

Chapter 37: Scripts You May Need to Provide with a Product

In **UPS** v4, the functions supported for use in table file actions will not always suffice for completing certain tasks, for instance configuration and tailoring. You may still need to provide executable scripts, and include appropriate functions in your table file to execute them. In this chapter we discuss some scripts you may need to provide with your product.

Since these types of scripts generally get executed only once, speed isn't critical. We plan to provide more functions in later **UPS** releases so that scripts will no longer be necessary for this purpose.



Note that these files can be binaries, but scripts are recommended.

37.1 configure and unconfigure

The `configure` executable must perform whatever steps are necessary to install the product on a system, minus anything that requires direct interactive input from the installer. In cases where the installer must supply some information, you can choose to use a `tailor` script to collect data, and pass the values to the `configure` script to use.

The `unconfigure` executable must undo everything that `configure` does. **UPS** is “smart” enough that if one of the functions **sourceOptCheck**, **sourceOptional**, **sourceReqCheck**, or **sourceRequired** is used in the CONFIGURE action, when **ups unconfigure** is run, **UPS** can find and source the `unconfigure.${UPS_SHELL}` script.

Here is an example. Say a CONFIGURE action specifies:

```
ACTION=CONFIGURE
```

```
sourceOptional(${UPS_UPS_DIR}/configure.${UPS_SHELL}, UPS_ENV )
```

When you run the **ups unconfigure** command, **UPS** first looks for ACTION=UNCONFIGURE, as usual. Failing to find it, **UPS** next looks for ACTION=CONFIGURE. Upon encountering the **sourceOptional** function, it searches for the file `unconfigure.${UPS_SHELL}` in the same directory (`${UPS_UPS_DIR}`), and sources it.

Sample Configure Script

The **tex_files** product has a good example of a `configure/current` script (they are identical in this case):

```
#!/bin/sh
# "current" and "configure" for $TEX_FILES_DIR/ups/current

case "$TEX_FILES_DIR" in
/afs*)
    find $TEX_FILES_DIR/texmf/fonts/tmp -type d \
-exec fs setacl {} system:anyuser rlidkw \;
    ;;
*)
    chmod -R 1777 $TEX_FILES_DIR/texmf/fonts/tmp
    ;;
esac
```

The directory `$TEX_FILES_DIR/texmf/fonts/tmp` must be writable by anybody using **tex_files**, in order that **TeX** can create the requested fonts on-the-fly from font metadata files. (This way, rarely-used fonts can be generated as the document is created, and they don't need to be stored.) The script evaluates `$TEX_FILES_DIR`. If it begins with `/afs`, it runs the appropriate AFS command to make the `tmp` area world-writable. If not, then it uses the standard UNIX **chmod**. **UPS** does not yet have “if-then-else” capability within table files, so we can't write these things into actions. The table file calls the scripts via the actions:

```
action=configure
    prodDir()
    execute(${UPS_UPS_DIR}/configure,UPS_ENV)
    unprodDir()
action=current
    prodDir()
    execute(${UPS_UPS_DIR}/current,UPS_ENV)
    unprodDir()
```

As described in section 34.5 *Actions Called by Other Actions*, one of the identical scripts could have been eliminated and a common action could have been used in this way:

```
action=configure
    exeActionRequired("common")
action=current
    exeActionRequired("common")
action=common
    proddir()
    execute(${UPS_UPS_DIR}/configure,UPS_ENV)
    unprodDir()
```

Configure Scripts for Products with Hard-Coded Paths

As discussed in section 16.1.3 *Third-Party Products Requiring a Hard-Coded Path*, many third-party products require a hard-coded path assigned when the product is built. Most of these products come with configurable Makefiles thereby allowing you to choose the path. 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 these products. The third approach that it discusses involves using the `configure` script to modify a trampoline executable. Please refer to TN0086 for information.

37.2 tailor

As discussed in section 4.6.2 *Tailoring a Product*, tailoring is the aspect of the product implementation that requires input from the product installer (e.g., the location of hardware devices for a software driver package, a specific area for log files, which node should run the server, etc.). If your product requires any interactive input from the installer, you will need to furnish a `tailor` executable for this purpose. Generally `tailor` files are scripts that ask the installer a series of questions, and write the answers to a `<node>.dat` file which in turn gets read by the `configure`, `current`, and/or `start` scripts.

Usually undoing the steps done via `tailor` require interactive input. However, if your tailor steps are such that they can be undone via a script, go ahead and provide an `untailor` script. When you run the **ups untailor** command (available via the unknown command handler discussed in section 34.4 *The "Unknown Command" Handler*), **UPS** will execute `untailor`, the same way as described for `unconfigure` in section 37.1 *configure and unconfigure*.

It still may be best to avoid including anything in `tailor` that needs to be undone when the product is removed and that requires input from a person. If `tailor` is used to collect information and pass it to the `configure` script (recommended), then anything that needs to be undone can be addressed in `unconfigure`.

For a sample tailor script, see `$JUKE_DIR/ups/tailor`.

37.3 current and uncurrent

Most things that need to be done when a product instance is declared current can be done directly via functions in the table file in a CURRENT action. However, if the available functions prove to be insufficient for your product, create a `current` script to perform the function(s).

Likewise, when a current chain is removed from a product instance, the `uncurrent` script (if it exists) should undo all the things that were done in `current`. It works the same way as UNCONFIGURE, described in section 37.1 *configure and unconfigure*.

A sample `current` script is shown in section 37.1 *configure and unconfigure*.

37.4 start and stop

The `start` and `stop` files may be needed if your product needs to startup automatically at boot time and run until system shutdown. Refer to Chapter 15: *Automatic UPS Product Startup and Shutdown* for information on this topic. In the table file for this type of product you must include the actions ACTION=START and ACTION=STOP. These actions must include all the steps necessary to startup the product and shut it down. You may need to put these steps in scripts and execute them from the table file. You can call the scripts whatever you like, but we recommend `start` and `stop` for easy recognition.

Sample start and stop Scripts

We'll use scripts for `juke v5_2` as examples.

The start script

```
#!/bin/sh

case "$0" in
/*)    JUKE_DIR=`echo $0 | sed -e `s;/ups/start;`
        export JUKE_DIR
        PATH=$JUKE_DIR/bin:$PATH
        ;;
*)     ;;
esac
```

```

cd $JUKE_DIR/log

host=`hostname`
local="`$JUKE_DIR/bin/juke show jukebox | grep $host | sed
-e `s/@.*//`"

if [ "$local" != "" ]
then
    if [ -f $JUKE_DIR/log/jukerpcd.$host.pid ]
    then
        # it looks like one is running
        if kill -0 `cat
$JUKE_DIR/log/jukerpcd.$host.pid`
        then
            #daemon is already running, we're done
            exit 0
        fi
    fi
    nohup $JUKE_DIR/bin/jukerpcd >> jukerpcd.$host.out
2>&1 </dev/null &
    echo $! > $JUKE_DIR/log/jukerpcd.$host.pid

    sleep 10      # wait for jukerpcd to wake up

    for i in $local
    do
        if [ "`uname -s`" = "AIX" ]
        then
            # AIX driver doesnt autoconfigure, so
            configure it
            dev=`$JUKE_DIR/bin/juke show jukebox |
                grep $local |
                sed -e `s:.*dev/;;` -e `s/[
].*//`
            mkdev -l $dev
        fi
        $JUKE_DIR/bin/juke online -j $i &
    done
fi

```

The stop Script

```
#!/bin/sh

if [ "" = "$JUKE_DIR" ]
then
    JUKE_DIR=`echo $0 | sed -e `s:/ups/stop;`´
    export JUKE_DIR
    PATH=$JUKE_DIR/bin:$PATH
fi

host=`hostname`

if [ -f $JUKE_DIR/log/jukerpcd.$host.pid ]
then
    kill -15 `cat $JUKE_DIR/log/jukerpcd.$host.pid`
    rm $JUKE_DIR/log/jukerpcd.$host.pid
fi
```