

# Chapter 12: UPS and UPD Pre-install Issues and General Administration

In this chapter we take a step back with regard to Chapter 11: *Maintaining a UPS Database*, and assume that you have not yet installed **UPS/UPD**, or created a **UPS** database and products area. We guide you through the administrative decisions and tasks that are involved in preparing to implement **UPS/UPD**. Towards the end of the chapter there is also some information regarding general administrative tasks. For machines running AFS or NFS, also see Chapter 13: *Providing Access to AFS Products*.

## 12.1 Choosing Installer Accounts

---

Here we assume that you're planning to implement **UPS/UPD** on your local system and maintain a **UPS** database and products area there<sup>1</sup>. In this section we discuss options regarding installer accounts. Choosing installer accounts wisely is important because the account used by the installer determines a product's ownership.

### 12.1.1 Single Installer Account

For a given system, if the number of people who install products from a distribution node and manage the local **UPS** database is small and relatively static, then, from a system management point of view, having a single account used for these purposes is often simplest. We suggest you create a standard account called *products* to be used for all product installations, then no special permissions are needed for other accounts. This single account method prevents problems for any product maintainer who has to remove or change a product installed by a different person. As far as the system is concerned, the installer/maintainer is always *products*.

---

1. We specifically don't say "planning to install **UPS/UPD**" because in some configurations (notably AFS machines), you may not need to install the products locally.

## 12.1.2 Multiple Installer Accounts

Our experience has shown that in many situations a single installer account is not adequate. It leads to confusion on systems where several people install products, especially if they don't establish and follow a procedure for communicating with each other regarding product maintenance. If they all use the *products* account, it is much more difficult to track which person performed a particular operation.

As systems grow in size, and more and more users become product installers, we have found that it is better for them to use their normal login accounts when installing products. Since installers need the correct group id to write to the products area (often this group is the *products* gid), system managers can simply add them to the *products* group.

This strategy does warrant more caution in two areas: file system semantics and **UPD** configuration, as described in sections 12.2 *Setting gids for Multiple Installer Accounts* and 12.3 *File Ownership, Permissions and Access Restrictions*.

## 12.1.3 Separate Installer Accounts for Different Product Categories

You might want the ownership of the product home area to be different based upon whether the product is being downloaded from a distribution node, or was developed on the local machine.

For example, say you have one person who installs all the products needed on your system from the distribution node, and several people developing **UPS** products locally on your system. In this case, you may find it simplest to use the single installer *products* account for the downloaded products, and the developers' own accounts for the locally-developed products.

## 12.2 Setting gids for Multiple Installer Accounts

---

When you allow multiple accounts to install products as described in section 12.1.2 *Multiple Installer Accounts*, you use group ids (gids) to control access to the product areas. Group ids get set on files and directories differently depending on whether you use System V or Berkeley semantics (the choice for most current Unix systems). With System V semantics, new directories and files have the same gid as the account which created them. With Berkeley semantics, new directories inherit the gid of the parent directory.

If you plan to allow product installs from multiple accounts, we strongly recommend using Berkeley semantics. On newer systems, you select Berkeley by setting the set-group-id bit on the directory (i.e., `chmod g+s <directory>`). On older systems this may require special options on the filesystem mount.

**UPD** unwinds tar files with the permissions given when they were created. Most of the product files do *not* have group-write enabled. You might find it useful to make the products you install group-writable so that the same set of accounts that installs products can also delete them. You will be able to set this in your local **UPD** configuration file (`updconfig`, described in Chapter 32: *The UPD Configuration File*) in a `postdeclare` action, e.g.,:

```
ACTION = POSTDECLARE
    Execute("chmod -R g+w ${UNWIND_PROD_DIR}", NO_UPS_ENV)
```

Whether products are downloaded from a server or developed locally, the group ownership is still an issue. For a locally developed product, sometimes many accounts need to be able to delete or modify it, and the group access needs to be set accordingly. There is no automatic way in **UPS/UPD** to enforce this, however.

## 12.3 File Ownership, Permissions and Access Restrictions

---

### 12.3.1 Product Files

Product file ownership is determined by the installer account, as discussed in section 12.1 *Choosing Installer Accounts*. To determine adequate file permissions for **UPS** products, it is important to consider your user community. Clearly users will need read access to the products they are allowed to use, but do you need to restrict access to any subsets of products? Do all users share the same group ids? Who needs to install/delete products? Who will be declaring products to the **UPS** database? These are the kind of questions you should keep in mind when setting up file permissions.

Some products, e.g., licensed products, should not be accessible to all users. A simple way to restrict user access is to configure a special **UPS** database for your restricted products and make that database visible only to the appropriate subset of users. Sometimes you may need to protect the files themselves, as well. When extra security is required, we recommend that you create a special gid for the restricted products and that you turn off world access. You will be able to configure **UPD** to set this group id on those products; this example shows how it can be done in `updconfig` (file described in Chapter 32: *The UPD Configuration File*):

```

group:
    product = some_licensed_product
common:
    UPS_PROD_DIR = ...
    UNWIND_PROD_DIR = ...
    ...
    action = postdeclare
        Execute("chgrp -R <special_group> ${UPS_PROD_DIR}",
NO_UPS_ENV)
        Execute("chmod -R o-r ${UPS_PROD_DIR}", NO_UPS_ENV)
end:

```

A stanza like this in the `updconfig` file should be otherwise identical to the existing default stanza that would have handled this product; only the `postdeclare` action should be added. Within the `updconfig` file, the default stanza should come *after* any specialized stanzas, since **UPD** uses the first match it encounters.

## 12.3.2 Database Files

The **UPS** database files and pointers therein (e.g., to man page areas) have their own set of file permissions that are generally more open than the product files themselves. In almost all cases, the **UPS** database files should be group-writable. Set your `umask` to `002` before running **upd install** or **ups declare** on products to ensure this. The files should be owned by an account with a `gid` that is shared by the installation account(s) and by any developers creating **UPS** products locally.

## 12.4 Product File Location and Organization

---

### 12.4.1 Considerations

**UPS/UPD** imposes no restrictions on where product files can reside. Product files can reside on any file system on any disk. Different instances of the same product can reside in different file systems. **UPD** allows you configure where you want specific products unwound. The more generic your rules, the simpler your **UPD** configuration file (`updconfig`, described in Chapter 32: *The UPD Configuration File*).

As a system manager, you should check how much space is available in each partition. If you have many machines sharing disks, determine how they are shared. Do they have the same mount points on all machines? Is the `/usr/local` area shared across any machines? Are you cross-mounting to

different OS flavors? How should products be organized in the file system? What do you want to keep local to a particular machine, local to all nodes of particular flavor, or shared cluster-wide?

Take into account these issues when deciding where to put product files, as well as databases and other files. Your system configuration affects how many copies of a given product instance may be required on a given group of systems. It also determines on how many separate machines a product instance may require that special actions be performed (e.g., configuring or declaring it current).

## 12.4.2 Single Flavor or Single Node Systems

For simple systems, e.g., single flavor or single node, the product files typically live under:

```
/path/to/<product_name>/<product_version>
```

However, this is not always adequate. An example is if you have more than one instance of the same version, but with different qualifiers. In cases like this, the following is a better model:

```
/path/to/<product_name>/<version><qualifiers>
```

To implement this, all you need is one stanza in your `updconfig` file which lays out all the products in this way.

However, experience has shown us that it's not always wise to assume that your system will forever have only one flavor; system upgrades are not so predictable. See the next section, 12.4.3, for more ideas.



## 12.4.3 Multi-Flavor and/or Multi-Node Systems

Two separate directory structure conventions have evolved for products installed on multi-flavor systems. Each has its own advantages and disadvantages.

- 1) /path/to/<product\_name>/<flavor>/<version><qualifiers>

In this configuration, all versions of a product are unwound on the same file system. This makes it easy to see what software is available using simple UNIX **ls** commands.

- 2) /path/to/<base\_flavor>/<product\_name>/<version><qualifiers><flavor>

This second case supports a separate file system for each operating system. If one disk is currently not available, work can still continue on other machines.

Note that in both cases, /path/to/ can be anything, and need not be the same for each product (but in general it's easiest to maintain all products in the same area). If more than one file system is used, **UPD** needs to be configured to know which products are installed in which file system. If the products need to be seen across multiple machines, it is important that all the file systems be visible under the same directory structure on all machines.

Whichever structure you choose, when you have multiple flavors (or possible future multiple flavors) you may find it useful to create a flexible configuration that allows **UPS/UPD** to pick a different product directory based on flavor.

The following is a sample `updconfig` file showing how to do this. The flavors of products we expect to install, in this case **NULL** and **IRIX+6** products, get put in with nice, short pathnames (see `UPS_PROD_DIR` in the file), while everything else gets a longer pathname that explicitly includes the flavor.

```
file=updconfig
group:
  flavor=NULL
  flavor=IRIX+6
common:
  UPS_THIS_DB = "/fnal/ups/db"
                    UPS_PROD_DIR =
"$ {UPS_PROD_NAME} / $ {UPS_PROD_VERSION} $ {UPS_DASH_QUALIFIERS} "
  UNWIND_PROD_DIR = "$ {PROD_DIR_PREFIX} / $ {UPS_PROD_DIR} "
  UPS_UPS_DIR = "ups"
  UNWIND_UPS_DIR = "$ {UNWIND_PROD_DIR} / $ {UPS_UPS_DIR} "
  UNWIND_TABLE_DIR = "$ {UPS_THIS_DB} / $ {UPS_PROD_NAME} "
  UPS_TABLE_FILE = "$ {UPS_PROD_VERSION}.table"
end:
```

```

group:
  flavor=ANY
common:
  UPS_THIS_DB = "/fnal/ups/db"

UPS_PROD_DIR="{UPS_PROD_NAME}/{UPS_PROD_VERSION}{UPS_DASH_FLAVOR}{UPS_DASH_QUALIFIERS}"
UNWIND_PROD_DIR = "{PROD_DIR_PREFIX}/{UPS_PROD_DIR}"
UPS_UPS_DIR = "ups"
UNWIND_UPS_DIR = "{UNWIND_PROD_DIR}/{UPS_UPS_DIR}"
UNWIND_TABLE_DIR = "{UPS_THIS_DB}/{UPS_PROD_NAME}"
UPS_TABLE_FILE = "{UPS_PROD_VERSION}.table"
end:

```

## 12.5 Database File Location and Organization

---

### 12.5.1 Choosing Single or Multiple UPS Databases

The **UPS** database files are not large and do not require much disk space. They can reside on any file system; either with the product files, or not. You can choose to have one **UPS** database for all products installed on a mixed flavor cluster, you can have a separate **UPS** database for each flavor type, or you can choose another configuration. Typically, a single **UPS** database is used for all flavors. Multiple databases are more often used to house sets of user-specific products (e.g., CDF off-line products) rather than to distinguish between operating system flavors.

### 12.5.2 UPS Database File Pointers

A **UPS** database contains pointers to directories, the most important of which are the ones that contain the man page files and the **UPS** environment initialization files (`setups.[c]sh`; discussed in section 2.7.1 *Initializing the UPS Environment*). These directories can be on different file systems from the database itself. In order to best determine the locations and permissions for these directories, system administrators need to understand how they are used.

When a product is declared current, its man pages, if any, are (optionally) copied to a system-wide **UPS** product man page directory for which the location is set in the database configuration file, `dbconfig`. This man page directory should be writable by anyone with the authorization to declare products current. Historically, this directory has been maintained separately from normal system man pages, just to avoid any confusion or overlap.

Often this **UPS** man page area is shared between OS flavors. This is an easy solution, but it can lead to confusion on mixed OS clusters. If you have different versions of a product declared current for each flavor, the man pages will likely get out of sync. For example, say you have an installed IRIX “current” chain for a product, and you later declare a SunOS instance “current”. If the man page area is shared, this new man page overwrites the older one.

The `setups.[c]sh` files make the **UPS** environment available by defining the **UPS** database(s) and setting up **UPS** itself. These files are invoked by each individual user’s login scripts, and their location is configured in the **UPS** database configuration file `dbconfig`. In the past, these files have been kept in `/usr/local/etc`; however, this has been a problem on machines where the person installing **UPS** does not have *root* access. A more common practice now is to put them in a directory parallel to the main **UPS** database itself, e.g., `$PRODUCTS/./etc`.

## 12.6 Installing UPS for Use Without a Database

---

**UPS** can be installed on a machine to manage products without a **UPS** database, as mentioned in section 2.5 *Using UPS Without a Database*. This flexibility is provided primarily for off-site users who for one reason or another do not want to maintain a **UPS** database on their local system.



Before making the decision to do without a **UPS** database, be aware that **UPS** allows much more flexibility now. **UPS** is no longer tied to products such as **futil** and **systools**, and the database can be maintained anywhere on your system.

To install **UPS** in this way:

- 1) Create a products area (not strictly necessary, but this keeps things more organized).
- 2) Download **UPS** into the products area using **FTP**, as described in Chapter 8: *Installing Products using FTP*.
- 3) Initialize your **UPS** environment as described in section 2.7.1 *Initializing the UPS Environment*.

A locally installed product instance would have no version or chain files, of course, but it would need a table file (very few products come without one). If there are any functions in the table file for setting up product dependencies

(e.g., the function `setupRequired` or `setupOptional`), you'll need to check each of these functions to make sure it includes a table file specification. This is necessary in order to continue to bypass the need for a database.

Make sure your users are made aware that:

- **UPS/UPD** functionality requiring or operating on a **UPS** database is not supported when **UPS** is implemented without a database (e.g., `setup` should work, but `ups declare` won't).
- Any **UPS/UPD** command must include all of the information that normally would have been read from a database. In particular, all commands require the `-m` option for table file name (and usually `-M` for the table file directory).

## 12.7 CYGWIN (Windows NT) Issues

---

A number of **CYGWIN**-specific problems have been encountered. We'll highlight the most frequent ones here.

### 12.7.1 Using Correct Perl Version

You must run a version of **perl** that is built against **CYGWIN** itself, not one arbitrarily obtained off the net. You can get a working **perl** for **CYGWIN** from **KITS**.

### 12.7.2 Mounting the CYGWIN bin Directory

The **CYGWIN** `bin` directory should be mounted in `/usr/bin` and a symbolic link must be made from `/usr/bin` to `/bin`.

### 12.7.3 Setting Environment Variables

For **UPD** to work properly, make sure that:

- `$TMPDIR` is set to a directory that really exists and that can be used for holding temporary files
- `$USER` is set to your userid

## 12.8 General Administration Issues

---

### 12.8.1 Upgrading an Older System

Prior to **UPS/UPD** v4, the Fermi User Environment (FUE) included a suite of utilities and binaries which were copied into `/usr/local`. The required FUE utilities, with a brief description of each, were:

**systools** system administration utilities, located under `/usr/local/systools`; the utilities included the Fermi login scripts and ups initialization files, as well as the **cmd** function (which allows non-privileged users to do specific privileged things) and various “**cmd <utilities>**”, e.g., **adduser**, **renice**, etc.

**funkern** programs called from the Fermi login scripts, copied into `/usr/local/bin`

**fulib** C-callable library of the **funkern** utilities

**futil** a random assortment of odds and ends that were considered to be Generally Useful Utilities, copied into `/usr/local/bin`

FUE has recently been redesigned utilizing the new features of **UPS** v4. The redesign has been very successful, and is in use on all FUE-compliant systems installed since the autumn of 1999. To upgrade an older system and clean up the vestiges of the old FUE, you must first upgrade to the new FUE which includes **UPS/UPD** v4\_0 or higher (v4\_5\_1 as of May 2000), and **systools** v6\_0 or higher (and all of its dependencies). You must also convert existing accounts' login scripts to the new FUE syntax. Two technical notes are available to guide you through this process. See:

TN0088 *Files which you may be able to remove from /usr/local*  
(<http://www.fnal.gov/docs/TN/tn0088.html>)

TN0089 *FUE Login Methodology*  
(<http://www.fnal.gov/docs/TN/tn0089.html>).

### 12.8.2 Adding a New Database and/or Products Area

There are different reasons for adding a new products area. For example, you may run out of room in one area and need another, or you may want different categories of products stored in different areas and accessible to different groups of people. For development and/or testing purposes, it is often convenient to install products in a separate products area and declare them in a separate, associated **UPS** database.

## Checklist

Here is a checklist of the tasks involved (assuming **UPS/UPD** is already installed and working on your system):

- Create a directory for the **UPS** database (e.g., `/path/to/db`).
- Create a products area
- Create a **UPS** configuration file  
(`/path/to/db/.upsfiles/dbconfig`).
- Create a **UPD** configuration file  
(`/path/to/db/.updfiles/updconfig`).
- Create an empty dummy directory under the database (e.g., `/path/to/db/xxx`).<sup>1</sup>
- Prepend your database path to `$PRODUCTS`; colon separated (e.g., `setenv $PRODUCTS "/path/to/db:${PRODUCTS}"`).
- If you are at **UPS** version `v4_4a` or earlier, include the file `updusr.pm` under `/path/to/db/.updfiles`. Insert as the file contents the following single line: `require 'default_updusr.pm'`;

## Example

Here we provide an example of creating a new product area (we'll use `/fnal/ups/prd`) complete with a new database (`/fnal/ups/db`). We've created a shell script called `newupsarea`. To run it, we must supply the directory which houses the database and products area (`/fnal/ups`) as an argument. First, let's look at this script:

```
#!/bin/sh

db=$1/db
pdp=$1/prd
mkdir -p $db/.upsfiles
mkdir -p $db/.updfiles
mkdir -p $pdp
cp $UPD_DIR/ups/updconfig.template $db/.updfiles/updconfig
cp $UPD_DIR/ups/updusr.pm.template $db/.updfiles/updusr.pm
cp $UPS_DIR/ups/dbconfig.template $db/.upsfiles/dbconfig
perl -pi.orig -e 's{/fnal/ups}{'$1'};'
$db/.upsfiles/dbconfig
# work around empty db bug...
mkdir $db/xxx
```

To run this, issue the command:

---

1. Given a database with the subdirectories `.upsfiles` and `.updfiles` and nothing else, the command `ups list -K PROD_DIR_PREFIX` fails to list that database and its prefix. In turn, **UPD** fails to find `PROD_DIR_PREFIX` for that database, and so on. Simply adding an empty subdirectory solves the problem.

```
% newupsarea /fnal/ups
```

This creates the database (`/fnal/ups/db`), its subdirectories, the `PROD_DIR_PREFIX` directory (`/fnal/ups/prd`), copies the **UPS** and **UPD** configuration files from templates, and changes the references to `/fnal/ups` in the copy of the template `dbconfig` file. Finally, it makes a directory (`xxx`) under the database<sup>1</sup>.

### 12.8.3 Collecting Statistics on Product Usage

**UPS** supports recording of the following statistics on product usage and **UPS** database access:

- Userid of person executing **UPS/UPD** command
- Date and time
- Which command was executed (including options and arguments)
- Which product instance was selected by command

Collection of statistics is controlled by an entry in one or more database files. See the reference section 28.6.3 *STATISTICS* for a description of how to implement this.

---

1. Given a database with the subdirectories `.upsfiles` and `.updfiles` and nothing else, the command `ups list -K PROD_DIR_PREFIX` fails to list that database and its prefix. In turn, **UPD** fails to find `PROD_DIR_PREFIX` for that database, and so on. Simply adding an empty subdirectory solves the problem.