Creating a Simple Package for Solaris 2.0

Patrick Stirling

Sun MIcrosystems Computer Corporation

November 9, 1992

The distribution format for software products is very different in Solaris 2.0 than it was for previous versions of SunOS. Solaris 2.0 uses the *package* mechanism from AT&T's SVR4. This defines the distribution format for software products, which can then be added to or removed from a system with standard utilities.

This paper is a tutorial on creating a simple Solaris 2.0 package; the example used will be a SCSI device driver.

Packaging is quite complex; this paper covers only those aspects relevant to the example driver. For more information refer to the *SunOS 5.0 Application Packaging and Installation Guide*, part number 800-6347-10 (also in the Answer-Book).

1.0 Packaging Overview

The components of a package fall into two categories: *package objects*, the files to be installed; and *control files*, that control how and where the package is installed. There are a number of tools to help you create a package.

Packages are distributed in a standard format on diskette or tape, and are installed with the `pkgadd` utility.

1.1 Package Objects

These are the files that are being distributed, the actual contents of the package. For the driver example here, they are the loadable driver module, the hardware configuration file, the driver's header file, and a simple test program.

Package objects can be *fixed* or *relocatable*. Fixed objects are installed into predefined directories on the installation machine; the administrator can choose where relocatable objects are installed.

1.2 Control Files

There are only two required control files: a *prototype* file that defines the package contents (including all control files); and a *pkginfo* file that sets package characteristics (e.g. its name).

There are several optional information files that can be included in the package; the only one used in this example is a *copyright* file.

There are three types of installation scripts, all of which are optional. These scripts must be executable by the Bourne Shell (i.e. can be shell scripts or executable programs). The package installation utility, `pkgadd(1M)` runs installation scripts by name. The three types of script are:

- a *request* script that solicits user input. There can be only one request script, and it must be called `request`.
- *class action* scripts that define a set of actions to be performed on a set of package objects. These scripts are named after the class and can be run during package installation or removal. See the next section for more information.
- *procedure* scripts that define actions that will occur at certain stages of the installation. There can be up to four procedure scripts, named after the stage at which they should run. During installation, `preinstall` is run before any files are installed and `postinstall` is run after all files have been installed. During removal (`pkgrm(1M)`), `preremove` is run before any files are removed and `postremove` is run after all files have been removed.

1.2.1 Classes

Package objects can be grouped into *classes*, defined in the *`prototype`* file. Class action scripts can then be written for each class, and will be run only against objects in the class. The scripts must be named `i.`*`class`* for installation, or `r.`*`class`* for removal scripts.

There are also three special classes:

- *sed*; this provides a method for using `sed` instructions to edit files on package installation and removal.
- *awk*; similar to sed, for `awk`.
- *build*; provides a method to dynamically construct a file during installation.

1.3 Creating a Package

The `pkgmk(1)` utility reads the `prototype` file and creates an installable package. Before running `pkgmk`, all of the control files must be written, the package objects must be organized, and the `prototype` file must be created. The next few sections discuss each of these steps.

2.0 Create the Package Control Files

This package, a SCSI device driver, will have the two mandatory control files (`prototype` and `pkginfo`) and `postinstall`, `sed`, and `preremove` scripts. There will also be a `copyright` file.

The `prototype` file defines all of the contents of the package, that is, it contains an entry for each package object and for each control file (except itself). Therefore, this file is discussed last (on page 6), immediately before creating the package itself.

2.1 The `pkginfo` File

This is an ASCII file, called `pkginfo`, that describes the characteristics of the package. It also contains installation control information. The file consists of a list of *parameter*=*value* pairs. This is the `pkginfo` file for our driver example:

```
PKG="SUNWsst"
NAME="Simple SCSI Target Driver"
VERSION=1
CATEGORY=system
ARCH="sparc"
VENDOR="Sun Microsystems"
BASEDIR=/opt
```

All but the `BASEDIR` entry are mandatory. `BASEDIR` defines where relocatable package objects will be installed. Its setting can be overridden during installation (see "Adding a Package" on page 9). If you don't specify a `BASEDIR` in the `pkginfo` file, and don't specify a location when installing, relocatable files will be installed into the root directory (probably not what you want).

The settings of `PKG`, `VERSION` and `ARCH` together define the package *instance*. The installation utility, `pkgadd(1m)`, distinguishes instances by appending an instance number to the package name.

The recommended naming convention for packages is the company stock symbol followed by the package name.

See the man page `pkginfo(4)` for full details of the parameters.

2.2 The `sed` Class Script

Sed class scripts allow you to modify files that already exist on the system. The script's name indicates the file that the `sed(1)` instructions in the script are to be executed against. Instructions after the keyword `!install` are executed during installation (after a `preinstall` script but before a `postinstall` script). Instructions after the keyword `!remove` are executed during package removal, in between the `preremove` and the `postremove` scripts.

In the driver example, a `sed` class script is used to add an entry for the driver to the file `/etc/devlink.tab`. This file is used by `devlinks(1m)` to create symbolic links from `/dev` into `/devices`. This is the script:

```
# Sed class script to modify /etc/devlink.tab

!install
/name=sst;/d
$i\
type=ddi_pseudo;name=sst;minor=character       rsst\\A1

!remove
/name=sst;/d
```

2.3 The `postinstall` Installation Script

This is a Bourne shell script that's run after all files have been installed and all class scripts have been run. In our example, all the script needs to do is run the `add_drv(1m)` utility:

```
# Postinstallation script for SUNWsst
SAVEBASE=$BASEDIR
BASEDIR=""; export BASEDIR
/usr/sbin/add_drv sst
STATUS=$?
BASEDIR=$SAVEBASE; export BASEDIR
if [ $STATUS -eq 0 ]
then
  exit 20
else
  exit 2
fi
```

Unfortunately, `add_drv` uses `BASEDIR`, so the script has to unset it before running the utility, and restore it afterwards.

The exit code from `postinstall` is significant. 20 tells `pkgadd` to tell the user to reboot the system (necessary after installing a driver), and 2 tells `pkgadd` to tell the user that the installation partially failed.

One of the actions of `add_drv` is to run `devlinks`, which will use the entry placed in `/etc/devlink.tab` by the `sed` class script to create the `/dev` entries for the driver.

2.4 The `preremove` Removal Script

This is also a Bourne shell script, and is executed before any package objects are removed from the system. It undoes the action(s) of the `postinstall` script. In the case of this driver example, it simply removes the links in `/dev` and runs `rem_drv(1m)` on the driver. Actually, the script could equally well be `postremove` since `rem_drv` could be executed before or after the driver files are removed.

```
# Pre removal script for the sst driver
echo "Removing /dev entries"
/usr/bin/rm -f /dev/rsst*

echo "Deinstalling driver from the kernel"
SAVEBASE=$BASEDIR
BASEDIR=""; export BASEDIR
/usr/sbin/rem_drv sst
BASEDIR=$SAVEBASE; export BASEDIR

exit 0
```

The script removes the `/dev` entries itself; the `/devices` entries are removed by `rem_drv`.

2.5 The `copyright` File

This is a simple ASCII file containing the text of a copyright notice. The notice is displayed at the beginning of package installation exactly as it appears in the file.

```
  Copyright (c) 1992 Drivers-R-Us, Inc.
  10 Device Drive, Thebus, IO 80586

All rights reserved. This product and related documentation is protected by
copyright and distributed under licenses restricting its use, copying, dis-
```

```
tribution and decompilation. No part of this product or related documenta-
tion may be reproduced in any form by any means without prior written
authorization of Drivers-R-Us and its licensors, if any.
```

3.0 Creating a Package

The main task in creating a package is to create the `prototype` file. This file specifies the locations of the package objects on both the development and the installation workstations. Before creating the `prototype` file, the layout of the package objects must be determined.

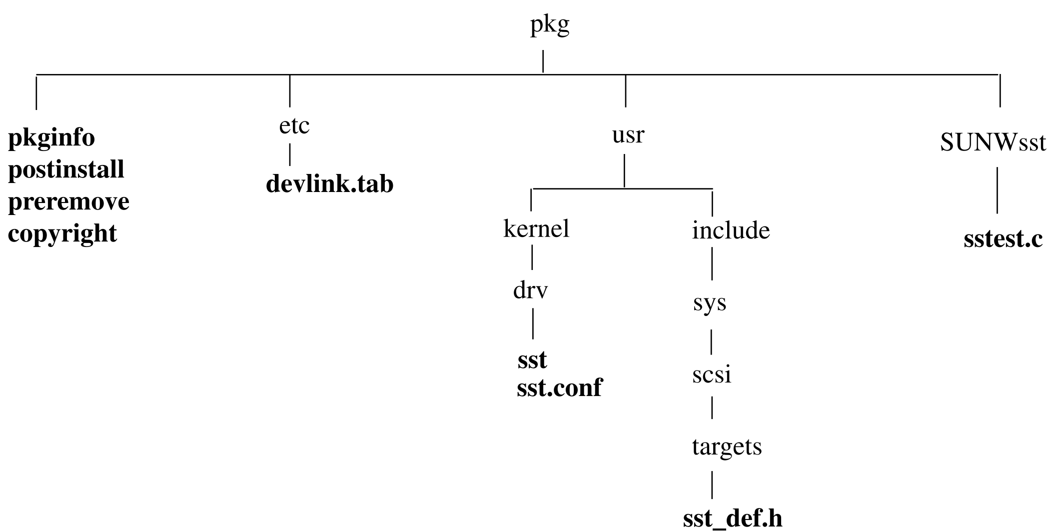
3.1 Organize the Package Objects

The first step in creating the package is to organize its contents. We'll look at two ways of doing this:

- *hierarchical*, where the objects on the development machine are in the same directory structure as they will be after installation;
- *flat*, where the objects on the development machine are in a single directory. In this case, the `prototype` file contains information on the placement of objects on both the development and installation workstation.

3.1.1 Hierarchical Directory Structure

The source file directory structure must be a mirror of the desired structure on the installation machine:

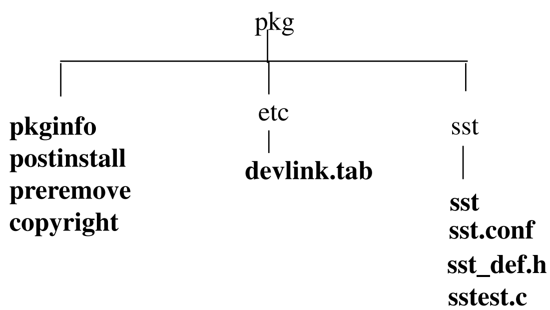


The package objects will be installed in the same places as they are in the `pkg` directory above. The driver modules (`sst` and `sst.conf`) will be installed into `/usr/kernel/drv` and the include file will be installed into `/usr/include/sys/scsi/targets`; these are *fixed* objects. The test program, `sstest.c`, and its directory `SUNWsst` are *relocatable;* their installation location is set by the `BASEDIR` parameter in the `pkginfo` control file (which can be overridden by the administrator, see "Adding a Package" on page 9).

The remaining components of the package (all the control files) go in the top directory of the package on the development machine, except the `sed` class script. This is called `devlink.tab` after the file it is to modify, and goes into `etc`, the directory containing the real `devlink.tab` file.

3.1.2 Flat Directory Structure

It may be more convenient to put all the package objects into a single directory on the development machine. In our example this is the case, since the installation directory structure is quite sparse.



3.2 Create the `prototype` File

This is an ASCII file that defines the contents of the package. It contains one entry per package object, and per control file. See the man page for `prototype(4)` for details. The `pkgproto(1)` utility will create an initial `prototype` file which you can edit. It reads a list of file names from `stdin` and writes its output to `stdout`. The format of `pkgproto`'s output is:

```
ftype  class    path     access   owner    group
```

The meaning of each field is as follows:

- `ftype` is the file type (e.g. `f` for a regular file, `d` for a directory, etc);
- `class` is the entry's class, which will be the default `none` for all files in the example except the sed class script which is of class `sed`;
- `path` is the name of the package object; it is a full pathname, specifying the directory on the installation machine that the object should be installed into. If `path` begins with a slash ("/"), the object is *fixed*, if not the object is *relocatable*. If the objects layout on the development workstation is different from that on the installation workstation, `path` has two components, specifying both locations.
- `access` is the object's UNIX access permissions;
- `owner` and `group` are the files ownership.

`pkgproto` sets all of these for you; however it will set the access and ownership to what they are on the development machine, so you'll probably need to change them.

3.2.1 Hierarchical Directory Structure

From the `pkg` directory, run the `pkgproto` utility as follows:

```
find usr SUNWsst -print | pkgproto > prototype
```

The entries for the control files in `prototype` have a different format, so we'll insert them manually rather than having `pkgproto` create them for us. The output from the above command looks like this:

```
d none usr 0775 pms mts
d none usr/include 0775 pms mts
d none usr/include/sys 0775 pms mts
d none usr/include/sys/scsi 0775 pms mts
d none usr/include/sys/scsi/targets 0775 pms mts
f none usr/include/sys/scsi/targets/sst_def.h 0444 pms mts
d none usr/kernel 0775 pms mts
d none usr/kernel/drv 0775 pms mts
f none usr/kernel/drv/sst 0664 pms mts
f none usr/kernel/drv/sst.conf 0444 pms mts
d none SUNWsst 0775 pms mts
f none SUNWsst/sstest.c 0664 pms mts
```

This needs to be fixed up a little. Entries are not needed for directories that already exist on the installation machine, the access permissions and ownerships need to be changed, and entries must be added for the control files. Finally, a slash must be prepended to the fixed package objects. This is the final `prototype` file:

```
i pkginfo
i postinstall
i preremove
i copyright
e sed /etc/devlink.tab ? ? ?
f none /usr/include/sys/scsi/targets/sst_def.h 0644 bin bin
f none /usr/kernel/drv/sst 0755 root sys
f none /usr/kernel/drv/sst.conf 0644 root sys
d none SUNWsst 0775 root sys
f none SUNWsst/sstest.c 0664 root sys
```

The questions marks in the entry for the `sed` script indicate that the access permissions and ownership of the existing file on the installation machine should not be changed.

3.2.2 Flat Directory Structure

Since the placement of files during installation is not indicated by the development directory structure, it must be specified to `pkgproto`. From `sst`, the directory containing the package objects:

```
find . -print | pkgproto .=/usr/kernel/drv > ../prototype
```

The parameter to `pkgproto`, "`.=/usr/kernel/drv`" says that the objects in the current directory on the development machine are to be installed into the directory `/usr/kernel/drv` on the installation machine. Here's the output from `pkgproto`:

```
d none /usr/kernel/drv 0775 pms mts
f none /usr/kernel/drv/sst=sst 0664 pms mts
f none /usr/kernel/drv/sst.conf=sst.conf 0444 pms mts
f none /usr/kernel/drv/sst_def.h=sst_def.h 0444 pms mts
f none /usr/kernel/drv/sstest.c=sstest.c 0664 pms mts
```

The entries in the `prototype` file specify the locations of the objects on both the development and the installation machines. Note that the source files are relative, we will have to tell `pkgmk` where they are when we run it. This would not be necessary if we had specified "`'pwd'=/usr/kernel/drv`" to `pkgproto` (but the `prototype` entries would be rather long).

The initial `prototype` file must be edited; entries for the control files are added and the access permissions and ownerships need to be changed. Further, the initial `prototype` file shows all files being installed into the same directory; since this is not what we want, the entries for `sst_def.h` and `sstest.c` must be changed. An entry for the `SUNWsst` directory must also be added. The final file looks like this:

```
i pkginfo
i postinstall
i preremove
i copyright
e sed /etc/devlink.tab ? ? ?
f none /usr/kernel/drv/sst=sst 0755 root sys
f none /usr/kernel/drv/sst.conf=sst.conf 0644 root sys
f none /usr/include/sys/scsi/targets/sst_def.h=sst_def.h 0644 bin bin
d none SUNWsst 0755 root sys
f none SUNWsst/sstest.c=sstest.c 0664 root sys
```

The questions marks in the entry for the `sed` script indicate that the access permissions and ownership of the existing file on the installation machine should not be changed.

3.3 Create the Package

Having organized the package objects, written all the scripts, and created the `prototype` file, we are now ready to actually create the package by running the `pkgmk(1)` utility. This reads the `prototype` file and creates a package that can be installed with `pkgadd(1m)`. The syntax for `pkgmk` is slightly different for the flat and hierarchical cases.

3.3.1 Hierarchical Directory Structure

From the `pkg` directory, run `pkgmk` as follows:

```
pkgmk -o -r 'pwd' -d 'pwd'/spool
```

The `-d` parameter specifies the location of the package to be created. This directory, `spool`, must already exist. The `-r` parameter specifies the "root directory" for the package objects on the development machine; its value is prepended to the paths in the `prototype` file. The `-o` parameter allows an existing package to be overwritten. You can safely ignore warnings about missing directory entries for directories that exist already on the installation machine (e.g. `/usr/kernel`).

3.3.2 Flat Directory Structure

In this case, we must specify the location of the package objects on the development machine with the `-b` parameter:

```
pkgmk -o -b `pwd`/sst -d `pwd`/spool -r `pwd`
```

The `-r` option is still needed because the `prototype` entry for the `sed` class script doesn't specify a location on the development machine.

3.3.3 The `pkgmap` File

One of the files created by `pkgmk` is the `pkgmap(4)` file. This is an ASCII file that contains a complete listing of the package contents. Its format is somewhat similar to that of the `prototype` file:

```
part ftype class path mode owner group size cksum modtime
```

The meaning of the field is:

- `part` is the part number of the object. we have only one part in the example, so all objects are in part 1.
- `ftype` is the file type; `e` means a file to be edited, `f` means a regular file, `d` means a directory, and `i` means a control file.
- `class`, `path`, `mode`, `owner` and `group` are all straight from the `prototype` file.
- `size` is the size of the file in bytes.
- `cksum` is a checksum of the file's contents.
- `modtime` is the time of last modification of the file.

The first line of the file contains information about the parts in the package; it consists of a colon followed by the number of parts in the package and the maximum part size (in 512 byte blocks).

Here is the `pkgmap` file for our example package (it's the same for both the hierarchical and the flat directory cases):

```
: 1 120
1 e sed /etc/devlink.tab ? ? ? 218 19258 719022555
1 f none /usr/include/sys/scsi/targets/sst_def.h 0644 bin bin 3623 \
  23380 711071279
1 f none /usr/kernel/drv/sst 0755 root sys 31808 28921 711830351
1 f none /usr/kernel/drv/sst.conf 0644 root sys 326 26818 711830359
1 d none SUNWsst 0775 root sys
1 f none SUNWsst/sstest.c 0664 root sys 3676 19733 711830366
1 i copyright 434 38929 719080369
1 i pkginfo 165 13317 719107352
1 i postinstall 666 55221 719078817
1 i preremove 424 34950 719079244
```

3.3.4 Transfer the Package to Diskette or Tape

The final step in creating a package is to transfer it to a distributable medium, i.e. diskette or tape. The `pkgtrans(1)` utility does this:

```
pkgtrans -s `pwd`/spool /dev/diskette SUNWsst
```

transfers the package `SUNWsst` from the local directory `spool` to a floppy diskette. The `-s` option tells `pkgtrans` to convert from file system format to datastream format. `pkgtrans` supports multiple volumes

4.0 Package Administration

4.1 Adding a Package

The `pkgadd(1m)` utility loads a package onto the system, from the distribution medium or a directory. The typical invocation is quite simple:

```
pkgadd -d /dev/rdiskette SUNWsst
```

installs a package from diskette, using the settings in the package's `pkginfo` file and the installation system's defaults, which are specified in the file `/var/sadm/install/admin/default`. There are two ways to change the settings:

```
pkgadd -d /dev/rdiskette -a adminfile SUNWsst
```

uses the file `/var/sadm/install/admin/`*`adminfile`* instead of the default file. You should always create a new file rather than changing the default administration file. The second way to change the setting is this:

```
pkgadd -d /dev/rdiskette -a none SUNWsst
```

This makes `pkgadd` prompt you for the settings during installation.

You can also spool a package from the distribution medium to a local directory. This enables you to look at the installation scripts before actually running them:

```
pkgadd -d /dev/rdiskette -s /var/tmp/spool SUNWsst
```

This will put an installable version of the package into `/var/tmp/spool` (which must exist before you run `pkgadd`). You can then install the package like this:

```
pkgadd -d /var/tmp/spool SUNWsst
```

4.2 Removing a Package

The `pkgrm(1m)` utility removes a package from the system:

```
pkgrm SUNWsst
```

`pkgrm` also takes the `-a` option to change administration parameter settings.

4.3 Other Package Utilities

4.3.1 pkgparam(1)

This command displays a package's parameter values:

```
pkgparam -v -d /dev/rdiskette SUNWsst
```

The `-v` option tells `pkgparam` to display the parameter names; without it only their values are shown.

4.3.2 pkginfo(1)

This command displays information about installed packages.

4.3.3 pkgchk(1m)

This allows you to check the contents of a package, before or after installation. The utility verifies the integrity of the directory structures and of the files.

5.0 Warnings

5.1 pkgmk(1m) -d option

The manpage for `pkgmk` says that you can give it a device on the `-d` option. This is not the case, for example:

```
pkgmk -d /dev/rdiskette SUNWsst
```

will not work. If you want to create a package directly onto a diskette, create a filesystem on the diskette, mount it, and then specify the directory to `pkgmk`:

```
newfs /dev/rdiskette
mount /dev/rdiskette /mnt
pkgmk -r `pwd` -d /mnt
```

Note that this will limit the package size to a single diskette.

5.2 The sed class script

`pkgrm` does not run the removal part of the script. This is a known bug in Solaris 2.0 that has been fixed in Solaris 2.1. You may need to add a line to the `preremove` script to run `sed` directly to remove the entry from `/etc/devlink.tab`.

5.3 This example is network dumb

It will probably not work correctly if you install it onto a diskless client. In this case you're better off making the whole package relocatable (install all files into `/opt/SUNWsst`), and then copy the necessary files to the right places in the `postinstall` script. Use `installf(1m)` to inform the packaging system about the new files. Don't forget to remove the files in the `preremove` script and also use `removef(1m)`.

6.0 About the Author

Patrick Stirling is an independent contractor. He wrote several of the sample drivers in the SVR4 Migration Kit available from Sun. He was a consultant in Sun's HQ Consulting Group from December 87 through October 90. Prior to that he was employed by Fortune Systems Corporation.

Patrick Stirling
1301 Guerrero St.
San Francisco, CA 94110

Phone/FAX: (415) 824-8737
Email: patrick.stirling@corp.sun.com