

Chapter 11: Maintaining a UPS Database

In this chapter we assume that you have **UPS/UPD** installed and that you have a working database and products area. We provide instructions and examples for performing the following functions:

- declaring product instances to a database
- declaring, removing and changing chains
- removing product instances
- verifying the integrity of a product instance
- modifying information in a database file
- determining if a product needs to be updated
- updating a table file or `ups` directory
- retrieving an individual file from a distribution node
- checking product accessibility
- troubleshooting



To get command usage information or on-line help, use the following resources:

- Refer to Part VI of this guide (in GU0014A), *Command Reference*, especially Chapter 23: *UPS Command Reference*.
- Run the command with `"-?"`, e.g., `ups declare "-?"`¹.
- Man pages are also provided; use an underscore with the **UPS** command when running `man`, e.g., `man ups_declare`.

11.1 Declare an Instance

A product instance must exist on the system before it can be declared to a **UPS** database². Product declaration is done with the `ups declare` command. Declaring a product instance makes it known to **UPS**, and therefore retrievable

-
1. The double quotes are necessary for C shell users; `-?` is interpreted by `sh`.
 2. At least a rudimentary root directory hierarchy for the product, its table file directory and table file must exist before declaration.

within the **UPS** framework. Normally products are installed on user nodes using the **upd install** command which, in addition to downloading and installing the product, runs **ups declare** to make the initial declaration of the product to the local **UPS** database. If you use **FTP** to download a product, then you'll need to declare it manually. Refer to Chapter 8: *Installing Products using FTP* for details about installing with **FTP**.

If you use **upd install** and you have more than one database, refer to section 6.3 *How UPD Selects the Database* to see how **UPD** determines the database for the declaration.

11.1.1 The ups declare Command

Before declaring, make sure the product is unwound into its final location. Also make sure that you've downloaded the table file and installed it in an appropriate directory. For an initial declaration you must specify at a minimum: the product name, product version, product root directory, flavor and table file name¹.

The full command description and option list is in the reference section 23.5 *ups declare*. Here we show commonly used command options (see the notes regarding **-z**, **-U** and **-M** which follow):

```
% ups declare <product> <version> -r /path/to/prod/root/dir/ \  
-f <flavor> [-z /path/to/database] [-U /path/to/ups/dir] \  
[-m <table_name>.table] [-M /path/to/table/file/dir] \  
[<chainFlag>]
```

- 1) If the database is not specified using **-z**, **UPS** declares the product into the first listed database in \$PRODUCTS (see section 27.1 *Database Selection Algorithm* for more information).
- 2) If the product's `ups` directory tar file was unwound in the default location (`$(<PRODUCT>_DIR/ups)`), then **-U** `/path/to/ups/dir` is not needed. If the `ups` directory is located elsewhere (or named differently), this specification must be included. If specified as a *relative* path, it is taken as relative to the product root directory.
- 3) If the product's table file was placed in either of the two default locations (under `/path/to/database/<product>/` or in the product's `ups` directory), then **-M** `/path/to/table/file/dir` is not needed. Only use the **-M** option if you have moved the table file to a separate location where **UPS** won't otherwise find it. If specified as a *relative* path, it is taken as relative to the product root directory. See

1. Two exceptions: (1) if the product consists only of a table file that sets up a list of dependencies, there is no product root directory; and (2) if the product has no table file (very rare) then there is no table file name.



section 29.4 *Determination of ups Directory and Table File Locations* for details on how **UPS** finds the table file.

Unless the product you're declaring has no table file (true for very few products), make sure its location gets declared properly, either explicitly or by default. Otherwise, users will need to specify its name and location on the command line every time they want to run or operate on the product. If it is neither declared nor specified on the command line, **UPS/UPD** assumes there is no table file.

You can opt to declare a chain to the product instance at this time or in a later declaration. To declare a chain, include the appropriate chain flag in the command (see section 2.3.5 *Chains* for a listing). This is described in section 11.2 *Declare a Chain*.

11.1.2 Examples

Additional examples are included in the reference section 23.5 *ups declare*.

Declaration of New Product to Non-default Database

The following command shows a fairly typical product declaration. We'll install a product called **histo** v4_0 onto a SunOS+5 node. We assume the product instance's `ups` directory is maintained under its product root directory, and that it contains the table file. We include the `-z` option to indicate that we want to override the default database selection. This is the first instance of this product to be declared to this database, therefore the **ups declare** command automatically creates the appropriate product directory under the specified database:

```
% ups declare histo v4_0 -f SunOS+5 -m histo.table -z $MY_DB -r\  
/path/to/products/SunOS+5/histo/v4_0
```

We can run a **ups list -l** command to see all the declaration information (include `-a` because it's not yet declared current):

```
% ups list -alz $MY_DB histo  
  
DATABASE=/path/to/ups_database/declared  
Product=histo Version=v4_0 Flavor=SunOS+5  
Qualifiers="" Chain=""  
Declared="1998-04-17 22.08.30 GMT"  
Declarer="aheavey"  
Modified="1998-04-17 22.08.30 GMT"  
Modifier="aheavey"  
Home=/path/to/products/SunOS+5/histo/v4_0  
No Compile Directive  
Authorized, Nodes=*  
UPS_Dir="ups"
```

```

Table_Dir=""
Table_File="v4_0.table"
Archive_File=""
Description=""
Action=setup
        prodDir()
        setupEnv()

addalias(histo,${UPS_PROD_DIR}/bin/histo)

addalias(hsdir,${UPS_PROD_DIR}/bin/hsdir)

envSet(HISTO_INC,${UPS_PROD_DIR}/include)

```

Declaration of Additional Instance of a Product

In the following example we declare an additional instance of **histo**, of the same version, but for the flavor **IRIX+5**. Again the table file resides under the product root directory's `ups` subdirectory, and we override the default database. This time we declare it with the chain "test" (**-t**):

```
% ups declare histo v4_0 -tf IRIX+5 -m histo.table -z $MY_DB -r\
/path/to/products/IRIX+5/histo/v4_0
```

Running a **ups list -a** to see what the database now contains for this product, we find:

```
% ups list -az $MY_DB histo
      DATABASE=/path/to/ups_database/declared
      Product=histo  Version=v4_0  Flavor=SunOS+5
      Qualifiers=""  Chain=""

      Product=histo  Version=v4_0  Flavor=IRIX+5
      Qualifiers=""  Chain=test
```

Declaration with Table File Located in Database

Depending on your configuration, you may want the table file to reside in the product's subdirectory under the database (e.g., `$PRODUCTS/<product>/<table_file>`).



A table file for the product must be placed in its permanent location before the instance is declared to the database. Therefore, if you are declaring the first instance of a product to the database, you need to manually create the product directory under the database and copy the table file into it before declaring the instance.

You still do not need to specify the table file location (**-M** option) on the **ups declare** command line; **UPS** will find it here.

11.2 Declare a Chain

Chains are described briefly in section 2.3.5 *Chains*, and in detail in Chapter 30: *Chain Files*. A chain can be declared when the product instance is initially declared to the database (see section 11.1 *Declare an Instance*), or at a later time.

11.2.1 The ups declare Command with Chain Specification

To add a chain to a product instance, use the **ups declare** command with a **chainFlag** option. The **chainFlag** option can be one of the standard ones: **-c**, **-d**, **-n**, **-o**, or **-t**. **chainFlag** can also be replaced by **-g chainName**, where **chainName** is either one of the standard chain names, e.g., **-g current**, or a user-defined one. The full command description and option list is in section 23.5 *ups declare*. Here are some examples:

```
% ups declare -c [<other options>] <product> <version>
```

```
% ups declare -g current [<other options>] <product> <version>
% ups declare -g my_chain [<other options>] <product> <version>
```

Declaring a chain is generally allowed on any node of a cluster, however if the corresponding chain action in the table file includes any node-specific or flavor-specific functions,¹ we strongly recommend that you declare the chain from that node, or from a node of that flavor to avoid mismatches. This should be noted in the `INSTALL_NOTE` file if it's necessary.

To include a chain in the initial declaration, simply add a chain option to the instance declaration as described in section 11.1 *Declare an Instance*. To add a chain to a previously declared product instance, include only the options required to identify the product instance and the chain option, e.g.,:

```
% ups declare -c <product> <version> [-f <flavor>] \
  [-z <database>]
```

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 flavor/qualifier pair to the same chain, the pre-existing chain will be removed automatically. In other words, **UPS** ensures that a chain for a particular flavor/qualifier pair is unique.

A couple of examples will help to clarify how this works. In these examples we assume that the product instance has previously 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 11.1 *Declare an Instance* for examples specific to that operation.

11.2.2 Examples

Declare an Instance to the Database as test

In a typical situation, a product instance is initially declared as test (`-t`) to the default database, to be made current at a later date. In this example, we make an initial declaration as “test” of the product **histo** version `v4_0`, flavor `IRIX+5`, located in `/usr/products/IRIX+5/histo/v4_0`, with the table file name `v4_0.table`:

```
% ups declare -tr /usr/products/IRIX+5/histo/v4_0 -f IRIX+5 \
  -m v4_0.table histo v4_0
```

We verify the declaration using `ups list -l -a`:

```
% ups list -la histo -f IRIX+5

DATABASE=/path/to/ups_database/declared
```

1. Actions are described in Chapter 34: *Actions and ACTION Keyword Values*, functions in Chapter 35: *Functions used in Actions*, and table files in Chapter 36: *Table Files*.

```

        Product=histo   Version=v4_0   Flavor=IRIX+5
        Qualifiers=""   Chains=test
        Declared="1998-04-17 22.27.16 GMT:1998-04-17
22.27.16 GMT:1998-
        Declarer="aheavey:aheavey"
        Modified="1998-04-17 22.27.16 GMT:1998-04-17
22.27.16 GMT:1998-
        Modifier="aheavey:aheavey"
        Home=/path/to/products/IRIX+5/histo/0
        No Compile Directive
        Authorized, Nodes=*
        UPS_Dir="ups"
        Table_Dir=""
        Table_File="v4_0.table"
        Archive_File=""
        Description=""
        Action=setup
                prodDir()
                setupEnv()

```

```
addalias(histobin,${UPS_PROD_DIR}/bin/histobin)
```

```
addalias(hsdirbin,${UPS_PROD_DIR}/bin/hsdirbin)
```

```
envSet(HISTO_INC,${UPS_PROD_DIR}/include)
```

Notice that **DECLARED**, **DECLARER**, **MODIFIED** and **MODIFIER** all have two values. The first value is for the declaration to the database, the second is for the test chain declaration. In the following example, you will see that these fields acquire a third value when the chain is changed.

Change instance from test to current

Once testing is complete and successful, you will want to take the product instance out of test and declare it as current. For the product instance of the previous example, we issue the command:

```
% ups declare -c histo v4_0 -f IRIX+5
```

This adds the current chain, but it does not remove or modify the test chain. (To remove the test chain, see the instructions in section 11.3 *Remove a Chain*.)

Verify using **ups list**:

```
% ups list -a histo -f IRIX+5
```

```

        Product=histo   Version=v4_0   Flavor=IRIX+5
        Qualifiers=""   Chains=test,current

```

If we use the long form, we see the additional declaration and modification userid and time (output edited for brevity):

```
% ups list -la histo -f IRIX+5
    DATABASE=/path/to/ups_database/declared
        Product=histo   Version=v4_0   Flavor=IRIX+5
            Qualifiers=""   Chains=test,current
                Declared="1998-04-17 22.27.16 GMT:1998-04-17
22.27.16 GMT:1998-04-18
22.00.16 GMT
                    Declarer="aheavey:aheavey:aheavey"
                        Modified="1998-04-17 22.27.16 GMT:1998-04-17
22.27.16 GMT:1998-04-18
22.00.16 GMT
                            Modifier="aheavey:aheavey:aheavey"
                                ...
```

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 and you want to declare a different version of the product as current for the same flavor. We'll use the previous example **histo v4_0**, and declare version **v4_1** as current:

```
% ups declare -c histo v4_1 -f IRIX+5
```

The previously current instance for this flavor/qualifier pair now has no current chain. Any other chains it may have had (test, in this case) remain unchanged.

11.3 Remove a Chain

To remove a chain from a product instance, you can use the **ups undeclare** command, or you can simply remove the chain file, or the portion of it that relates to the instance in question. It is usually easier and less error-prone to use the **ups undeclare** command. The full command description and option list is in section 23.19 *ups undeclare*.

The **ups undeclare** command has a simple syntax for removing chains:

```
% ups undeclare <chainFlag> <product> [-f <flavor>] \  
  [<other options>]
```



Do not include the version in the command; it is incompatible with including the chain, and may result in removing the product declaration! We recommend always including the **-f <flavor>** option if you have a multi-flavored database.

As an example, let's remove the current chain from the current instance of **ximagetools**. Running **ups list** before and after, we should see the current chain disappear:

```
% ups list -K+ ximagetools  
  "ximagetools" "v4_0" "NULL" "" "current"
```

```
% ups undeclare -c ximagetools -f NULL
```

```
% ups list -aK+ ximagetools
```

```
  "ximagetools" "v4_0" "NULL" "" ""
```



If multiple flavor/qualifier pairs have the same chain and thus share the chain file in question (in which case you *must* specify the flavor/qualifier information on the command line), only the portion of the file relating to the specified instance will get removed; the file itself will not be deleted.

11.4 Change a Chain

In general, changing the chain to a product instance requires removing the pre-existing chain (see section 11.3 *Remove a Chain*) and adding a new one (see section 11.2 *Declare a Chain*). There is no way to directly change a chain.

When a current instance of a product already exists, if you declare a new instance of a different version but of the same flavor/qualifier pair as current, the `current.chain` file contents changes to point to the new version. This is true for any chain value, not just for current.

11.5 Undeclare and Remove an Instance

To undeclare a product instance means to remove all information pertaining to it from the **UPS** database in question. The information that gets removed includes:

- the version file, or the portion of the version file, that pertains to the instance

- any chain files, or the portions of any chain files, that pertain to the instance

The command **ups undeclare** is provided for this operation. You can opt to remove the actual product in the product instance's root directory, as well, by using either the **-y** or **-Y** option, as described in section 11.5.1 *Using ups undeclare to Remove a Product*. The **ups undeclare** command executes **ups unconfigure** by default (see section 4.6.1 *Configuring a Product*). The unconfigure process can be suppressed by using the **-C** option with **ups undeclare**, however normally you want this process to execute. The full **ups undeclare** command description and option list is in section 23.19 *ups undeclare*.

It is also possible to configure **UPP** to remove a product automatically. This is discussed in section 11.5.3 *Using UPP to Remove a Product*.

Before removing anything, you should find out if any other products have the product instance in question declared as a dependency.¹ If so, you may want to reconsider removing it. Removal of the product instance may affect the operation of its parent products.

11.5.1 Using ups undeclare to Remove a Product



To remove a product instance, you must specify the *version* of the instance, not its *chain*, in the **ups undeclare** command. Specifying the chain removes only that chain, not the instance itself.



Using ups undeclare 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, which can lead to a fragmented products area.

If you choose to completely remove the product, and you want to delete the product instance's directory tree starting from its root directory, use one of the options **-y** or **-Y** with **ups undeclare** (**-y** queries you for confirmation, **-Y** does not). We recommend always including the **-f** option if you have a multi-flavor database. You may also need to include the **-z** option if you have more than one database. The command syntax is (showing commonly used options):

```
% ups undeclare [-f <flavor>] [-q <qualifierList>] [-y|Y] \
  [-z <database>] <product> <version>
```



Special case: If a product has a CONFIGURE action that modifies files outside of its product root directory, and if this instance is used by more than one node, flavor or file system, then you may need to run **ups undeclare**

1. The **ups parent** command will provide this information. The command is not available as of **UPS** version v4_5_2; it is planned for a future release.

or **ups unconfigure** on all of the nodes before removing the product files on any node. The `INSTALL_NOTE` file should indicate if this is the case. If you're not sure, check in the product's table file.

Example 1

In this first example, we remove the product **tcl v7_6a**. We undeclare it and opt to remove the product root directory after query, taking a “snapshot” before and after. First, verify the declared instances of **tcl** in the database:

```
% ups list -aK+ tcl
      "tcl" "v8_0_2" "IRIX+5" "" "current"
      "tcl" "v7_6a" "IRIX+5" "" ""
```

Next verify the product root directory contents (run **setup** to set `$TCL_DIR`, check contents of the **tcl** products area, and then list contents of `$TCL_DIR`):

```
% setup tcl v7_6a
% cd $TCL_DIR/../../ ; ls -l
total 8
drwxrwxr-x    9 aheavey  g020           140 Sep 15 15:29 v7_6a/
drwxrwxr-x    9 aheavey  g020       4096 Sep  8 15:50 v8_0_2/
% cd v7_6a ; ls -l
total 40
-rw-r--r--    1 aheavey  g020           165 May  1 1997
BUILD_INFO
-rw-r--r--    1 aheavey  g020       5861 May  1 1997
Makefile
drwxrwxr-x    2 aheavey  g020           83 Sep 15 15:29 alt-ups
drwxrwxr-x    2 aheavey  g020           40 Sep 15 15:29 bin
...
```

Now undeclare **tcl v7_6a** and remove its product root directory structure. The **-y** option queries before removing, and we respond “**y**” for yes (one would enter “**n**” for no):

```
% ups undeclare -f IRIX+5 tcl v7_6a -y
Product home directory -
      /export/home/t1/aheavey/upsII/products/tcl/v7_6a/
Delete this directory?y
```

Once it finishes, verify the deletion:

```
% ups list -aK+ tcl
      "tcl" "v8_0_2" "IRIX+5" "" "current"
% cd $TCL_DIR/../../ ; ls -l
total 8
```

```
drwxrwxr-x    9 aheavey  g020          4096 Sep  8 15:50 v8_0_2
```

We see that the declaration was removed and the `v7_6a` directory is gone from the `tcl` product area.

Example 2

The following command is a dangerous example! We include it as a caution. It finds the best flavor match using the standard instance selection algorithm (see section 27.2 *Instance Matching within Selected Database*) and removes that instance of the product **pine** version `v3_91` and any chains that point to it. It also removes the product root directory for this instance of **pine**; it does not query for confirmation before doing so.

```
% ups undeclare -Y pine v3_91
```

Depending on the instances you have in your database, you may end up removing the instance for, say, `OSF1+V3` when you really wanted to remove the one for `OSF1`!

11.5.2 Undoing Configuration Steps

There is a `ups unconfigure` command for undoing configuration steps, described in section 23.18 *ups unconfigure*. Normally this command does not need to be run explicitly; the `ups undeclare` command undoes the reversible configuration operations by default.¹ Refer to the `INSTALL_NOTE` file for instructions.

11.5.3 Using UPP to Remove a Product

It is possible to configure **UPP** to remove a product automatically. To do this you must create or edit a subscription file for **UPP**, which is documented in Chapter 33: *The UPP Subscription File*. Within the file you identify the instance(s), set a condition to trigger its removal, and provide the instruction to remove it.

There are two conditions that **UPP** recognizes:

- a new version of the product is available on the distribution node
- the chain on the specified product instance changes

1. When a product is undeclared, any steps in the table file under `ACTION=UNCONFIGURE` get executed by default, or the (reversible) functions under `ACTION=CONFIGURE` get undone. These concepts are explained in section 34.2.2 “*Uncommands*” as *Actions*.

The appropriate instruction to use for removing a product is `delete`, as documented in section 33.2.2 *Conditions and Instructions*. When the condition is met, **UPP** executes `ups undeclare -Y` for you (which removes the product root directory structure in addition to the declaration).

Here we provide a sample subscription file stanza for removing a product when its current chain gets removed on the server (we include the `notify` function in addition to `delete`, which is always a good idea):

```
product = exmh      Identify subscribed product as exmh (the exmh versions
                    remain unspecified in this example, therefore act on all ver-
                    sions for the flavor specified below).

flavor =           Identify flavor of product (this is optional)
SunOS+5.5

action =          List in the following lines one or more functions to perform
uncurrent         when an instance of the listed product-flavor combination is
                  unchained from current on the server.

notify           Send a notification message to <userid>@fnal.gov
                  (specified in file header)

delete           Remove the instance (declaration and product root direc-
                  tory) from the local node.
```

11.6 Verify Integrity of an Instance

The `ups verify` command checks the information in all the database files for the specified instance in order to determine if there are any errors or inconsistencies. The full command description and option list is in section 23.20 *ups verify*. Shown here with some commonly-used options, the command syntax is:

```
% ups verify -a <chainFlag> [-f <flavor>] <product> \
  [<version>]
```

Here is sample output for a product for which several files and directories listed in the version file were not found (**-a** is included to match all instances):

```
% ups verify -a blt

DATABASE=/path/to/upsdb
WARNING: File not found - /myman/
WARNING: File not found - /mycatman
WARNING: File not found - /myinfo/
WARNING: File not found - /myhtml/
WARNING: File not found - /mynews/
WARNING: File not found - /path/to/upsdb/.updfiles
INFORMATIONAL: Verifying product 'blt'
WARNING: File not found - /usr/products/IRIX/blt/v2_1
WARNING: File not found - /usr/products/IRIX/blt/v2_1/ups
```

11.7 Modify Information in a Database File

The **ups modify** command allows you to manually edit any of the database product files. It runs **ups verify** on the instance to perform syntax and content validation before and after the editing session. The full command description and option list is in section 23.12 *ups modify*. The command syntax with some commonly used options is:

```
% ups modify <product> [<version>] [-E <editor>] [<chainFlag>] \  
  [-N <fileName>] [-z <database>]
```

ups modify performs the following steps (if you specify the file using **-N**, the menu will not appear):

- presents menu of files that you can edit and asks you to either select one or quit
- verifies pre-modification contents of file (runs **ups verify**)
- starts up the editor given by **-E <editor>** or, if that is not specified, then \$EDITOR, if set. If neither is specified, it starts up **vi** by default.
- makes a copy of the file to be edited
- pulls copy of file into the editor
- after user exits the editor, runs **ups verify** on the edited file
- if the validation succeeds, writes the new file over the old one and quits
- if the validation does not succeed, provides informational messages, asks if you want to save changes, and quits
- if no changes made to file, again presents menu of files

Sample Session with (1) Unsuccessful and (2) Successful Validation

```
% ups modify teledata v1_0 -N $MYDB/teledata/v1_0.version
```

In this example, we select the version file (via **-N**) for the product **teledata v1_0** (default flavor, no qualifiers). Since **-E** is not given, **UPS** will use the editor set in **\$EDITOR**, or **vi** if that variable is not set. First, **UPS** runs **ups verify** and produces the output:

```
Pre modification verification pass complete.
```

No errors were detected. The version file is next displayed in the editor.

1) To illustrate an unsuccessful validation, we add a bogus line:

```
TESTKEYWORD = value
```

and save and quit. **UPS** returns the following messages, and we opt to save the erroneous change:

```
INFORMATIONAL: Unexpected key word 'TESTKEYWORD' in
'/home/t1/aheavey/upsII/decl
ared/teledata/v1_0.version', line 17
INFORMATIONAL: Unexpected key word 'TESTKEYWORD' in
'/home/t1/aheavey/upsII/decl
ared/teledata/v1_0.version', line 17
Post modification verification pass complete.
Do you wish to save this modification [y/n] ? y
```

UPS quits, saving the file as we requested.

2) To illustrate successful validation, we'll correct the error introduced above. We run the same **ups modify** command. **UPS** finds the error during the pre-edit validation:

```
INFORMATIONAL: Unexpected key word 'TESTKEYWORD' in
'/home/t1/aheavey/upsII/decl
ared/teledata/v1_0.version', line 17
INFORMATIONAL: Unexpected key word 'TESTKEYWORD' in
'/home/t1/aheavey/upsII/decl
ared/teledata/v1_0.version', line 17
Pre modification verification pass complete.
```

We remove the incorrect line from the version file, then save and quit.

UPS displays the following message, and we elect to save the change (**y**):

```
Post modification verification pass complete.
Do you wish to save this modification [y/n] ? y
```

UPS quits, saving the file as requested.

Sample Session with No Changes

In this example, we select the current instance of the product **teledata**, and (by default) request a menu of files to edit:

```
% ups modify teledata
[0] /home/t1/ahavey/upsII/declared/teledata/current.chain
[1] /home/t1/ahavey/upsII/declared/teledata/v1_0.version
[2]
/export/home/t1/ahavey/upsII/products/teledata/v1_0//ups/v1
_0.table
[3] /home/t1/ahavey/upsII/declared/.upsfiles/dbconfig
Choose file to edit [0-3] or 'q' to quit: 1
Pre modification verification pass complete.
```

UPS starts up the editor and makes the selected file available to edit. We quit without making any changes. **UPS** displays the message:

```
No modifications, nothing to save.
```

UPS then displays the menu again, and we opt to quit:

```
[0] /home/t1/ahavey/upsII/declared/teledata/current.chain
[1] /home/t1/ahavey/upsII/declared/teledata/v1_0.version
[2]
/export/home/t1/ahavey/upsII/products/teledata/v1_0//ups/v1
_0.table
[3] /home/t1/ahavey/upsII/declared/.upsfiles/dbconfig
Choose file to edit [0-3] or 'q' to quit: q
```

11.8 Determine If a Product Needs to be Updated

UPP can be configured on a local machine to alert users via email when a newer version of a product is available in **KITS**, or when a product instance's table file or **ups** directory needs to be updated. If your installation is not configured to do this, you can use **UPD** interactively to find this information.

11.8.1 Using UPP

UPP can be used for several functions as described briefly in section 2.1 *Introduction to UPS, UPD and UPP*, and in detail in Chapter 33: *The UPP Subscription File*. For instructions on how to configure **UPP** to notify you regarding a product, see Chapter 33 or 11.5.3 *Using UPP to Remove a Product*.

11.8.2 Using UPD

To determine if you need to reinstall a product, use the **upd install** command with the **-s** option, as shown, while logged on to a node of the flavor you wish to check (or use the **-H** option to specify a different flavor). The full command description and option list is in the reference section 24.8 *upd install*.

```
% upd install -sv <product> [<version>] [-h <host>] \  
  [-H <flavor>]
```

If it's ok, you'll see no output. If there's a discrepancy between what's on your node and what's on *fnkits* (or on the host specified using **-h**), you'll see output of the form:

```
Installing <product>  
I would make directory /path/to/<product>/<flavor>/<version>  
I would fetch directory <kitsflavor> from  
ftp://fnkits.fnal.gov/ftp/products/<product>/<version> as  
/path/to/<product>/<flavor>/<version> now  
...
```

Although it says "Installing", it's only telling you what it would have to do in order to install.

If you are interested in knowing only if the product's table file or **ups** directory has been changed on the server and needs an update on your machine, use the **upd update -s** command. It compares the MODIFIED dates in the remote and local nodes. The full command description and option list is in the reference section 24.13 *upd update*.

```
% upd update -s <product> [<version>] <component> \  
  [-H <flavor>]
```

The argument **<component>** can take the value **table_file** or **ups_dir**, or both, colon-separated. If no update is needed, there is no output. If an update is needed, the messages will inform you.

11.9 Update a Table File or ups Directory

The **upd update** command is used to update a product's table file and/or **ups** directory. It operates on the specified product instance and its dependencies by default. It retrieves the specified components from a distribution node and downloads them to the local node, overwriting the corresponding pre-existing component(s). The full command description and option list are in section 24.13 *upd update*. The command syntax with some commonly used options is:

```
% upd update <product> [<version>] <componentList> \
  [-H <flavor>] [<chainFlag>] [-h <host>] [-i] [-j]
```

In the following example, we overwrite the table file for the product instance **xntp v3_4**, flavor SunOS. This operation will succeed if the MODIFIED date in the remote version file that points to the table file on the distribution node is later than that in the comparable local version file; no overwrite will occur otherwise. Before running **upd update**, we compare the MODIFIED dates for the product by using a **ups list** command like the following:

```
% ups list -f SunOS -K MODIFIED xntp v3_4
":1998-04-01 20.08.02 GMT"
```

on the local node, and running **upd list** with similar options on the distribution node (the default *fnkits* is used here):

```
% upd list -H SunOS -K MODIFIED xntp v3_4
"1998-09-10 08.13.07 GMT"
```

The MODIFIED date in the remote version file is later than that in the local version file, therefore we expect an update to occur.

Now we run the **ups update** command requesting the component **table_file**:

```
% upd update table_file xntp v3_4 -H SunOS
updcmd::updcmd_update - Updating xntp.
upderr::upderr_syslog - successful transfer
ftp://fnkits.fnal.gov//ftp/upsdb/xntp/v3_4SunOS.table ->
/tmp/mwmdb/xntp/v3_4.table
upderr::upderr_syslog - successful ups touch xntp v3_4 -f
SunOS -q "" -U ""
```

Rerun the **ups list** command to verify that the MODIFIED date changed, indicating that the update took place.

To update several instances of **xntp v3_4** for a list of flavors, use the **-H** option like this:

```
% upd update table_file xntp v3_4 -H SunOS:IRIX:OSF1:Linux
```

Using **-H** ensures that all the dependencies are updated with the appropriate flavor rather than with the best match flavor to the local machine.



Note: When updating several instances at a time, you can exclude a particular instance from being updated by running **ups touch** on it. See the reference section 23.17 *ups touch* for more information.

11.10 Retrieve an Individual File

The **upd fetch** command retrieves a single file or directory maintained in a **UPS** distribution database, and downloads it to the user node, placing it relative to the current working directory. The **-J** option is used to specify the individual filename to fetch. If **-J** is omitted, the output is a recursive list of directories and files that are available for individual retrieval. Nothing actually is retrieved when **-J** is omitted. The full command description and option list is in section 24.6 *upd fetch*. The command syntax with some commonly used options is:

```
% upd fetch [-H <flavor>] [<chainFlag>] [-h <host>] \  
  [-J fileName] <product> [<version>]
```

First we issue the **upd fetch** command without the **-J** option to find out what files are available for the specified product instance (output edited for brevity):

```
% upd fetch -H IRIX+6.2 rbio v9_3d  
  
Listing of table_dir [/ftp/products/rbio/v9_3d/IRIX+6.2]:  
total 3172  
drwxrwx---   3 updadmin upd           512 May   7   1999  
rbio_v9_3d_IRIX+6.2  
-rw-rw-r--   1 updadmin upd          1235 May   7   1999  
rbio_v9_3d_IRIX+6.2.table  
-rw-rw----   1 updadmin upd        1597440 May   7   1999  
rbio_v9_3d_IRIX+6.2.tar  
-rw-rw-r--   1 updadmin upd          14848 May   7   1999  
rbio_v9_3d_IRIX+6.2.ups.tar  
  
rbio_v9_3d_IRIX+6.2:  
total 6  
-rw-rw-r--   1 updadmin upd          1540 May   7   1999 README  
drwxrwsr-x   5 updadmin upd           512 May   7   1999 ups  
  
rbio_v9_3d_IRIX+6.2/ups:  
total 28  
-rw-rw-r--   1 updadmin upd           210 May   7   1999 Version  
...  
rbio_v9_3d_IRIX+6.2/ups/toInfo:  
total 0  
...
```

Now we use **upd fetch -J** to retrieve the **README** file listed. The file will be copied to the current working directory:

```
% upd fetch -J README -H IRIX+6.2 rbio v9_3d  
informational: transferred README
```

```
from
fnkits.fnal.gov:/ftp/products/rbio/v9_3d/IRIX+6.2/rbio_v9_3d
_IRIX+6.2
to ./README
```

To verify the successful transfer, we check the current working directory for the new file:

```
% ls -l README
-rw-rw-r-- 1 aheavey g020 1540 Sep 7 15:57 README
```

As another example, you can retrieve the table files for several flavors of a product. When specifying the flavors on the remote node, be sure to use **-H**, not **-f**:

```
% upd fetch -H SunOS+5:IRIX+6:Linux+2:OSF1+v4 -J @table_file \
tex v3_14159
```

This command retrieves the table file(s) for the best match product instances of **tex v3_14159** for the listed flavor families. Depending on how the product was configured, the same table file may be used for all, or they may be separate files. The file(s) will be copied to the current working directory.

11.11 Check Product Accessibility

The **ups exist** command is used to test whether a **setup** command issued with the same command line elements is likely to succeed. It checks for a properly declared matching instance, and verifies that you have the necessary permissions to create the temporary file used by the **setup** command.¹ This command is rarely used from the command line, and is more useful in scripts where a failed setup could cause the script to abort. The full command description and option list is in section 23.7 *ups exist*. The command syntax with some commonly used options is:

```
% ups exist [-f <flavor>][<chainFlag>] [-j] <product> \
[<version>]
```

When issued from the command line, it returns no output if the command succeeds. In the C shell family **ups exist** sets the `$status` variable to 0 if it was able to create the temporary file, or to 1 for error. In the Bourne shell family, it sets the `$?` variable similarly. As an example, we can run **ups list** (not shown here) and find that there is a current instance of the product **tex** for the flavor IRIX+6 but not for IRIX+6.2. Running **ups exist** for each flavor, we see that the variables get set accordingly. For the C shell family:

1. Specifically, it determines whether **setup** can create the temporary file. If so, it creates it, but it does not execute it.

```
% ups exist tex -f IRIX+6; echo $status
```

```
0
```

```
% ups exist tex -f IRIX+6.2; echo $status
```

```
1
```

For the Bourne shell family:

```
$ ups exist tex -f IRIX+6; echo $?
```

```
0
```

```
$ ups exist tex -f IRIX+6.2; echo $?
```

```
1
```

To run this on a product distribution node, use the corresponding command **upd exist**, documented in section 24.5 *upd exist*.

11.12 Troubleshooting

This section provides a few hints if things don't seem to work after declaring/removing/changing a product, or otherwise modifying files in a **UPS** database.

- If the `$PATH` goes away, restore it by running:

```
% setup setpath
```

and check if the `pathSet` function is used in the table file -- if it is set wrong, this may be the cause.

- To print out diagnostic information about what might be wrong with a product declaration, run **ups verify**:

```
% ups verify -a <product> [<version>]
```

- Try setting up just the main product and none of its dependencies. This should help determine which file has the problem, the main one or a dependency. Use `-j` in the **setup** command:

```
% setup -j <product>
```

- Print out verbose information using the `-v` option with **setup**:

```
% setup -v <product>
```

To get progressively more information, use multiple `v`'s, e.g., `-vv`, `-vvv` (up to four).

- Check file permissions. Any scripts called by the table file must be both readable and executable. The product executable(s) must of course be executable. The product database files must be readable.
- To examine the temporary file that the **setup** command creates and sources, run the command:

```
% ups setup <product> [<version>]
```

This returns the path of this temporary file, and you can then go look at the file. For example:

```
% ups setup ocs
```

```
    /var/tmp/aaaa00273
```

- For most **UPS** commands, the **-s** option can be used to simulate the command (i.e., create the temporary file) without executing it. It also returns the path of the temporary file it created, for example:

```
% setup -s -z /products/ups_database/upsII/main  
xpdf
```

```
    INFORMATIONAL:  Name of created temp file is  
    /var/tmp/aaaa005Mt
```

- If home directories move or if older versions of products have been deleted, you might want to prevent execution of **unsetup** files prior to a subsequent setup. In this case, don't **unsetup** the product. Just setup the product again using **-k**:

```
% setup -k <product>
```