

Chapter 16: UPS Product Development:

General Considerations

This chapter discusses the **UPS** product development methodology and tools that can be used in product development. It also provides recommendations for organizing your local product development area and the individual product root directories you will need to create.

16.1 Product Development Considerations and Recommendations

In this section we will provide some guidelines for product development as it affects the product's inclusion in the **UPS** framework.

Simple scripts which run on any architecture are naturally quite straightforward to implement under **UPS**. Products which are obtained from the outside world (third party) as executable images with no source code are also generally straightforward. The ones that get complicated are the products which must be compiled and/or otherwise built for each and every supported architecture.

16.1.1 All Products (Locally Developed and Third Party)

Shell Independence

The product should run the same way under both shell families, **sh** and **csh**. If the product requires any actions to take place before it will run (e.g., its `bin` directory added to your `$PATH`, some environment variables set), provide a table file containing these actions. The **UPS** environment is described in Chapter 2: *Overview of UPS, UPD and UPP v4* and table files in Chapter 36: *Table Files*. The functions supported in table files are designed to work in a shell-independent manner, in general.

Flavor Declaration in UPS

On your development system, we recommend that you declare your products according to the *fully specified flavor* of the machine on which you build them (or on which they were built). We consider this to be very important, especially if your target systems contain or will ever contain mixed OS releases (e.g., IRIX+6.2 and IRIX+6.5). This will help to avoid problems when a new OS release doesn't run images built on an older one, or vice-versa. You don't want to have to go back and comb out which OS release a particular product instance was built for, you want to be able to tell immediately from looking at the database. Installers and users also need this information to facilitate their database maintenance.

Products which have no flavor-dependence at all (shell scripts, for instance), should be declared as NULL to the database (use the "zero" option, `-0`; see Chapter 25: *Generic Command Option Descriptions*). For other products, include the entire flavor string of the build platform in the declaration, or the major portion of that string. For example, if you build on an OSF1 machine running V3.2, declare your products with the flavor OSF1+V3.2 or OSF1+V3 rather than just OSF1 (e.g., use the flavor level corresponding to the options `-3` or `-2` rather than `-1`)

Policy Regarding Use of /usr/local/bin



Outside Fermilab, in the UNIX world at large, products typically get put in `/usr/local/bin`. With this in the user's `$PATH`, all the products are accessible. This practice is **inconsistent** with the goal of **UPS** to provide concurrent versions of products. Therefore *only* products specially approved by the FUE working group may write into `/usr/local/bin`. **No other products should write to this area, or to any other area within `/usr/local`.**

16.1.2 Products that You Develop

If you're writing your own product for implementation within **UPS**, you have the luxury (and, we might add, the *responsibility*) of creating it such that it exploits the important features of **UPS**, thus making it easy for the user to install and run, and easy for you or another developer to maintain in the future. We urge you to follow the guidelines we present here.

Self-Containment and Location Determination

First, design the product such that it is self-contained. It should identify its location and the location of any required files *at run time* (as opposed to compile time). You as the product developer have total control over the structure and contents of the product root directory, but no control at all over where the product root directory will reside on a target system.



If you write the product such that it calculates its location at compile time, you'll be putting the hard-coded path to your development environment into the image -- most likely *not* the correct path on the user machine.

You can choose to define the environment variable `$<PRODUCT>_DIR`, which points to the product root directory. In **UPS** v4, this variable is no longer always necessary since much of its usefulness is taken over by the local read-only variable `${UPS_PROD_DIR}`, described in section 35.6 *Local Read-Only Variables Available to Functions*. However, users will still find `$<PRODUCT>_DIR` to be useful since they will have access to it as long as the product is setup.

As an example of the use of `${UPS_PROD_DIR}`, take **myproduct** written in **perl** which requires the file `lib/myprod-headers.pl`. You should refer to this file in the **perl** code as

```
$ENV:: {MYPROD_PERL_LIB} /myprod-headers.pl rather than by  
its full path, e.g., /path/to/lib/myprod-headers.pl. In the table  
file, set MYPROD_PERL_LIB to ${UPS_PROD_DIR} /lib. You should  
make no assumptions about where users will put the file.
```



As stated above, products should *not* use or copy files into the areas under `/usr/local`.

Reproducible Build Procedure

All products should be built using a build script in order to ensure that the build procedure is reproducible. If your product is at all complex, we recommend that you use Makefiles for this purpose. We have created a template product for creating **UPS** products, described in Chapter 19: *Using template_product to Build and Distribute UPS Products*. It includes Fermi-standard Makefiles, and automates much of the process. The general UNIX **make** utility and the associated Makefiles are beyond the scope of this document, but the subject is introduced in *UNIX at Fermilab*, and treated in many standard UNIX texts.

System Independence

The various flavors of UNIX have many differences. You will generally have to release separate instances of your (compiled) products for the different flavors. However, the more you are able to insulate your product from flavor/release dependencies, the easier your product will be to maintain, and the less rigid it will appear to installers and users.

16.1.3 Third-Party Products Requiring a Hard-Coded Path

If you're installing a third-party product, downloaded from the Web or elsewhere, you may not have the opportunity to code it such that it identifies its location at run time based on `${UPS_PROD_DIR}` or the `$<PRODUCT>_DIR` environment variable. Whereas many products never need to know their location (they only need to be in your `$PATH`, for example), many other products *do* need to know their location in order to locate auxiliary commands, libraries, utilities, and so on.

Techniques for Implementing these Products

For those that do, the technical note TN0086 *Use of "/usr/local/products" now deprecated*, on-line at <http://www.fnal.gov/docs/TN/TN0086/tn0086.html>, describes recommended techniques for implementing the products. Please refer to it for information. The three approaches it describes are, briefly:

- For a product that is already setup and which contains a script that requires an interpreter, start the script with `#!/usr/bin/env <interpreter>` (e.g., `#!/usr/bin/env perl`). The `env` program will run the first copy of the interpreter it finds on your command search path, and your script is then executable.
- Create a “wrapper” shell script which sets up the **UPS** environment, sets up your product, and then invokes the appropriate commands. (An example is `www v2_6a`.)
- Sometimes getting a product setup before one of its scripts is invoked is not practical, and wrapper scripts may be unacceptably slow to start up. In cases where the product is considered important enough by the FUE working group that it must work properly even in the absence of **UPS**, a “trampoline” executable is provided, usually in `usr/local/bin`. The wrapper script should contain `#!/path/to/trampoline`.

When the product is configured, its `CONFIGURE` action inserts the product path into the trampoline executable. The wrapper script is then executable. Note that these products generally need to be declared as *root*.



In the past for **UPS v3** we used the `/usr/local/products` convention. We include this information for reference purposes only. This convention had serious drawbacks. The old (now deprecated) procedure was standard only on fully FUE-compliant systems (defined in the on-line document DR0009), and required that you:

- configure, build, and/or (re-)code the product so that the hard-coded path it uses is
`/usr/local/products/{product}/{version}` (e.g.,
`/usr/local/products/tk/v4_2a`).
- write a `configure` script which creates the directory `/usr/local/products/{product}`, and creates in it the symbolic link `{version}` back to the *real* product root directory (e.g., `/usr/local/products/tk/v4_2a` is a symbolic link to `/path/to/products/OSF1+V3/tk/v4_2a`).
- write a `current` script that creates a symbolic link called `current` in the same directory, pointing to the link for the instance which is declared as `current` (e.g.,
`/usr/local/products/tk/current` is a symbolic link to
`/usr/local/products/tk/v4_2a`).

Examples of Products Requiring Hard-coded Paths

Here are examples of situations in which hard-coded paths are unavoidable:

- Pre-built products which have hard-coded paths.
- Products that you can rebuild, but which were not coded with the idea of calculating where the files sit at run time. You need to tell them where to look for files at compile time, and this leads to hard-coded paths in the images.¹
- Commands that are not executed in the context of a shell, but rather as a program. An example is the **mh** utility **slocal** (for automatically sorting and foldering your incoming **mh** mail). This command is called via a command line in one of the configuration files (`.forward`).

1. Most vendors (freeware, shareware, and the few paid packages where you get the source code and rebuild it) now make it possible to modify the Makefiles so that you can decide where you want the output files, images, and so on, to go. Unfortunately, these are still frequently hard-coded at compile time, not run time. Therefore, packages that you build in this manner on your development system will not be right when installed on a user system with a different product root directory path.

You can't use the construct `" | ${MH_DIR}/lib/slocal -user joe"` to identify **slocal** because the program running this command will not expand the `${MH_DIR}` environment variable. You also don't want to spell out the whole actual path because you'd have to edit the `.forward` file every time a new version of **mh** is released.

- **cgi** scripts, **rsh** scripts and other situations in which you can't be sure that the product will necessarily have been setup when it is called by another one, and it needs to work anyway. We recommend that you consult with the UAS group (uas-group@fnal.gov) to determine the best course of action. Frequently you can create product configuration scripts that copy or link the product files into the correct location on the target node. In some cases for **cgi** scripts, you can have your Web server setup the product and pass the relevant environment variables.

16.2 Tools for Developing and/or Packaging Products

The tools that we introduce in this section can be used separately or together. They are all available as **UPS** products in **KITS**. See the on-line documentation *Integrating buildmanager, cvs, template_product, and upd* at <http://www.fnal.gov/docs/products/buildmanager/Integrating.html>.

16.2.1 Buildmanager

The **buildmanager** application is a configurable tool which lets you build software on multiple systems simultaneously, in an organized and consistent fashion. It allows you to set up standardized build sequences and define actions to be performed automatically. It can stop if things go wrong, and allows interaction with various build systems to correct problems. It is available as a **UPS** product in **KITS**. Any system to which you can telnet and run commands can be used as a build system with **buildmanager**. See the on-line documentation at <http://www.fnal.gov/docs/products/buildmanager/>.

16.2.2 CVS

It is a common practice to maintain a product's source code as well as its Makefile and **UPS** management files in a **CVS** repository for development and maintenance. **CVS** allows each developer to check out files into a private working directory and to modify them as necessary. With **CVS** you can

maintain all the different versions and flavors in a single work area, and you can pull them out to the separate nodes as needed. Developers working with prebuilt binaries (downloaded from the Web or purchased from a vendor) can use **CVS** for just the Makefile and **UPS** management files (e.g., the local `README` and `INSTALL` files, the table file, tests, documentation, and so on) so that they can be properly source-controlled. Documentation for **CVS** can be found on-line at <http://www.fnal.gov/docs/products/cvs/>.

It is useful to be able to use **UPS** to setup these checked-out areas. One way that this can be accomplished is by declaring the checked-out area to either the main or a private **UPS** database, but this is often cumbersome, as these checked-out areas are by nature fairly transient.

A better solution is to exploit the **UPS** capability of setting up a product instance without having it actually declared to any database. To do this, you simply need to supply the `setup` command with all of the necessary information, shown here:

```
% setup <product> -r /your/checked/out/area -M <tableFileDir> \  
-m <tableFile> -q <qualifierList> -f <flavor>
```

16.2.3 Template_product

To simplify and somewhat automate the process of building **UPS** products, we have designed the product `template_product`. Once this product is installed on your system, it can be cloned into a new product area and customized to the new product. `template_product` can be used to build products of all types (shell script, pre-built binary, source code). We discuss this product in detail in Chapter 19: *Using template_product to Build and Distribute UPS Products*.

16.3 Directory Structure for a UPS Product Instance

The top level directory of a **UPS** product instance is called the *product root directory*, and in general it should contain files and subdirectories in which almost everything related to the product instance resides: the executables, the library files, the documentation, and so on. The `ups` directory files (i.e., the **UPS** metadata) and the table file usually reside here, but are not required to do so.

UPS is very lenient in the directory structure it allows. Nothing is required in all situations beyond a product root directory. Normally product instances have a table file containing actions that are run during operations like product installation and setup.

We recommend that you follow a few directory structure guidelines simply to conform to a generally recognized format. This will make it easy for yourself and others to identify each file and directory later on. The following is a relatively complete sample directory structure underneath the product root directory. Most products won't require all of these elements. On the other hand, you may include other directories and/or files not listed here. Elements which we *strongly recommend* that you provide (in addition to the executables) for every product include a `README` file, man pages, a user guide, test scripts and example files.

<code>README</code>	text file containing information such as origin of the product (by whom, from where, etc.), support level, support group/person, caveats and known bugs (may be contained in the <code>ups</code> directory)
<code>bin</code>	directory containing the executable(s)
<code>ups</code>	directory containing metadata files and other executable and data files used during implementation and invocation; may also contain <code>INSTALL_NOTE</code> (described below) file and the directories <code>toman</code> , <code>toInfo</code> , <code>tonews</code> and <code>tohtml</code> . Often the table file resides here. (This directory is no longer a required element of a UPS product.) Default location of the <code>ups</code> directory is directly underneath the product root (for compatibility with UPS v3), but it may reside anywhere.
<code>ups/INSTALL_NOTE</code>	text file containing a detailed description of any installation actions that are more easily performed directly by the installer rather than by a script (beyond or instead of running <code>configure</code> and/or <code>tailor</code> and/or <code>current</code>). This should not be a script. This file is not usually needed. If provided, mention it in the <code>README</code> file so that product installers know to run it.
<code>lib</code>	directory containing libraries
<code>src</code>	directory containing source code
<code>include</code>	directory containing include files
<code>doc</code>	directory containing a user guide and any other documentation as appropriate; should include the source files (e.g., LaTeX, Word) as well as the printable files (e.g., PostScript)
<code>man</code>	directory containing unformatted man pages. The files get copied into the location specified in <code>\$PRODUCTS/.upsfiles/dbconfig</code> (keyword <code>MAN_TARGET_DIR</code>).

	Default location (for compatibility with UPS v3): ups/toman/man
catman	directory containing formatted man pages. The files get copied into the location specified in \$PRODUCTS/.upsfiles/dbconfig (keyword CATMAN_TARGET_DIR).
	Default location (for compatibility with UPS v3): ups/toman/catman
html	directory containing the html version of the user guide and any other documentation as appropriate (automatic copy of files to standard area defined by HTML_TARGET_DIR not implemented in UPS v4)
	Default location (for compatibility with UPS v3): ups/tohtml
news	directory containing news files to be posted to a newsgroup (automatic copy of files to standard area defined by NEWS_TARGET_DIR not implemented in UPS v4)
	Default location (for compatibility with UPS v3): ups/tonews
Info	directory containing any text files that are to be displayed as a login announcement via the Info feature. The files get copied into the location specified in \$PRODUCTS/.upsfiles/dbconfig (keyword INFO_TARGET_DIR). Info is generally used to communicate to users about the Fermilab computing systems events, (e.g., shutdowns), software upgrades and other systems-related information.
	Default location (for compatibility with UPS v3): ups/toInfo
test	directory containing test scripts and any other test-related files
examples	directory containing example files to help users learn how to use the product

Product Documentation Storage

Each different type of product documentation (e.g., man pages, html files, PostScript files, and so on) must reside in a separate subdirectory. The subdirectories usually reside under the product root directory, but do not have to. In the product's table file, you should use the keywords

XXX_SOURCE_DIR as listed in section 28.4 *List of Supported Keywords* (e.g., MAN_SOURCE_DIR, INFO_SOURCE_DIR) to identify the directory in which each form of documentation is maintained. For example:

```
CATMAN_SOURCE_DIR = ${UPS_PROD_DIR}/catman
```

```
MAN_SOURCE_DIR = ${UPS_PROD_DIR}/man
```

```
INFO_SOURCE_DIR = ${UPS_PROD_DIR}/Info
```

UPS currently requires that all files in a directory specified by XXX_SOURCE_DIR be of the corresponding file type; you cannot mix file types.

The `ups/toman` directory and its subdirectories `man`, `catman` and `toInfo` are used as defaults in **UPS** v4 for backwards compatibility. This structure is not necessarily our recommendation. If a product comes with `man` pages, `Info` files, `html` files and so on, we recommend that you leave them where they are, and simply specify their locations in the table file. If you are writing your own, you can put them in subdirectories directly under the product root directory, which is generally the most convenient place.

In **UPS** v4, NEWS_SOURCE_DIR and HTML_SOURCE_DIR are not implemented.