNode: no
   Start/Stop: yes

Do you wish to continue(y/n)[n] ? y


UPS INITIALIZATION FOR STARTING/STOPPING PRODUCTS

Ups is must be initialized before products can be
started.  At Fermilab this action is usually performed
 by "/usr/local/etc/setups.sh"

    If this is not satisfactory, please enter the
    fully qualified name of the initialization file
    The default is [/usr/local/etc/setups.sh]

ups initialization file name :

ups has now been tailored to stop/start. You will need to make it current from a privileged
account.


ups: Information - ups tailoring now complete
ups: Information - instance = v3_8c exists
                   but I'll gladly declare chain(s)

ups: Warning - A start file for ups v3_8c SunOS+5 exists. Please inform
            your system manager that this product needs to be
            added to the list of products to start at boottime.

ups: Warning - A stop file for ups v3_8c SunOS+5 exists. Please inform
              your system manager that this product needs to be
              added to the list of products to stop at shutdown.

 +=======================================================+
|             I N S T A L L   S T A T U S


|
|             ups version:     v3_8c
|                 flavor:      SunOS+5
|                 chain:       current
|
|                Status:      Successful.
|
+=======================================================+
```

Source the setups Executable

Once the `ups_install` script finishes, source the `setups.$UPS_SHELL` file (read: *set UPS*) appropriate for your shell family:

% source /usr/local/etc/setups.csh

or

$. /usr/local/etc/setups.sh

This file sets the environment variables *$PRODUCTS* and *$UPS_DIR*, sources `ups_init.$UPS_SHELL` (which creates the alias for the **`setup`** command and defines *$UPS_SHELL*), and sets up **UPS**.

Verify that you now have *$PRODUCTS*, *$UPS_DIR*, and *$UPS_SHELL* defined, and that you can run **UPS** commands (e.g., try running **`ups list -a ups`**).

☞ Note that **UPS** was designed to be able to run independently of the FUE packages, so if you want to run **UPS** without **UPD** or FUE, your installation is now complete.

4.3 Installing UPD

☞ Installing **UPD** is optional, but is strongly recommended since it will greatly facilitate the installation of all the FUE packages. **UPS** commands can be used instead, but they are more cumbersome.

UPD has undergone a few changes since late 1996 to provide better defaults and clearer options, so use version v2_5 or later, as available. Throughout this section we provide sample user sessions in which we assume a flavor of SunOS+5 and **UPD** version v2_8d.

4.3.1 Select Instance of UPD to Install

Use anonymous **ftp** or the **`upd list`** command (if you have access to **UPD** on another machine) to find an appropriate version of **UPD** for your system's flavor. The **`upd list`** command could be used as in this example for the flavor SunOS+5:

`%` **`upd -l -f SunOS+5 -p upd`**

Before beginning the installation, create the product root directory according to the version of **UPD** that you have chosen, for example:

`/usr/products/SunOS+5/upd/v2_8d` (owner *products*; writeable by owner)

If you're installing onto a single node that is not part of a cluster, you can leave off the level associated with flavor in the directory hierarchy.

4.3.2 Retrieve and Unwind the UPD tar file

If you haven't already retrieved the tar file, use anonymous **ftp** to copy over the appropriate instance of **UPD** from the `KITS` area to a temporary area.

`%` **`cd /usr/tmp`**

`%` **`ftp ftp.fnal.gov`**

Login as user *anonymous*, use your userid as your password. Then find and retrieve the tar file.

`> cd /KITS/SunOS/upd/v2_8d`

`> get updv2_8dSunOS+5.tar`

Quit out of **ftp** and change to the product root directory for **UPD**.

`%` **`cd /usr/products/SunOS+5/upd/v2_8d`**

Use **`tar`** to unwind the tar file (then remove the old tar file).

```
% tar -xvf /usr/tmp/updv2_8dSunOS+5.tar
% rm /usr/tmp/updv2_8dSunOS+5.tar
```

4.3.3 Declare UPD to the UPS Database

We recommend declaring a product to the database first, and declaring it as current in a separate step. This is discussed in *The current and uncurrent Executables* under section 5.4.5.

First declare this instance of **UPD** to the **UPS** database using a command of the form:

```
% ups declare -f SunOS+5 -r /usr/products/SunOS+5/upd/v2_8d upd
  v2_8d
```

The **`-r`** option is required. Even if you have chosen to eliminate the flavor level in the directory hierarchy, you may want to include the **`-f`** option here; otherwise **UPD** is declared with a flavor of NULL. If you have multiple databases, also include the **`-z`** ***database*** option (place it before **`upd v2_8d`** on the command line).

Secondly, declare it as current:

```
% ups declare -c upd v2_8d
```

Now if you run **`setup`** on **UPD**, it is available to you.

4.4 Installing the FUE Packages

The FUE packages are maintained as **UPS** products, and therefore can be installed in the same way as other **UPS** products. The **UPD** menu interface is the most straightforward method. Refer to section 3.3 for instructions on using this interface to install the packages.

4.4.1 The Basic FUE Packages

☞ FUE currently consists of three basic packages: **funkern** (pronounced *foo-kern*), **futil**, and **systools**. Note that the list of programs contained in each of these packages is subject to change.

funkern

funkern provides the core portion of the Fermi files, as well as programs called from the Fermi files. The current version for SunOS+5 as of this writing, v5_0, includes:

funame	**logage**	**logshell**	**grpmembers**	**addpath**
dropit	**logdata**	**loguid**	**grpname**	**delpath**
logdir	**loggid**	**loguser**	**grpnumber**	

futil

futil contains printing utilities, compression programs, the **less** pager program, **Info**, and other utilities. The current version for SunOS+5 as of this writing, v5_2, includes:

Info	**less.help**	**flpk**	**zmore**	**gzip**	**telephone**
obtain	**lesskey**	**findcpp**	**zgrep**	**gunzip**	**tele**
vt100	**a2ps**	**getdate**	**znew**	**zdiff**	**stock**
getlock	**psnup**	**cwi**	**zforce**	**zcat**	**lsof**
namefix	**flpr**	**zap**	**gzexe**	**subm**	
less	**flpq**	**fpasswd**	**zcmp**	**subm.cron**	

systools

The current version for SunOS+5 of **systools** as of this writing, v4_3, contains the majority of the Fermi files...

`fermi.cshrc`	`fermi.shrc`	`setups.csh`	`stdXdefaults.IRIX`
`fermi.login`	`setpath.csh`	`setups.sh`	`stdXdefaults.AIX`
`fermi.profile`	`setpath.sh`	`stdshrc`	`stdlogin`
`stdmailrc`	`stdprofile`	`stdcshrc`	

... as well as system administration utilities:

genpasswd	**dellocal**	**addlocal**	**chmod**	**shells**	**echo**	**clearaccounts**
finduser	**addmember**	**sys.env**	**moduser**	**renice**	**kill**	**resetuser**
delmember	**addproduct**	**greeting**	**chguser**	**vipw**	**rm**	**delproduct**
deldata	**adduser**	**functions**	**disuser**	**shell**	**vigr**	**modproduct**

fulib

There is also a **fulib** package which provides a library of C-callable functions which are essentially function versions of the utilities in **funkern**. You don't need **fulib** to be FUE-compliant, it is only required if you need to rebuild the software (e.g., you are trying to install FUE on an OS for which we have not already built the products).

4.4.2 Full FUE

Full FUE is the minimum level of FUE compliance that we recommend. In addition to **UPS**, **UPD** and the three FUE packages, it includes **shells**. **shells** contains alternative login shells and UNIX scripting lanuages (**tcsh**, **bash**, **perl**, **gawk**).

Once **UPS** and **UPD** are installed, install the packages:

1) **funkern**

2) **futil**

3) **systools**

4) **shells**

Some of the FUE packages come with an `INSTALL_NOTE` indicating further steps you need to take to complete the installation (e.g., copying files to different locations). It is displayed to the screen as you unwind the tar file if you use the **UPD** menu interface.

You need to copy the Fermi files to the home directories of the logon ids that were created prior to the installation of FUE on the system, namely *root*, *products* and *local*[1]. Perform the following file copies for *root*:

```
% cp /usr/local/etc/stdprofile /.profile
% cp /usr/local/etc/fermi.shrc /.shrc
% cp /usr/local/etc/stdlogin /.login
% cp /usr/local/etc/fermi.cshrc /.cshrc
```

... and for *products*:

```
% cp /usr/local/etc/stdprofile /usr/products/.profile
% cp /usr/local/etc/fermi.shrc /usr/products/.shrc
% cp /usr/local/etc/stdlogin /usr/products/.login
% cp /usr/local/etc/fermi.cshrc /usr/products/.cshrc
```

Finally, perform any steps indicated in the `INSTALL_NOTE`s to complete the installation of Full FUE.

4.4.3 FUE Plus

FUE Plus consists of Full FUE plus some products related to user communication like email and **www** access. Many of the products in FUE Plus are graphical, therefore installation of **X11** and **Motif** releases from the operating system vendor is required. The additional products that you need to install are:

exmh with use requirements:

- **mh** (with use requirement **mailtools)**
- **glimpse**
- **www** (with use requirements **ghostview**, **ximagetools**)
- **mimetools**
- **expect**
- **ispell**
- **popclient**

1. *products* and *local* share a home directory, so this only needs to be done for one of them. Note that you will be able to create new logon ids automatically equipped with FUE using programs provided in **systools**.

xpdf

tk

tcl

There is no a priori order of installation of these products; you just need to remember that use requirements must be installed, declared and setup before the parent product can function properly.

Some of these packages may come with an `INSTALL_NOTE` indicating further steps you need to take to complete the installation. It is displayed to the screen as you unwind the tar file.

Note: During the product installations you will see warning messages about missing dependencies. You can safely ignore these messages while you are installing a list of mutually-dependent products.

Appendix A. Further Information on FUE

A.1 FUE Conformance Levels

The Fermilab Computing Division has defined a set of FUE conformance levels. These definitions benefit both users and service providers. Once they know the conformance level of a system, users can determine what is necessary for a particular product to work, and service providers know which products already exist on a system.

We list the levels in order of increasing conformance:

1) **None**: The system cannot be made FUE-compliant because it doesn't meet at least one of the minimum requirements: 1) a supported architecture or OS release, and 2) sufficient local system management support.

2) **Minimal**: The system does meet the minimum requirements listed under **None**, but no FUE software has been installed. At this level, the Computing Division may provide OS support. FUE can be installed at any time.

3) **Core FUE**: The system has enough FUE software installed to allow access to products in **UPS** format. This includes **funkern** and **UPS**. **funkern** provides the core portion of the Fermi files, as well as programs called from the Fermi files. **UPS** provides the **setup** program, allowing users to gain access to other products.

4) **Full FUE**: This level is regarded as fully FUE compliant. In addition to the components of **Core FUE**, it includes **UPD**, **futil**, **systools**, and **shells**. **UPD** provides access to other **ups** products. **futil** contains printing utilities, compression programs, the **less** pager program, **Info**, and other utilities. The **systools** product contains the majority of the Fermi files as well as system administration utilities. The **shells** product contains alternative login shells and UNIX scripting languages (**tcsh**, **bash**, **perl**, **gawk**).

5) **FUE plus**: The system has the **Full FUE** installed as well as products related to user communication (e.g., e-mail and Web access). The specific products required are: **www**, **ghostview**, **ximagetools**, and the mail utilities **exmh**, **mh**, **mailtools**, and **tk**. In addition, **FUE plus** requires installation of **X11** and **Motif** releases from the Operating System vendor, since many of the products specified in **FUE plus** are graphical in nature.

A.2 The FUE Working Group

A.2.1 Accomplishments

- the establishment of the *UNIX Product Distribution* area (`KITS`)
- development of a process for distributing proprietary products
- the ongoing design and development of the **UPS** (*UNIX Product Support*) family (**UPS**, **UPD**, **UPP** (*UNIX Product Push*))

• access to `KITS` area via anonymous **ftp**

• the establishment of a certification process for new operating system and compiler releases

• coordination of multiple shell support

• the establishment of recommendations for UNIX code management

A.2.2 Ongoing Activities

UNIX Support Methodology	Continue the traditional role of **UPS**, **UPD**, `KITS`, and operating system certification review as it affects users of the systems and the applications they use.
Public Domain Software	Discuss, propose, determine and review needs, priorities, support levels, and resource loads of public domain UNIX software that is not restricted to one application or project. Be responsible for a “policy on support of public domain software”.
Software Development and Distribution Mechanisms	Provide software developed by the Computing Division to the Fermilab user community in a fashion that makes the software easy to obtain and use.
General Software Development Methodology and Standards	Develop standard, consistent, and developer-friendly code management and maintainance standards and practices to improve the quality of the software products and to provide for meaningful Quality Assurance. Enforce adherence to the methodology and standards within the Computing Division in order to promote successful introduction of software to the user community.
Identification of New Product/Application Needs	Discuss, propose, organize and introduce new areas of relevant interest (e.g., evaluation of CASE tools, new programming languages).

Appendix B. Creating a UPS Database for Individual Use

You can create a **UPS** database for individual use in addition to or instead of a common one. Full FUE or FUE Plus must be installed on your system.

First, create the necessary directories. The precise structure is up to you, but we recommend a database and products area like the following:

`~`*`username`*`/ups_area/ups_database/declared`

`~`*`username`*`/ups_area/products`

In your `.login` or `.profile` (or wherever you set your environment variables at login time) create or modify your *PRODUCTS* variable according to one of the following options:

1) Add your new database to the *PRODUCTS* variable, as the first value.

C shell family	**`setenv PRODUCTS "$HOME/ups_area/ups_database/declared $PRODUCTS"`**
Bourne shell family	**`PRODUCTS="$HOME/ups_area/ups_database/declared $PRODUCTS"; export PRODUCTS`**

This option pre-pends your area to the existing *$PRODUCTS* database. The advantage to this option is that you will still have access to the products installed on your system for common use, because you include the common *$PRODUCTS* area as well as your individual area. Products in your individual area will be accessed in preference to copies of the same products in the other database(s).

You may need to use the **`-z`** option on **UPS** commands (or fill in the "ups database" field in the **UPD** menu) to tell the system which database you want to use when declaring/modifying products. Or you can set the variable *DPRODUCTS* each time to the one you plan to use.

2) Redefine *PRODUCTS* to point only to your private **UPS** database.

C shell family	**`setenv PRODUCTS "$HOME/ups_area/ups_database/declared"`**
Bourne shell family	**`PRODUCTS="$HOME/ups_area/ups_database/declared"; export PRODUCTS`**

In this case you will not need to use the **`-z`** option with **UPS** commands, because you only have one directory in the *$PRODUCTS* path. But, if you set it this way you lose access to all the products installed in the common area; you can only run **`setup`** on products in your database.

Appendix C. Enabling Automatic Start and Stop of UPS Products

This appendix is intended to help you prepare your system for implementation of daemon processes controlled by **UPS**. The first section discusses configuring **UPS** to support automatic start and stop of processes. The procedures described in the subsequent sections discuss further steps that you need to take in order to install other **UPS** products as daemons.

Note that section 2.3.2 describes the steps required to implement an individual product as a daemon process after your system is fully configured to support automatic start/stop under **UPS**. If you need to run **UPS** as a list server (sometimes required for experiment machines; your Computing Division liaison will be informed if this is necessary), you will need to implement **UPS** itself as a daemon process.

C.1 Automatic Start/Stop of Products: Introduction

UPS can be implemented to allow products to run as background (or daemon) processes. Daemons require automatic start up at boot time and termination at system shutdown.

C.1.1 The Role Played by the UPS tailor Script

In order to exploit this feature of **UPS**, you must tailor it accordingly. The `tailor` script for **UPS** is run automatically for you by the `ups_install` script which we discussed in section 4.2.2. The portion of the `tailor` script that addresses automatic start and stop of products is shown below:

```
TAILORING FOR STARTING/STOPPING PRODUCTS

Ups supports starting products at boottime and stopping them at
shutdown. Special files need to be created if your system
contains products that need to be started/stopped. These files
usually need only be created once per cluster, not per node.
Do you want to create the start/stop files (y/n)[n] ?
```

A reply of `y` is required to enable automatic start/stop of products. Let's clarify what actually happens here. The script asks if you want to create "the start/stop files". The files it refers to are `ups_startup.`*`$UPS_SHELL`* and `ups_shutdown.`*`$UPS_SHELL`*. If you answer `y`, they get created dynamically in the `ups` directory for **UPS**. These files run daemon process control files, described in section C.3.

Confusion may arise because there are files called `start` and `stop` which come packed in the tar file in the `ups` subdirectory. In fact all **UPS** products that are appropriate to run a daemons (should) come equipped with `start` and `stop` files. They are necessary for implementing the individual product as a daemon. Section C.4 shows the relationship between the different files involved in the automatic startup process.

C.1.2 The Role Played by the UPS current Script

Prior to making this instance of **UPS** current, two new directories are needed for the system to access at boot time and shutdown. These directories must be called `startup` and `shutdown`, and be located in the same directory that contains `declared`. For example, if your **UPS** database is `/usr/products/ups_database/declared`, then `startup` and `shutdown` must also be in the directory `/usr/products/ups_database`. The `current` file for **UPS** creates these directories for you if they do not yet exist.

These two directories must have permissions set appropriately to allow install accounts to write to them.

When this instance of **UPS** gets chained to current (this cannot happen until tailoring is complete and may or may not happen automatically within `ups_install`; see section 4.2.2), `ups_startup.`*`$UPS_SHELL`* and `ups_shutdown.`*`$UPS_SHELL`* get copied to the `startup` and `shutdown` directories so that the machine can access them during system startup and shutdown.

C.2 OS-Dependent Preparatory Procedures

Besides configuring **UPS** to enable automatic startup and shutdown of products, each node must also "know" to look for and execute the `ups_startup.`*`$UPS_SHELL`* and `ups_shutdown.`*`$UPS_SHELL`* files.

Note that all the examples shown in the following procedures assume the products area to be `/usr/products` and the database to be in `/usr/products/ups_database`. If your system is configured differently, simply make the appropriate substitutions.

C.2.1 Procedure for AIX Systems

On AIX systems, boot time and shutdown procedures are initiated through the **init** process which is controlled by the `/etc/inittab` file. The following steps allow you to connect the **UPS** start and stop procedures to this AIX-standard control file:

1) Add to the *top* of the `/etc/inittab` file:

   ```
   rcups2:0:wait:/etc/ups.shutdown
   ```

   This causes **UPS** to take control at the *beginning* of the shutdown process.

2) Add to the *bottom* of the same file:

   ```
   rcups:2:wait:/etc/rc.ups
   ```

   This causes **UPS** to take control at the *end* of the boot process.

3) Create the file `/etc/ups.shutdown` and enter as the contents the path to your `shutdown/ups_shutdown.`*`$UPS_SHELL`* file. For example:

   ```
   /usr/products/ups_database/shutdown/ups_shutdown.sh
   ```

4) Create the file `/etc/rc.ups` and enter as the contents the path to your `startup/ups_startup.`*`$UPS_SHELL`* file. For example:

   ```
   /usr/products/ups_database/startup/ups_startup.sh
   ```

5) For these last two files, change the permission to 744, the user ownership to *root* and the group ownership to *sys*.

C.2.2 Procedure for IRIX Systems

1) In the `/etc/init.d` directory, create and edit a file called `ups`. Enter the following text into the file:

   `/usr/products/ups_database/startup/ups_startup.csh`

2) Create a symbolic link from `/etc/rc2.d/S99ups` to the `/etc/init.d/ups` file with the following command:

   `ln -s /sbin/init.d/ups /etc/rc2.d/S99ups`

3) In the `/etc/rc0.d` directory, create and edit a file called `K01ups`. Enter the following text into it:

   `/usr/products/ups_database/shutdown/ups_shutdown.csh`

4) Change the permission on `/etc/init.d/ups` and `/etc/rc0.d/K01ups` to 744, the user ownership to *root* and the group ownership to *sys*.

C.2.3 Procedure for SunOS Systems

1) In the `/etc/init.d` directory, create and edit a file called `ups`. Enter the following text into the file:

   `/usr/products/ups_database/startup/ups_startup.csh`

2) Create a symbolic link from `/etc/rc3.d/S99ups` to the `/etc/init.d/ups` file with the following command:

   `ln -s /etc/init.d/ups /etc/rc3.d/S99ups`

3) In the `/etc/rc0.d` directory, create and edit a file called `K01ups`. Enter the following text into it:

   `/usr/products/ups_database/shutdown/ups_shutdown.csh`

4) Change the permission on `/etc/init.d/ups` and `/etc/rc0.d/K01ups` to 744, the user ownership to *root* and the group ownership to *staff*.

C.2.4 Procedure for OSF1 Systems

On OSF1 systems the system startup directories `init.d`, `rc2.d` and `rc0.d` are under `/sbin`, not `/etc`.

1) In the `/sbin/init.d` directory, create and edit a file called `ups`. Enter the following text into the file:

   `/usr/products/ups_database/startup/ups_startup.csh`

2) Create a symbolic link from `/sbin/rc2.d/S99ups` to the `/sbin/init.d/ups` file with the following command:

   `ln -s /sbin/init.d/ups /sbin/rc2.d/S99ups`

3) In the `/sbin/rc0.d` directory, create and edit a file called `K01ups`. Enter the following text into it:

   `/usr/products/ups_database/shutdown/ups_shutdown.csh`

4) Change the permission on `/sbin/init.d/ups` and `/sbin/rc0.d/K01ups` to 744, the user ownership to *root* and the group ownership to *staff*.

C.3 Daemon Process Control Files

You may want some products to run as daemon processes or servers on all nodes of a given flavor in your cluster, and you may want other products to run only on particular nodes. Before any product is configured to run as a daemon you will need to create one or more control files.

The control files provide the actual start commands for the daemon products. To create the files, go to the `$PRODUCTS/../startup` directory[1]. There you need to create one or more files with the name *node*`.products` (where *node* is one of the nodes in your cluster, e.g., `fsgi02.products`) and/or *flavor*`.products` (where *flavor* is one of the flavors in your cluster, e.g., `IRIX+5.products`). If you want a product to run as a daemon on a single node in your cluster, create the file appropriate to that node. If you want it to run on all nodes of a particular flavor, create the file appropriate to that flavor. Records for a particular product can exist in more than one file, i.e. both the node-specific and the flavor-specific files. The system runs all these files at boot time and uses only the first record it finds for a product.

For each product that you want to include in automatic start/stop, you must complete these steps:

1) chain it to one of the defined chains (preferably current, as explained in 2 below)

2) add a record of the following format to the appropriate file(s):

/bin/su - ***login_id*** **-c "ups start [*options*] *product* [*version*]"**

The ***login_id*** is that under which the program must be run. The login id used must be set either to source the file `fermi.cshrc` or `fermi.shrc`, or to run **setup ups** so that other products can be successfully setup. If the ***login_id*** is *root* and the product instance is declared as current, you can leave out the portion **/bin/su -*login_id* -c**, and the line can start from **ups start** (no quotes needed in this case). The **-c** shown here reflects general usage; it can in fact be any chain flag (see 1 above). The chain flag must match the product instance chain.

Similarly, to tell your system to stop the daemon at shutdown, go to the directory `$PRODUCTS/../shutdown`. There you must create files with the same names, *node*`.products` and *flavor*`.products`. Add a record of the following format to the appropriate file(s):

/bin/su - ***login_id*** **-c "ups stop [*options*] *product* [*version*]"**

The same notes regarding the ***login_id*** and chain flag apply.

☞ Make sure the mode of all the *node*`.products` and *flavor*`.products` files is set to 744. This ensures that the files will be executable by *root* and that they have appropriate permissions for avoidance of security holes. If your products area is NFS mounted to all the appropriate machines (i.e. common to them), you only need to create these files once. If not, you need to create these files once for each products area.

C.4 A Summary of the Automatic Startup Process

Since so many different files and directories have similar names, it can be difficult to keep track of the role each plays. The process which takes place at system startup when automatic start and stop are enabled can be summarized as follows:

1) IRIX, SunOS and OSF1: At boot time, the link in the system's startup area (e.g., `/etc/rc3.d/S99ups` for SunOS) points to the `/etc/init.d/ups` file (IRIX and

1. If *PRODUCTS* points to more than one database, use the `startup` file in the same database as the currently running instance of **UPS**.

SunOS) or the `/sbin/init.d/ups` file (OSF1) which runs `$PRODUCTS/../startup/ups_startup.`*`$UPS_SHELL`*.

AIX: At boot time, the `/etc/inittab` file calls `/etc/rc.ups` which in turn calls `$PRODUCTS/../startup/ups_startup.`*`$UPS_SHELL`*.

2) `$PRODUCTS/../startup/ups_startup.`*`$UPS_SHELL`* runs the appropriate `$PRODUCTS/../startup/`*`node`*`.products` and/or `$PRODUCTS/../startup/`*`flavor`*`.products`.

It also runs `setups.`*`$UPS_SHELL`* so that appropriate environment variables and aliases get set.

3) The *`node`* or *`flavor`*`.products` in turn runs the **`ups start`** command.

4) **`ups start`** executes the `start` file in the `ups` directory of the daemon product.

The process for automatic stop is similar.

Glossary

This appendix defines terminology as it is used in this document.

active product instance

The product instance that is currently setup. The environment variable *$**PRODUCT**_DIR* (where ***PRODUCT*** is the product name) points to the root directory of the active instance. The *active instance* may be different than the *current instance*.

build

The process by which a distributable version of a software product is constructed. The build procedure results in a unique combination of product name, version, and flavor of the product. The actual process varies by product and by developer. It can simply consist of a set of copy commands, or be as sophisticated as generation of executables from a master source library of the software.

build requirements

Any **UPS** product instances that must be installed and declared prior to successfully building the product instance in question. During the build process the setup scripts for the build requirements are run with the **`-b`** option. Generally build requirements are not distributed with a product instance.

chain

A chain is a **UPS** database entry (a record in a product file) that points to a declared product instance, tagging the product instance according to its status (e.g., current, test). Chains allow users to specify the version of a product according to its status, rather than by its version number. The defined chain names are: *current*, *test*, *development*, *new*, and *old*. Their corresponding options (or flags) used in commands are: **`-c`**, **`-t`**, **`-d`**, **`-n`**, **`-o`**. Product instances do not have to be chained.

Chains are set by the **`ups declare`** command; hence the term *declare a product instance as current* or *declare a product instance as test*.

chain record

See *chain*. A chain record in a product file starts with `Chain:` and ends with `end.`

cluster

For the purposes of this document, a cluster is set of CPU nodes which share one or more **UPS** databases and product areas. Generally the nodes of a cluster also share (at least) login areas.

configure a product instance

For any product that requires configuration, a `configure` script is provided in the product's `ups` directory. In **UPS**, *configuring a product instance* means sourcing this script, if it exists, automatically during product declaration. The script performs all the configuration needed for the product to run, minus that which requires input from the installer (see *tailor a product instance* for that portion).

current instance (of a product)

A product instance that is declared as current in the database (i.e. to which the chain current is set to point). The current instance of a product is the default for the **`setup`** command. For a given product, there may be a current instance for each (extended) flavor.

daemon process

A background process that is configured to start up automatically on a system at boot time and to stop at shutdown.

declare a product instance to UPS

The **`ups declare`** command makes a product instance known to the **UPS** database by adding an instance record to the appropriate product file (or creating the product file and the instance record, if this is the first instance of the product to be declared). Declaration does not by itself make the product instance usable since any use requirements must also be satisfied, but declaring the product instance is a prerequisite for use.

declare a product instance current

Declaring a product instance as current to the **UPS** database modifies the product's product file by adding or changing the *current* chain record such that it points to this instance. Product instances can also be declared as test, development, new or old.

declared product instance

An instance of a product which has been declared to the **UPS** database, and therefore for which an instance record exists in the product's product file.

dependency

A dependency (or product dependency) is a use requirement or a build requirement.

development instance of a product

The product instance that is declared as development in the database (i.e. to which the chain *development* points).

extended flavor

First, see *flavor* below. In some cases separate product instances are needed for different releases of a single operating system, all else being the same. This allows a mixed cluster (e.g., IRIX 4 and IRIX 5) to share a **UPS** database. In these cases the flavor specification must indicate the operating system release, since that is the only area of differentiation. A flavor specification may also contain information about compilation options. A flavor specification that contains either of these types of extra information is called an *extended flavor.*

flavor

To indicate the operating system (OS) dependency of a product instance, we use the term *flavor.* This extra term allows us to differentiate by operating system while maintaining the same product name and version number. A simple flavor (as opposed to an extended flavor) is the UNIX operating system name as returned by the command **`funame -s`** (for example `IRIX` or `SunOS`). Some products do not require customization for the different operating systems (typically those without compiled code), but most do and therefore come in several flavors.

install a product instance

To copy a product instance to a local system from another location and perform the necessary steps to make it work.

instance record

In a product file there is one instance record for each installed, declared instance of the product that is declared to **UPS**. This record is the actual product declaration which allows **UPS** to locate and retrieve the product instance. An instance record starts with `Instance:` and ends with `end.`

KITS

The **UPS** distribution database on the central product distribution node at Fermilab, upd.fnal.gov, is called `KITS`. **UPS** products are stored in `KITS` as tar files. **UPD** accesses the `KITS` database by default.

operating system (OS)

A control program for a computer that allocates computer resources, schedules tasks and provides the user with a way to access the resources. See document DR0010 in the Computing Division web pages for the latest information on supported operating systems.

operating system release (OS release)

Like other software, an operating system gets fixed and enhanced periodically, and is released by the vendor with a new release (or version) number (e.g., IRIX 5.1, IRIX 5.2). Sometimes **UPS** products must be changed to continue to work properly under a new operating system release.

operating system type (OS type)

The name of the basic operating system, without release number, as returned by the command **`funame -s`** (for example `IRIX` or `SunOS`). This corresponds to what we call *simple flavor*. See *flavor*.

operating system version (OS version)

See *operating system release*.

parent product

A dependent product's *parent product* is that for which it is a *dependency*. A product may have multiple parent products.

platform

Platform technically refers to the machine type (hardware) of a computer system. However, since up to now in the UNIX world there has been a near-perfect correspondence between hardware platform and OS type (e.g., Digital Alphastations run OSF1), sometimes platform is used loosely to refer to the OS type.

product

See *UPS product*

product declaration file

See *product file*.

product developer

A person who develops and maintains one or more software products, and makes them available for distribution.

product file

There is one file in the **UPS** database for each product, called its *product file*. The name of each product file is the name of its associated product. A product file contains all the information **UPS** needs to locate, retrieve, and setup any of the installed, declared instances of the product. This information consists of chain records if any chains have been defined, and one instance record for each instance.

Following is an example for the product **exmh**. The product file's name is `exmh`:

```
Product:        exmh

# This file is maintained by Unix Product Support.

# DO NOT EDIT THIS FILE DIRECTLY.



db_version:     "1.2"



Chain:
        test    SunOS+5 v1_6_6  11-13-96        "aheavey"
end.

Instance:       "v1_6_6"        SunOS+5 /home/dcdsv0_lv0/aheavey/ups_play/products/
SunOS+5/exmh/v1_6_6
                11-13-96                "aheavey"



                Authorized_Nodes: [*]

                Requirements:    {(< glimpse,current,SunOS+5),(< mh,current,SunOS+5),(<
www,current,SunOS+5),(< mimetools,current,SunOS+5),(< expect,current,SunOS+5),(<
ispell,current,SunOS+5),(< popclient,current,SunOS+5)}
                                 {}
                                 {}
                                 {}

end.
```

product installer

A person who gets a product from a distribution node (through **UPD** or anonymous **ftp**) and installs it on his or her local system (often the local system administrator acts as the product installer).

product maintainer

See *product developer*.

product instance

The term *product instance,* or just *instance*, is used to represent the unique combination of product name, version and (extended) flavor. For a given product, multiple instances may exist in the **UPS** database to allow users a choice of version and/or flavor. For products that are not customized to a particular UNIX operating system, flavor is not part of this designation. A product instance may be chained; hence the term "the current instance of a product".

product name

The name of a **UPS** product as it appears at the top of its product file.

product root directory

The directory in which a product instance and its associated files reside. The product instance generally has a directory structure of its own, starting at this root directory. Each instance of a product has a separate product root directory.

product user

Also called *end user*; a person who uses products that have been installed in and declared to the **UPS** database.

product version

The net result of any change to an existing product is that a new *version* of the product is created; it is still the same product, but it will usually run a little differently. The versions of a product are tracked by version numbers, e.g., v1_0, v1_1, etc. **UPS** allows for multiple versions of a given product to be accessible to end users.

PRODUCTS

The environment variable that FUE sets automatically to point to the **UPS** database on your system. If multiple **UPS** databases exist, *PRODUCTS* can be reset to a space-separated list of declared databases.

PRODUCT_DIR

PRODUCT here is the name of the product in upper case. This is the environment variable that points to the the product root directory of the active instance of the product; it gets set when the **`setup`** command is run on a product instance.

recursive dependency

A product instance may have dependencies which in turn have dependencies of their own. A recursive dependency is one that gets setup by default if its parent product is setup. Use requirements are always used recursively. Build requirements never are.

root directory for product

See *product root directory.*

setup

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). **`setup`** makes the appropriate changes to the user's software environment in order to make the product available for use. It sets the environment variable ***$PRODUCT_DIR*** to point to the root directory of the product instance and sources the setup script appropriate to the shell family, if it exists (e.g., `setup.csh` or `setup.sh`). Any dependent products get setup by default at the same time.

simple flavor

See *flavor.*

tailor a product instance

Tailoring is the aspect of the product configuration that requires input from the product installer (e.g., information such as the location of hardware devices for a software driver package). If tailoring is required (for most products it is not), the file `tailor` in the product instance's `ups` directory is supplied in the format of an interactive script. Then during the product declaration (if done in **UPD**), the user receives a message indicating that tailoring is required. To *tailor* means to run this script, if it exists. The user must explicitly tailor the product instance using either the **UPD** menu interface or the **UPS** command **`ups tailor`**; tailoring is not performed automatically.

tar

The tar (tape archive) utility can create, add to, list, and retrieve files from an archive file.

tar file

A tar file is in the archived format, and must be unwound for use. **UPS** products are generally stored in `KITS` as tar files. **UPD** can be used both to make tar files and to unwind them.

token

Extended flavor designations have optional suffixes that indicate compilation options. These optional elements of the designation are sometimes called *tokens*.

unsetup

`unsetup` sources the appropriate unsetup script in the product instance's `ups` directory to undo the changes to the user's software environment made by **`setup`** in order to make the product no longer available for use. Any use-dependent products get deactivated automatically at the same time.

UPS - Unix Product Support

UNIX Product Support is a software support toolkit which provides a uniform interface for accessing all the UNIX products provided and/or supported by the Computing Division. **UPS** also allows for a more unified and coordinated product support strategy across all the supported UNIX operating systems. **UPS** is itself a product that must be installed on any machine that will be used to run other **UPS** products.

UPS has two parts: a database which functions as a central repository of information about the products, and a set of procedures/programs to manipulate the database.

UPS database

A directory that functions as a repository of information about all the product instances declared to **UPS**. **UPS** allows multiple installed and declared instances of each product. The database contains one file for each product, called the product file, which stores pointers to and information about the declared instances of the product.

UPS_EXTENDED_FLAVOR

The environment variable which can be used to specify elements of an extended flavor designation beyond the operating system name and release.

UPS product

All software products distributed and managed by the **UPS** system are called **UPS** products. **UPS** products include Fermilab-written programs, a wide range of public domain software, and a host of third party licensed (proprietary) products.

UPS_SHELL

The environment variable that FUE sets to either `sh` or `csh` at login time, according to which shell family your default shell belongs to, Bourne or C. In cases where differentiation by shell family is needed, separate executables are provided in a product's `ups` directory for the same function (e.g., `setup.sh` and `setup.csh`). **UPS** uses *UPS_SHELL* to determine which executable to run. In this manual, we use `filename.$UPS_SHELL` to indicate "either `filename.csh` or `filename.sh`".

use requirements

Additional **UPS** products that must be installed, declared, and setup to ensure the successful operation of the parent product. These along with build requirements are often referred to as *dependencies*. When a product instance is setup, its use requirements also get setup by default.

version

For a product see *product version*; for an operating system see *operating system release*.

Index

Symbols

A

B

C

D

E

F

G

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z