

Chapter 2: Overview of UPS, UPD and UPP v4

UPS, **UPD** and **UPP** are the utilities provided by Fermilab's Computing Division for managing and standardizing software product development, distribution, support and access. This overview chapter describes these utilities briefly, and discusses the reasons for which this particular product methodology was chosen and developed. The chapter also describes:

- features of products maintained and distributed under this system (called **UPS** products)
- **UPS** databases
- the **UPS** software environment

2.1 Introduction to UPS, UPD and UPP

UPS (UNIX Product Support) is a software support toolkit developed at Fermilab for the management 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 three principal user benefits are:

- a uniform interface for accessing all products on a UNIX (or UNIX-like, e.g., Cygwin) system via the **setup** command
- unified and coordinated support of in-house and vendor-supplied software across all the supported UNIX operating systems
- the capacity for running multiple concurrent versions of products on the same system, with a standard, simple version-selection mechanism

UPD (UNIX Product Distribution) is a companion product to **UPS**, and provides the functionality for uploading/downloading products between local systems and product distribution servers.

UPP (UNIX Product Poll) is a layer on top of **UPD** that allows a client to request notification of changes in a distribution node database and to download pre-specified products. **UPP** can be automated. This is a useful tool for keeping abreast of changes/enhancements to your favorite products.

A **UPS/UPD/UPP** installation usually has several 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 **UPS** database configuration file which contains system-specific information that customizes the **UPS** installation on a node or cluster
- a **UPD** configuration file that tells **UPD** which database to use to declare a product, in what directories to unwind the product and its related files, what naming conventions to use for various metadata files, and other information.
- the **UPS**, **UPD** and **UPP** executables which manipulate and view the database(s) on client and server nodes
- a **UPP** subscription file that lists product instances you are interested in tracking and a list of commands to perform when new instances appear (notify by email, install the instance, etc.).
- an entry in the crontab file to run **UPP** at some periodic interval.

2.2 Motivation for the UPS Methodology

Why has Fermilab developed and implemented the **UPS** product support methodology? The principal reason is to provide self-contained products. Each release of a self-contained product can be installed and accessed independently of other releases. There are two main advantages to this, which are especially important for applications such as real-time data acquisition:

- Multiple versions of a software product can be made available concurrently. This is useful in many situations, especially when some products depend on particular versions of others for compiling or running. Multiple concurrent versions also makes possible the second 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.

The **UPS** methodology also provides tracking. A glance at the **UPS** database tells you what version of a product you're running; you don't need to keep track of it elsewhere or risk forgetting. You can also easily tell if different machines are running the same version of a product.

All users of UNIX utilities and software products on a system running **UPS** will appreciate these additional features:

- capability for supporting a cluster of UNIX systems from different vendors fitted with common software products (and optionally a common products disk)
- an easy and consistent method of accessing a variety of 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
- a single product development/distribution methodology and set of standards
- a wider audience available to test new products

2.3 UPS Products

Products distributed and managed by **UPS** on a distribution or user node are called *UPS products*. Typically, **UPS** products on a system are declared in a **UPS** database on that system. **UPS** products can be maintained on different disks, and/or in different directories. Each product in a **UPS** database is managed via a set of files that provide product management information to **UPS**, e.g., the location of each copy of the product, what its status is (e.g., appropriate for general use, for testing, for development), what needs to be done when the product is installed or accessed, and so on.

UPS products are generally self-contained and portable, laid out in a directory structure under what we call a *product root directory*. The structure of a product's directory tree is not dictated by **UPS**, but generally it includes (at least) areas for the executables (`bin`), for test scripts (`test`), and for documentation (e.g., `man`, `catman`, `docs`, `html`).

2.3.1 Versions

UPS supports multiple concurrent versions of software products. Each version of a product is installed and accessed independently of other versions. When a new version of a product becomes available, an existing directory tree is not replaced with another; rather, a branch is added for the new version. (See the diagram in section 2.4.2 *UPS Database Structure*.)

2.3.2 Flavors

Many programs require separate compilations for each of the different UNIX operating systems. **UPS** allows you to maintain a separate directory tree for each compilation (and related files) of the same product. To distinguish between different OS-dependent compilations, we use the term *flavor*. This additional term allows us to maintain the same product name and version across the different operating systems and OS versions, which is desirable since the same program source files are generally used in the separate compilations.

Several different copies of a product may exist on a given system. When you use **UPS** commands to manipulate or use a product, the system needs to have enough information to select the appropriate one. All **UPS** commands support a **-f** option¹ allowing you to specify flavor.

The flavor of a particular product compilation as declared in a **UPS** distribution database may or may not indicate the version of the OS for which it was compiled. The standard we have adopted for flavors in the **KITS** distribution database on the Computing Division's central product distribution node *fnkits.fnal.gov* is:

- For products which have no compiled programs, and are thus operating system-independent, a special flavor of NULL is used.
- Flavored products are declared with the full flavor specification of the OS on which they are built.

On user systems the flavor of a product may be declared differently. For example, products which can run on multiple releases of a single OS are sometimes declared on user systems with the UNIX OS name only. For example, a product that runs on all IRIX releases but is declared in **KITS** as IRIX+6.5 may be declared on a user system with the flavor IRIX. (This practice is declining. It is discouraged because it is difficult to maintain and can cause setup problems when the product is a dependent product of another. Dependent products are defined in section 2.3.6 *Product Dependencies*.)

2.3.3 Qualifiers

The product developer may include information about options used at compilation time (e.g., `debug` or `optimized`) or other qualifying information for easy identification of special compilations. This information is declared in the form of *qualifiers* in a distribution database. When a product is declared to **UPS**, the installer has the choice of declaring these qualifiers or omitting them in the declaration.

1. There are also the number options **-0** through **-3** which can be used in place of **-f**; **-H** can be used to run a **UPD** command as if the local flavor were as specified. See Chapter 25: *Generic Command Option Descriptions* for descriptions.

Declaring the qualifiers allows full identification of the product compilation on your system. The drawback is that to access a product compilation declared with one or more qualifiers, a user *must* specify the qualifier(s) when accessing that compilation.



No standard set of qualifiers has been defined; the naming of qualifiers is at the discretion of the product provider, and thus may vary from product to product.

2.3.4 Product Instances

Each installed copy of a product that is declared to a **UPS** database is called an *instance* of the product. Within a database, a product instance is distinguished by a unique combination of product name, version, flavor, and qualifiers. In the case of multiple databases, the database specification is often also needed to distinguish an instance because identical instances can be maintained in different databases. The concept of *chains*, discussed in section 2.3.5, allows users to easily access the appropriate instance of a product according to their needs without having to remember its version number and other details.

2.3.5 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 can be thought of as an alias for a particular product instance. It indirectly “attaches” a chain name to a product instance, thereby tagging the product instance according to its status.

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

Chain Names, Options, and Usage

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

Additional chains may be defined by developers and installers and other users. The command option **-g** is provided for this purpose. For an example of how it's used, see section 11.2 *Declare a Chain*.

2.3.6 Product Dependencies

Many **UPS** products require that other **UPS** products be installed, declared and setup for proper functioning or for use of special features. These other products are referred to as *dependencies*. A dependency is generally an independent product that is maintained in its own individual product root directory. Non-**UPS** products are sometimes declared as dependencies as well; for instance, **gcc** is not installed under **UPS** on some machines, but is a dependency of some **UPS** products. End users don't normally need to know what dependencies a particular product has, as long as the product runs without problems.

We distinguish two categories of dependencies: those which are required for the main product to function, and optional ones which generally enable nonessential features of the main product.

The coupling of products with their dependencies facilitates product setup (setup is described in section 3.4 *Setting up a Product*). You need only setup a single **UPS** product to access any and all of the products listed as dependencies for that product.

Multiple levels of dependencies are possible. As an example, the mail product **exmh** has several "first level" dependencies, one of which is **www**. **www** has dependencies of its own, which in turn may have dependencies, and so on. These are all referred to as "lower level" dependencies of **exmh**.

UPS product dependencies can exist across databases. For this to work, the databases in which they are declared must all be included in `$PRODUCTS`, or in a database list specified on the command line.

A note about what used to be called *build dependencies*: These don't exist as such as of **UPS** v4. The qualifier "BUILD" or "build" now provides essentially the same function as the old build dependencies. There is an example in the reference section 23.1 *setup* that illustrates the use of this qualifier.

2.3.7 Product Overlays

An additional class of required product is supported by **UPD**, called an *overlaid product*. An overlaid product gets distributed and maintained in the product root directory of its main product. For example, the overlaid products **cern_bin**, **cern_ups**, **cern_lib**, etc., all reside in the product root directory for the main product **cern**. A patch is another good example of the use of overlaid products. The set of products overlaid on a main product is collectively referred to as *the overlay*. An overlaid product is not precluded from being a dependency of other **UPS** products.

2.4 UPS Database Overview

The information **UPS** needs for managing products is maintained in a database. A **UPS** database is a directory tree which contains a subdirectory for each product declared to the database, the subdirectory having the same name as the product. It also usually contains the hidden subdirectories `.upsfiles` with **UPS** database configuration information, and `.updfiles` with **UPD** configuration information.

Within each product-specific directory under the **UPS** database directory, a set of ASCII files collectively contains the **UPS** management information for the product on that system. We call these files *UPS database files*.

UPS commands refer to the database directory via the **UPS** environment variable `$PRODUCTS` described in section 2.7.2, or the `-z <databaseList>` option. `$PRODUCTS` can be set to point to one or 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).

2.4.1 UPS Database Files

There are two types of **UPS** database files for each product: version and chain files.

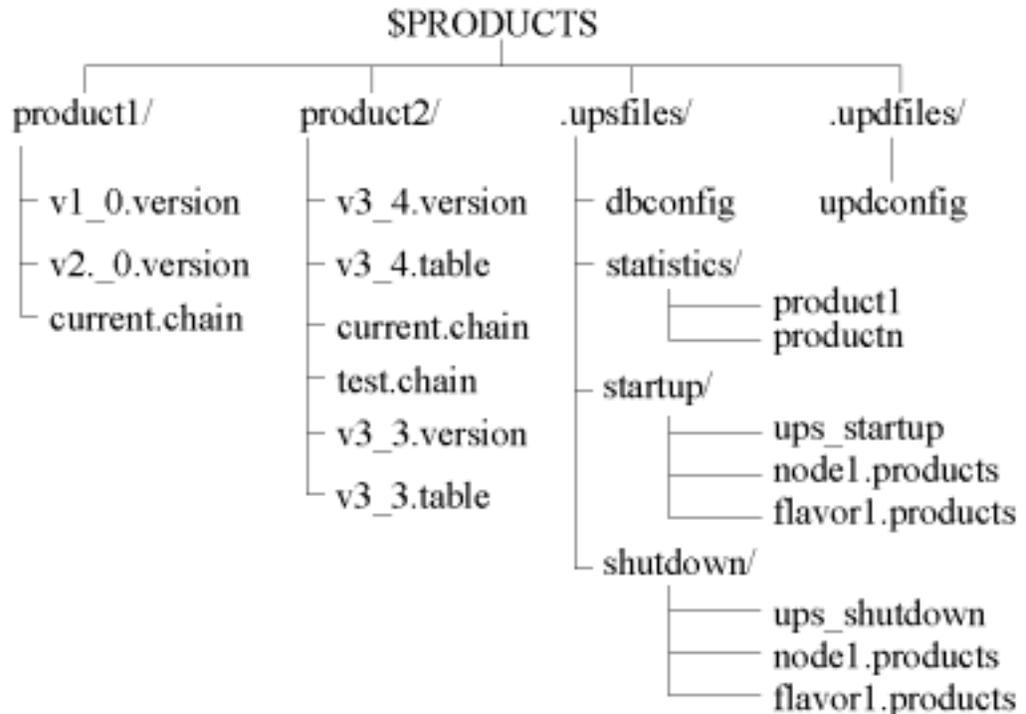
- *Version files* tell **UPS** where to find all the files associated with a particular version of a product on the local system, and contain some other system-specific information. They are generally named according to the scheme `vx_y.version`, e.g., `v1_0.version`. These are described in Chapter 29: *Version Files*.
- *Chain files* are optional and contain pointers to version files, thus providing convenient access to particular product versions on the local system. They are generally named according to the scheme `chainname.chain`, e.g., `current.chain`. These are described in Chapter 30: *Chain Files*.

Another file that strictly speaking is not a database file, but is also used in product management is a *table file*. Table files contain information which is independent of any local installation and which is specific to one or more particular instances of the product. For example, table files tell **UPS** what needs to be done to configure a product or to make it accessible for use. Table files are provided by the product developer when needed. Not all products have table files. And conversely, some products consist only of a table file. Table files are described in Chapter 36: *Table Files*.



Whereas version and chain files are constrained to reside under the **UPS** database, table files can reside anywhere. Table files are usually kept either in the database or within the product instance's root directory structure. Table file paths are indicated in version files, and they are thereby accessible to **UPS** regardless of location.

2.4.2 UPS Database Structure



2.5 Using UPS Without a Database

It bears mentioning that **UPS** can be installed on a machine to manage products without a **UPS** database, albeit in a limited way. (Except where noted, this manual is written with the assumption that **UPS** is used with a database.) In the absence of a database, there are no database files and every **UPS/UPD** command must include all of the information that normally would have been read from a database. In particular, all commands require specification of the table file name, and usually its location. Functionality requiring or operating on a **UPS** database is not supported when **UPS** is implemented in this way.

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. Product developers may also work in this type of environment, especially if they're using **CVS** or another code management product; this is described in section 16.2.2 *CVS* under 16.2 *Tools for Developing and/or Packaging Products*.

2.6 UPS and UPD Commands

Please see Part VI *UPS and UPD Command Reference* of this manual for complete information on **UPS** and **UPD** commands. To get you started, we describe briefly here the command syntax and defaults used.

2.6.1 Syntax

Most **UPS** and **UPD** commands are of the form **ups <command>** or **upd <command>** (the exceptions are **setup** and **unsetup**), and take a variety of command line options and arguments. Multiple arguments must be separated by colons (:). The standard syntax is:

```
% ups <command> [<options>] <product> [<version>]
```

For example:

```
% ups list -f IRIX+6:OSF1+V3 xemacs v20_4
```

The first occurring unflagged argument on the line after the command is generally interpreted as the product name¹, and the next (if any) is interpreted as the version. With that limitation, the name, version and options can occur anywhere on the command line.

2.6.2 Defaults

If no database is given, **UPS** uses \$PRODUCTS to determine the database. If no instance-identifying options or version are given, **UPS/UPD** operates on the instance declared as current for the flavor of the machine on which the command is issued (to the highest specification level possible). If there is no instance declared as current or if the current instance has any qualifiers, then the default instance matching will fail.

For **UPD** commands, if no product distribution node is specified, **UPD** uses the central Computing Division product server *fnkits.fnal.gov* as the default. (The product distribution database on this node is known as KITS.)

1. An exception is the **<componentList>** element in the **upd update** command, documented in section 24.13 *upd update*.

2.7 The UPS Environment

2.7.1 Initializing the UPS Environment

In order to access and use **UPS** products or manipulate a **UPS** database, your environment must be initialized to make the **UPS** commands available to you.

Systems using Fermi UNIX Environment

Many on-site Fermilab systems, and some off-site nodes as well, have been configured with the Fermi UNIX Environment (FUE).¹ If your system runs FUE, your **UPS** environment gets initialized automatically when you log in.



The **UPS** initialization does not carry over to any scripts which either create a new login process or invoke a shell from the “other” shell family. If such a script needs to setup and run **UPS** products from that new process, you need to include a line in the script which sources the appropriate `setups.[c]sh` file, as described below.

Systems NOT using Fermi UNIX Environment

If your system does not run FUE, you will need to initialize your **UPS** environment yourself unless your system administrator has taken care of this. The **UPS** initialization is accomplished by sourcing a **UPS**-provided script; namely, one of the following (depending on your login shell):

Bourne shell family	<code>setups.sh</code>
C shell family	<code>setups.csh</code>

It must be sourced from the directory where the **UPS** setup files have been installed on your system. Virtually all supported systems on-site provide “courtesy links” in `/usr/local/etc` so that you don’t need to know exactly where the `setups.[c]sh` files actually reside². If the `setups.[c]sh` files are *not* in `/usr/local/etc` and your system doesn’t maintain these courtesy links, you will need to ask your system administrator or **UPS** database maintainer where to find these files.

You can either source the `setups.[c]sh` scripts manually (for occasional use), or you can add the following to your `.[c]shrc` file so that **UPS** gets initialized every time you log in:

Bourne shell family	<code>. /path/to/setups.sh</code>
---------------------	-----------------------------------

1. FUE was updated in the fall of 1999. It requires **UPS/UPD** v4_0 or higher and **systemtools** v6_0 or higher plus dependencies.

2. There are some exceptions; including the system where **UPS** development takes place.

```
C shell family          source /path/to/setups.csh
```

You also need to include this line in any other scripts which setup and invoke **UPS** products, once per login session and any time a shell from the other shell family is invoked.

2.7.2 Changes UPS Makes to your Environment

The following environment variables get set/modified by `setups.[c]sh`:

<code>\$PRODUCTS</code>	If only one UPS database is defined, this points to it; if two or more are defined, this variable can be set as the colon-separated ¹ list of UPS databases. The order of the databases in this list reflects the order of precedence for accessing products.
<code>\$UPS_DIR</code>	This points to the top level directory (called the product root directory) of the active instance of UPS .
<code>\$UPS_SHELL</code>	Set to <code>sh</code> or <code>csh</code> , depending on shell family in use.
<code>\$PATH</code>	Modified to include <code>\$UPS_DIR/bin</code> .
<code>\$SETUP_UPS</code>	a string containing all the information that the unsetup command needs in order to identify the active instance of UPS (e.g., <code>ups v4_5_2 -f SunOS+5 -z /path/to/db</code>)

2.7.3 Changes UPD Makes to your Environment

The following environment variables get set/modified by the `setup upd` command.

<code>\$FTP_PASSIVE</code>	allows UPD to work behind certain firewalls
<code>\$SETUP_UPD</code>	a string containing all the information that the unsetup command needs in order to identify the active instance of UPD
<code>\$PATH</code>	modified to include <code>\$UPD_DIR/bin</code> .
<code>\$UPD_USERCODE_DIR</code>	directory containing UPD configuration
<code>\$UPD_USERCODE_DB</code>	database containing UPD configuration

1. Using whitespace as a separator in place of colons is allowed in `$PRODUCTS` for backwards compatibility. However, colons are recommended. Using colons is more consistent with the new command format that requires multiple option arguments to be separated on the command line with colons.

