SEPP
Software Packaging System

Tobias Oetiker
Department of Electrical Engineering
Swiss Federal Institute of Technology

December 9, 1998

Contents

1 Motivation

The Swiss Federal Institute of Technology has a large installation of Unix workstations. At the Department of Electrical Engineering alone, more than 400 workstations are in use.

Most laboratories in the Electrical Engineering Department have their own system manager and file server. There are about 16 system managers responsible for 12 server. The advantage of this setup is that the management happens where the customer is located. The disadvantage is, that it leads to a multiplication of efforts in respect to software installation and system configuration.

In an education and research environment, the users require a large variety of software packages in their day to day work. These range from little utilities to large applications taking up several GB of disk space. The IT Support Group of the department alone provides software with a total size of over 40 GB.

The IT Support Group of the Department of Electrical Engineering has started the SEPP project with the intention to devise a software installation system which allows close collaboration between the system managers while retaining maximum independence between the different laboratories.

2 Goals

The project had five main goals:

Package Separation Each software package should be installed in its own subdirectory. This helps keeping the system setup clean because it is always clear to which package certain files belong.

Simple to Use Every software package should be immediately available to the user without any explicit setup action. For that purpose the binaries have to be made available in one central `bin` directory and the program itself has to take care of preparing its own environment.

Simple to Maintain A system manager participating in SEPP should not be required to completely alter the setup of her system nor should the creation and installation of new packages be inherently more difficult or time consuming than without SEPP.

Distributed Organization The system should allow to share packages between the participating systems, either by copying them or by accessing them via NFS.

Heterogeneous Systems Although Sun SPARC Solaris is the dominant architecture and OS within the department, the solution should work in a heterogeneous Unix environment as well.

3 Existing Solution

A review of existing solutions found four products, but each only partly achieved to satisfy our needs. All four are based on the idea of keeping software packages in separate directories.

Pack Distribution Project by Peter Krisensen. *Pack* filled many of the requirements, but in the end, the emphasis on the package distribution was too strong. No real support was given for efficiently sharing an installed package via NFS across several systems running off different servers.
http://sunsite.auc.dk/pack/build-package.html

GNU Stow by Bob Glickstein. Stow is a Perl program which basically takes care of efficiently setting up symbolic links in the `/usr/local` tree pointing to directories and files contained in a package directory. While Stow is very sophisticated in doing this, it lacks other important features such as a concept for collaboration between different Stow systems running Stow. The impression was that Stow is designed for consultants who have to setup a workstation, for a customer. The customer wants a number of additional packages, apart from the plain vanilla OS, installed on his computer. Stow has all the means of doing this perfectly without requiring the use of ***pkgadd*** and friends.
http://www.gnu.ai.mit.edu/software/stow/stow.html

The Depot Configuration Management Project by Wallace Colyer and Walter Wong. The Depot Project is the Unix System Configuration Management component of the Workstation Administration/Host Configuration Andrew II project. Although the whole system sounds impressive and extremely well thought trough, the author was put off by its complexity.
http://andrew2.andrew.cmu.edu/depot/

SFI Director by Peter B. Stevens. This is a commercial system management package which also includes application distribution facilities. Director is a complete management solution written mostly in TCL. Some of the concepts like auto-mounting of packages and distributed setup were very interesting. But the overall package is built on the assumption that you either use it completely or you do not use it at all, which was in conflict with the requirements.
http://www.sfi.ch/cstools/appmgmt.html

4 SEPP Features

Building on the goals stated in section 2, a more detailed feature list is presented below.

4.1 User visible Features

The main user visible feature of SEPP is *ease of use*. This means:

- No changes to `.login` or `.cshrc` are required to use any of the applications installed under SEPP, except the addition of `/usr/sepp/bin` to the `PATH` variable. Wrapper scripts do everything necessary to prepare the system for successful execution of the programs the user wants to run.
- Documentation about all the installed packages is provided in a www-ready form and where available also as manual pages. Part of this documentation is generated automatically from information present in each package directory. The SEPP tools will actually refuse to install a package which doesn't have at least minimal documentation.
- Version concurrency is ensured by providing an elaborate naming scheme for application binaries. The user can start the default version of a program by using the plain program name, while older versions are available through *`program-version`*. This means, for example, that Emacs 20.2 gets started by using the command *`emacs`*. But version 19.23 is also available to users who start it with *`emacs-19.23`*. This concept was taken even further, by adding a suffix for the maintainer of the package. If two people are maintaining *`perl-5.004_4`* packages and set them up quite differently, both packages can be installed concurrently on the same system. The end user can pick one of them by using the command *`perl-5.004_4-ew`* or *`perl-5.004_4-gk`* to explicitly pick one version or the other. The system manager on the other hand can decide which package and version is the local default and gets started by typing *`perl`*.

4.2 Making the System Managers happy

Much effort was spent on making life for the system manager as easy as possible:

- Through the use of automounting, all applications appear to be installed under `/usr/pack/package` while the physical location can be anywhere on the local server or a remote machine.
- The SEPP installations on several servers can be linked to provide transparent application sharing. Even mirroring of applications is supported. This allows to fine tune redundancy for each application level. All this is transparent to the end user, as every SEPP package appears to be local.
- Each package is installed in its own directory, keeping it separate from other packages. The only SEPP specific element in this directory is a subdirectory called `SEPP`. The `SEPP` subdirectory contains some text files with meta information about the package and a startup wrapper script. These files have to be edited to fit package.

- A package handling script is provided, which helps to create new packages, to install existing packages on the local server, to build a web site containing information on the installed packages, to remove existing packages, and to generate a list of all packages available within the network.
- Full usage logging for all SEPP applications is provided through *syslog*. This allows to see which applications are actually in use and thus helps in deciding which ones can be uninstalled.
- Even though package directories are shared read-only, an application can be configured differently on each server if this is necessary. SEPP packages can contain symbolic links pointing to `/usr/sepp/var/package` plus a template directory which is copied to this location at installation time. The `/usr/sepp/var/` tree must be exported read-write to all the clients of the local server. This setup allows to install packages like TeTEX which need to write font files to a central directory, or commercial packages which need a local license file. More information on the licensing issue can be found in section 8.5.

5 Implementation

When implementing SEPP, the author tried to work as much as possible with services and utilities which are present on an 'out of the box' Unix. A few external tools like Perl, TC-Shell and GNU cp were used, copies of these programs come with the SEPP base package and are installed in a special directory, together with the SEPP administration tool *seppadm*. The normal users will not note their presence.

The following section shows what happens when a user starts a SEPP application:

1. The user has put `/usr/sepp/bin` into her `PATH` variable and types *gls*[1] into her shell.
2. The shell finds `/usr/sepp/bin/gls` which is a symbolic link pointing to `/usr/sepp/stub/fileutils-3.16-to`.
3. *fileutils-3.16-to* is a tiny Perl script whose purpose is to load and run `/usr/pack/fileutils-3.16-to/SEPP/`*start.pl*, the actual startup wrapper for *gls* and all the other tools in the fileutils package. The intermediate step through the stub file prevents the system from automounting all application directories on every rehash, as the symbolic links do not point directly to the *start.pl* files. The implementation of the stub file uses Perl's *do* command so that the Perl just reads and executes the contents of *start.pl* without an expensive *exec* or *system* call.
4. In *start.pl* the environment is prepared for the execution of *gls*. In the case of *gls* this is a very simple task because *gls* runs "out of the box". Depending on the application it might be necessary to set, for example, `LD_LIBRARY_PATH`, `ELMHOST` or `PATH` to special values to run the application

[1] SEPP does not allow to install applications with names that mask "official" OS binaries. Therefore GNU ls is called *gls*

successfully. Even certain configuration files may have to be copied into the users home directory if the application is started for the first time.

If a package is to be run on different architectures, the startup wrapper has to make sure that it starts the correct binary for the architecture.

5. When the application terminates, the wrapper script will generate a syslog event telling how long that application has been running and what its exit code was. For long running applications like daemons it might not be of interest to know how long they have been running, especially not at the expense of having a Perl interpreter hanging around in swap space all the time. It is therefore also possible to generate the log before the actual application is started, loosing the runtime information, of course.

6 SEPP Setup

Before SEPP can be used on a system, the SEPP base package must be installed and a few files have to be edited. While installing the base package requires root permissons, using SEPP does not. The use of a special user account (local) for installing applications is strongly recommended. Here is a little cookbook[2]:

1. Create a directory called `/usr/sepp`. If using a Solaris autoclient setup, you might want to create the SEPP installation directory in a different location and just put a symbolic link into `/usr` to avoid getting problems with the demand-const mounted caching file system on the client side.

2. Get the latest `sepp-base` and the `sepp.sbin` apropriate for your architecture. Unpack these archives within `/usr/sepp`.

3. Touch the `/usr/sepp/conf/autosepp_indirect` file.

   Solaris: Prepare the automounter by updating `/etc/auto_master` to include the line:

   ```
   +/usr/sepp/conf/autosepp_master
   ```

   This has to be done on the server as well as on each client running of the server. If the `/usr/sepp` tree itself resides on an automounted partition, make sure to run *`automount`* somewhere in `/etc/rc3.d`.

   The `autosepp_master` contains a reference to `autosepp_indirect`. On Solaris, this indirect map has the advantage, that changes to the map become active immediately on all the clients. No automounter reload or restart is required. On other automounter implementations this may have to be solved differently.

   Activate the new configuration by running *`automount -v`*.

   DEC Unix (osf1): Add the contents of `/usr/sepp/conf/autosepp_master` to your `/etc/auto_master` and restart the *`automountd`*.

   (please mail me if you have a more verbose explanation for what is necessary to get SEPP to work on DEC OSF1)

[2] As SEPP is being developed in a Solaris Environment this cookbook is partly Solaris specific. Systems like the automounter or autoclient may operate differently on other Unixes. Also the three binaries *`perl`*, *`gls`* and *`tcsh`* will have to be replaced on a different architecture. As these programs are widely available this should not pose a problem.

4. Update `/etc/syslog.conf` on the server to enable SEPP logging.

```
# write SEPP log to a local file
local4.info      /var/log/sepp.log
# send log events to a central logserver
local4.info      @sepp.ee.ethz.ch
```

   You can write the log entries to a local file, but more efficiently you send them to your local server and there forward them to the central SEPP logging server. In our local setup, the central logging server for SEPP is called sepp.ee.ethz.ch. The central logging allows collection of statistical information on application usage across the whole network.

   Add the following line to your clients `/etc/syslog.conf` to make them forward their SEPP syslog events to your local server.

```
# send log events to the local server
local4.info      @local-server
```

5. Customize `/usr/sepp/conf/sepprc.system` to reflect your local setup. The `seppdomain` variable can be set to whatever you think is a descriptive name for the computers running of your local server. Please check appendix B.1 for an example of this file.

6. Edit `/usr/sepp/conf/sepp.conf` to match your system. There are comments in the file explaining the individual entities. In appendix B.2 there is an example `sepp.conf` file. Make sure to do a *`touch`* on your future pack-list, as *`seppadm`* will otherwise complain when you try to run it.

7. Finally make the directory `/usr/sepp/html` available on your web site. As this directory will contain documentation about all your SEPP packages. Once you have some packages installed you can run *`seppadm webbuild`* to populate this directory. You have to run *`seppadm webbuild`* every time new applications get installed, to update SEPP WebPage.

7 Using *seppadm*

Making a new application available to the end user on a SEPP system is a two step process: The first step is called "package preparation" the second stage is called "package installation". The preparation step takes some extra work, compared to a normal application installation because SEPP requires to write a startup wrapper and to provide some documentation. The install step on the other hand is very simple.

If several system managers have linked their SEPP installations, every application available on any of the participating servers can be installed locally with a single command, regardless on which server it is physically located.

To make using SEPP as simple as possible the *`seppadm`* tool is provided. *`seppadm`* is a Perl script, with a number of functions which help the system manager to administrate the SEPP installation. The following sections gives a short introduction about how to use the *`seppadm`* script. Appendix A shows the complete manual page of this tool, explaining all the options.

7.1 Preparing a SEPP package

This section gives an illustration of the package preparation process based on the installation of the GNU File Utilities. First the directory for the package has to be prepared. This is done with

```
seppadm prepare fileutils-3.16-to
```

which picks the file system with the most free space, creates a directory and configures the automount map to make this directory available as as `/usr/pack/fileutils-3.16-to`.

The name of a SEPP package is made up from 3 components which are separeted by a dashes: `application-version-maintainer`

`application` is the name of the application which is to installed as a package. (emacs, gimp, fileutils, ...)

`version` is the version number of the application. (22.1, 0.99b2, ...)

`maintainer` stands for the initials of the package maintainer. (to, ak, fz, ...)

When *seppadm prepare* has setup the directory it starts an application install environment. This is a *tcsh* providing a controlled environment, so that the installed package does not unintentionally tie itself to some package which is installed locally, but might not be available on another system. Section 8 has some additional hints on this subject. The fileutils package should be compiled and installed with the installation root set to `/usr/pack/fileutils-3.16-to`.

After compiling and installing the GNU file utilities, the SEPP specific files in `/usr/pack/fileutils-3.16-to/SEPP` must be adapted to the package. The *seppadm* tool has already placed the following template files into the `SEPP` directory:

`INSTALL`: free form transcript of what was necessary to get the package to compile and install properly. Appendix C.1 gives an example for GNU File Utilities.

`CHANGES`: list of all the modifications done to a package after it was first installed. See appendix C.2 for an example.

`README`: brief description of what the package does. The file has to be in a defined format because it gets converted to html for inclusion into the SEPP Application Catalog. In appendix C.3 is a copy of the `README` template file which contains instructions how to write a README file in the format required by SEPP.

`start.pl`: startup wrapper for the package. It is responsible for preparing an adequate environment for running the application. For well behaved applications this script is very simple, as all it has to do is to start the binary. For other applications it might be necessary to set certain environment variables or copy startup files to the users home directory. The template `start.pl` in appendix C.5 explains the basics needed to know when writing startup wrappers. The startup wrappers have to be written in Perl. To make this process as simple as possible a few special Perl functions for "SEPP startup wrapper writing" are available in these scripts. See appendix D for a manual page explaining these commands in detail.

`patch`: directory for storing patch files. If extensive modifications are necessary to certain parts of the package in order to get it to work, the necessary patches should be copied into this directory.

`META`: file where *seppadm* expects to finds a meta information about the package. The version number, related URLs, the email address of the package maintainer, a one line description of the package and information about which binaries should be available to the end user. In appendix C.4 is a copy of the standard `META` template.

7.2 Installing a SEPP package

Once a package is prepared, the actual SEPP installation process is very simple:

```
seppadm install fileutils-3.16-to
```

Everyone who has `/usr/sepp/bin` in the `PATH` should now be able to use the gnu file utilities.

Note, that you might have to use the option *--default* if this is a new version of a package you have already installed. Don't forget to update your SEPP website with *seppadm webbuild*.

8 Package compilation tips

Installing a package can be tricky sometimes. Above all one has to avoid system dependencies. Packages should run even when they are started on a system which has no other software available besides the SEPP package and a regular OS installation. The following tips are somewhat Solaris specific because this is the main platform used by the author. The basic principal will be the same on any other system.

8.1 Use only standard system binaries

Some packages use external utilities when they run. If the package comes from the Linux world, it is likely that it expects to find the GNU versions of the utilities. If the GNU version of a utility happens to be available on the local system, maybe even installed under the same name as the original version from the vendor, the package will run quite happily. But when the application started on another host with stock vendor utilities, things will fail badly.

One step to prevent such problems is the install environment provided by *seppadm*. It changes the `PATH` so that the directories `/usr/bin`, `/usr/sbin` and friends come first. If the package looks for certain utilities at install time and then stores their location with absolute pathnames, it will find the system default binaries and use these. If the package insists on using the GNU variant of a utility, it should be copied into the package. One of the guiding principals here should be that disk space is not an issue.

If a large external package is required, it should be provided as a separate SEPP package and then listed in the `META` file of the package that requires it.

8.2 Shared library search path

Modern applications often use shared libraries. These libraries are linked to the application binary at startup time. If the loader can not find the shared library at startup time the application will not be able to run. There are two ways to inform the loader where to find the libraries. The preferred one is to hard code the location of the library files into the application binary[3]. This can be achieved by specifying the shared library directory with the `-R` option on the compiler command line.

```
gcc -R/usr/pack/gimp-0.99.27-to/lib ...
```

Instead of giving the -R option on the command line, the `LD_RUN_PATH` environment variable can be set to point to the library location.

For precompiled binaries which can not find their libraries, the environment variable `LD_LIBRARY_PATH` must be set in the `start.pl` script of the application.

The command *ldd* can be used to debug problems with shared libraries because it can list all the libraries required by a binary.

8.3 Avoid name clashes

When using *seppadm* to install a package, it will automatically check that the names of the installed programs do not clash with any of the binaries installed in the default OS binary directories and with binaries from other SEPP packages.

8.4 *configure*

Today, many programs are quite easy to install thanks to GNU autoconf. Nevertheless there are a few points to observe. Because autoconf can lead to application binaries which are much too tightly bound to one system to run reliably on another system. Before *configure* is started, a few environment variables can be set to guide *configure* in the right direction.

- If the package needs special include files and libraries or compiler and linker flags, those can be defined by setting the `CPPFLAGS`, `CFLAGS` and `LDFLAGS` environment variables.

  ```
  setenv CPPFLAGS "-I/usr/pack/gtk-1.0.0-to/include"
  setenv CFLAGS "-O2"
  setenv LDFLAGS "-L/usr/pack/gtk-1.0.0-to/lib \
    -R/usr/pack/gtk-1.0.0-to/lib"
  ```

- The compiler can be chosen by setting the `CC` or `CXX` variable:

  ```
  setenv CC egcc; setenv CXX eg++
  ```

Further configuration of *configure* is possible by specifying command line switches ...

[3] SEPP packages will always be installed under the `/usr/pack` path which makes this hard coding very convenient

- Every package in SEPP should be installed under a separate subdirectory tree. The option `--prefix=/usr/pack/package` of *`./configure`* defines this.

- GNU applications often duplicate the functionality of "official" system utilities. By default they even use the same names for their binaries as the vendors do for theirs. *`seppadm`* will refuse to install such programs as explained before. With the option `--exec-prefix=g` the *`configure`* program will create Makefiles which prepend the letter 'g' to the normal name of each binary in the package. This makes *`gls`* out of *`ls`*.

- If X11R6 is installed on the system in addition to the X11 setup provided by the vendor. It should be ensured that the SEPP pack uses the vendor libraries[4] and not the local X11R6 installation as it can not be assumed that X11R6 is installed on every system. So far we have not found a program which strictly required X11R6 libraries. To make sure *`configure`* uses the vendors X11 libraries the `--x-includes` and `--x-libraries` parameters can be used. In a Solaris environment they would be set to `--x-includes=/usr/openwin/include` and `--x-libraries=/usr/openwin/lib`

8.5 Access Control

If an application requiring a license file or license number is installed as a SEPP package, this information should *not* be included in the actual package directory. The SEPP package might be shared with people who do not share the same licenses. Most importantly no environment pointing to a license server, should be set in the *`start.pl`* file of the package, as this file will be visible on the local SEPP Application Catalog web site.

This problem is solved by using the `vartmpl` parameter in the `META` file of the package to setup a template directory containing a dummy license file or a little Perl fragment containing the name of the license server. When the application is installed, the contents of this template directory is copied into the `/usr/sepp/var/` tree. The local system manager can then edit these files to contain real data. For environment variables, a line like

```
do "$PackVar/licvar.pl";
```

can be used in the *`start.pl`* script. The *`licvar.pl`* file would then contain the code necessary to set the license server environment variable. If the application expects to find a license file somewhere inside its own directory tree, this file can be replaced by a symbolic link pointing to a location in the `/usr/sepp/var` tree.

A The *seppadm* Manual Page

All SEPP administration tasks can be performed with the *`seppadm`* utility. The manual page below explains in detail the functions of this tool.

[4] `/usr/openwin/` in the case of Solaris

```
NAME
    seppadm - SEPP package administration tool

SYNOPSIS
    seppadm prepare [pathname/]package [--noshell]
    seppadm install package [--default]

    seppadm remove package [--final]

    seppadm mirror package

    seppadm mirrorupdate

    seppadm report [--datesort]

    seppadm webbuild

DESCRIPTION
    The seppadm command is used to administer software
    packages within the SEPP system. Its primary functions
    are: installing and removing packages, preparing
    directories for package installation, mirroring packages
    from other servers, list available packages, maintain
    the SEPP Application Catalog web site with information
    on the available packages and keeping the local mirrors
    up to date.

    prepare [pathname/]package
              Creates a skeleton package directory. seppadm
              will automatically choose the filesystem with
              the most free space. By specifying a pathname,
              this automatic choice can be overridden.

              The package name must have the format
              application-version-maintainer. The package
              directory will become visible as
              /usr/pack/package. This directory will
              contain one subdirectory called SEPP which
              contains a number of template files. These
              templates have to be edited to fit the
              package.

              When the prepare function completes, it will
              execute an 'install shell' which provides a
              controlled environment for application
              compilation and installation.

    install package [--default]
              Installing a package means two things: First
              the binaries get linked into the /usr/sepp/bin
```

```
            directory and second, the application gets
            listed in your packlist and can therefore be
            installed on cooperating systems.

    remove package [--final]
            Removes a package from the system. Without the
            option --final the package does not get
            removed for real. It just gets earmarked, so
            that people who still use the package get a
            warning, telling them that the package is
            going to be removed soon. The option --final
            removes the package for real.

    mirror package
            Create a local copy of a package. This is to
            increase reliability and performance.

    mirrorupdate
            Verifies that all local mirrors are up to
            date.

    report  [--datesort]
            Generates a list of available packages using
            information from all linked servers.

    webbuild  regenerates the local web site.

BUGS
    No Idea ... But if you tell me I'll fix 'em.

AUTHOR
    Tobias Oetiker <oetiker@ee.ethz.ch>
```

B SEPP System File Examples

SEPP is configured via several text files. Some are global and some are specific to each SEPP package.

B.1 /usr/sepp/conf/sepprc.system

This file gets read whenever a SEPP application is started. Its contents can be accessed in the startup wrapper script.

```
##################################################################
# Global config file for all sepp wrapper scripts. The values
# defined in here are available in the start.pl wrapper scripts
# to allow the proper configuration of some values which are
# not easily determined otherwhise. See the SeppStart manpage
```

```
# for more information on this.
#####################################################################

# Used as a marker in sepp syslog events. Should identify
# local installation.
$seppdomain = 'isg-net';

# general configuration values which can be used from
# start.pl files by prepending $CF:: to the variable name
# for ex: $CF::maildomain.

$maildomain = 'stud.ee.ethz.ch';
$inetdomain = 'ee.ethz.ch';
$httpproxy  = 'proxy.ee.ethz.ch:3128';
$ftpproxy  = 'proxy.ee.ethz.ch:3128';
$popserver = 'pop.stud.ee.ethz.ch';
$imapserver = 'imap.stud.ee.ethz.ch';
$smtpserver = 'smtp.stud.ee.ethz.ch';
```

B.2 /usr/sepp/conf/sepp.conf

This is the main configuration file for *seppadm*.

```
#####################################################################
# Main configuration file for the SEPP installation
# It is read by seppadm
#####################################################################

*** local server ***
# the name of the local sever and the path to the local
# packlist. The packlist is a file which is maintained by the
# seppadm tool. It contains information about all the packages
# available on this server

drwho.ee.ethz.ch /usr/drwho/pack-b/SEPP.packlist

*** remote server ***
# seppadm can mirror and install packages from remote servers.
# If a package is available from multiple servers, the auto-mounter
# will be configured to automatically pick the best available
# machine. The number in braces after the host name defines the
# preference of the host. The lower the number the higher the
# preference. The packlists of the remote servers must be
# mounted locally. Seppadm will NOT automount them ...

nariworkserv.ethz.ch(3) /net/nariworkserv.ethz.ch/export/pack/SEPP.packlist
#min.ethz.ch(2) /net/min.ethz.ch/export/pack/SEPP.packlist
lunghin.ee.ethz.ch(2) /usr/lunghin/r2/pack/SEPP.packlist
goomba.ethz.ch(2) /usr/goomba/local2/sepp/SEPP.packlist
```

```
*** preferred operating system ***
# seppadm will only let you install packages which are
# available for the OS configured on this line.
solaris

*** sepp user ***
# seppadm will only run as this user. If you run seppadm as root,
# it will try to become this user. It is sensible NOT run run seppadm
# as root. And this runeable ensures this ...

local

*** sepp disks ***
# when preparing a package, seppadm tool will automatically
# pick one of the disks listed below, choosing the one with the
# most space available.

/usr/drwho/pack-a
/usr/drwho/pack-b
/usr/drwho/pack-c
/usr/drwho/pack-d

*** compile shell ***
# The application preparation environment is configured to use
# tcsh but for strong minded people this can be configured
# here.
/usr/sepp/sbin/tcsh

*** compile env ***
# these lines define the environment variables which will be
# set when the compile shell is started. In the example below
# the HOME variable gets set to /usr/sepp/conf where a .cshrc
# file is located which will be sourced when the shell starts.
#
# The lines below may contain special keywords in curly braces.
# these will be replaced by their real values when seppadm
# starts the shell. {SEPP} = /usr/sepp, {PATH} = $PATH,
# {HOME} = $HOME {PACKDIR} = /usr/pack/app-vers-maint

OPATH=/usr/bin:/usr/ccs/bin:/usr/openwin/bin:/usr/sbin:\
      /usr/sepp/bin:/usr/sepp/sbin:{PATH}
ORPATH =/usr/openwin/bin
OHOME={HOME}
HOME={SEPP}/conf
PACK={PACKDIR}

##################################################################
```

C Package Config Files

Every SEPP package contains a `SEPP` subdirectory. In this directory, a few files must be present. Templates for these files are copied to the `SEPP` directory by *seppadm*.

C.1 /usr/pack/fileutils-3.16-to/SEPP/INSTALL

This is an example install transcript file. The example file is from the GNU file utilities mentioned earlier in this paper.

```
seppadm prepare fileutils-3.16-to

* get source and unpack

* because fileutils contains tools which have the same
  name as standard system tools (ls,mv,...) I used the
  --program-prefix=g option. It prepends the letter g
  to all executable names (gls,gmv,...) and therefor prevents
  naming clashes.

./configure --prefix=/usr/pack/fileutils-3.16-to \
            --program-prefix=g --disable-nls

gmake
gmake install

* update the files in /usr/pack/fileutils-3.16-to/SEPP

seppadm install fileutils-3.16-to
```

C.2 /usr/pack/fileutils-3.16-to/SEPP/CHANGES

Every change made to the package after it is installed should be documented in this file. Apart from telling what changes have been made, the `CHANGES` file is also used to determine if a mirror has to be updated or not.

```
*** history ***
1998-03-13 Package Prepared
```

C.3 /usr/sepp/conf/template/README

The text of this file will show up in the SEPP Application Catalog web site.

```
## REMOVE THIS LINE WHEN YOU HAVE EDITED THIS FILE ############

############################
# Description of the package
############################

Type a longer description of the package and possibly some
```

```
information for new users on how to get started with the
program and how further help can be obtained.

In order to make it simple to parse this text please follow
these guidelines when writing the README file.

* separate paragraphs by one empty line.
* lists should look like this
* to _emphasize_ a word add a underscore to the begin and
  the end of the word.

1.) enumerated lists look
2.) like this

# lines starting with a hash are comments and will be ignored.

Mark |commands| with vertical bares. They force
a typewriter font when the README is converted to HTML.

#####################################################################
```

C.4 /usr/sepp/conf/template/META

When a package gets installed, removed or documented on the website, *seppadm* needs some to know some details about the package. This information is stored in the `META` file.

```
## REMOVE THIS LINE WHEN YOU HAVE EDITED THIS FILE ############
#
#####################################################################
# The META file contains structured information about the
# package. It is read by seppadm whenever some operation is
# performed on this package.
#####################################################################
#
# all the values shown below are dummy entries which should
# explain what to write into the fields they must be
# adapted to fit the installed package. Lines starting with a
# hash (#) are ignored by seppadm.
#
*** package name ***
The full name of the package
#
*** package version ***
version.number
#
*** one line description ***
type a ONE LINE description of the package (max 60 char)
#
*** maintainer ***
```

```
Email Address <of@the.maintainer>
#
*** local support ***
Address Of <local@person.who.can.help.with.this.package>
Address Of <other@local.person.who.helps>
#
*** license ***
# pick one of the following licensing schemes
#
# Unlimited use within the organization (site license)
unlimited
#
# Limited number of licenses available
limited
#
# This product is available for free. It does not mean
# that this is an Open Source product, only that no licensing
# restricts the distribution of this product through SEPP.
free
#
*** license contact ***
Who to contact <if@the.licenses.run.out>
#
*** urls ***
# Important urls for this package - tutorials, home page. This
# information will be listed in the SEPP Application List
# web site.
<A HREF="http://catholic.heaven.org/~john.pope">Homepage</A>
<A HREF="ftp://no.no.no.thatcher.uk/no/demopack">FTP Home</A>
#
*** operating system ***
# which OSes are supported by the package. The examples given below are
# the minimal requirement, but perferably you should enter the exact
# type of os you are runnning. Eg: 'solaris-2.5.1 sparc sun4u' or
# 'hpux-10.20 9000 735'
sunos
solaris
hpux-9
hpux-10
aix-3
aix-4
linux
ofs1
irix-5
irix-6
irix64
#
*** categories ***
# pick the categories that apply to the package
system
```

```
sci&eng
programming
internet
graphics
multimedia
fun
text
office
#
*** binaries ***
# list of binaries provided by the package. The file name part
# may contain a regular expression the path part is relative to
# the installation root of the package
bin/demo.+
# normally SEPP starts its apps through a wrapper script. With
# interpreter languages, which are referenced in
# #!/usr/sepp/bin/lang scripts-headers this does not work. Use
# '>>' as a prefix to make SEPP omit the wrapper layer for
# certain binaries.
>>bin/perl
#
*** manpages ***
# the same procedure as for binaries applies to manpages
man/man1/.+
man/man5/.+

*** vartmpl ***
# the contents of the specified directory will be copied to
# /usr/sepp/var/packname-ver.sion-maint. The package has to
# take care that symbolic links point to this directory where
# appropriate.
template
#
*** html doc root ***
# this directory will be linked with a symlink into the
# /usr/sepp/html tree. It must at least contain an index.html
# file
html
#
*** depends on ***
# when installing this package, seppadm will verify that the
# packages mentioned in this section are also installed on the
# local system. Package dependencies should be prevented
# whenever possible.
miracles-1.45.2-ak
#
*** alarm ***
# the alarm feature (not implemented) will make sepp send a
# mail to the person mentioned at the date specified. This can
# be used to notify the maintainer of the package about a
```

```
# license which will soon expire ...
YYYY-MM-DD email address <to@send.mail.to> contents of mail
#
#################################################################
```

C.5 /usr/sepp/conf/template/start.pl

The App* and Env* commands shown in this template are not normal Perl commands. They are only available in SEPP startup scripts!

```
## REMOVE THIS LINE WHEN YOU HAVE EDITED THIS FILE ############

# The start.pl file is written in Perl. Apart from standard
# Perl you can also use the additional commands provided by
# SeppStart.pm. Read 'man SeppStart' for detailed information.

# The variable $Pack contains the SEPP name of the current
# package. $PackDir is the pathname to the installation
# directory of the current package and $PackVar is the path to
# the SEPP var directory of the package if such directory
# exists.

## DO NOT put things like license server names into this file
## as its contents will be available on the SEPP web site. The
## same holds true for all other files in the SEPP directory.

# Fix the contents of some well known environment variables.
# These lines are only examples of what could be done. A
# well behaved application does not need any of these ...
# PreENV prepends the arguments to the contents of the
# environment variable and separates it with a ':'

#PreENV "PATH",      "/usr/bin";
#PreENV "XFILESEARCHPATH",   "$PackDir/xresources/%N";
#PreENV "LD_LIBRARY_PATH",   "$PackDir/lib";

# The SetENV command (re)defines the value of an environment
# variable.

#SetENV "LD_LICENSE_FILE",   "$PackVar/license.dat";

# Here comes the heart of the start.pl script
# ------------------------------------------

# One of the two App* commands has to stand at the end of the
# script. It starts the application binary which the user
# wanted to run in the first place. The argument after the App*
# command has to point to the directory where the application
# binaries are stored.
#
```

```
# The difference between AppRun and AppExec is that AppRun uses
# system to start the application and AppExec uses exec. While
# AppRun lets the wrapper wait until the application ends,
# AppExec just execs the application, replacing the wrapper
# job.
#
# Because AppRun waits until the application terminates, it can
# then write an entry to the SEPP log, telling how long the
# application has been running and what it's exit code was.
# AppExec just logs the fact that the application is going to
# be started.

AppRun "$PackDir/bin";
# AppExec "$PackDir/bin";

#####################################################################
```

D SeppStart.pm Manual Page

As mentioned above there are a few special commands available when writing start.pl wrappers. This is the manual page explaining these commands.

```
NAME
    SeppStart.pm - SEPP startup wrapper Module

SYNOPSIS
    PreENV EnvVar, Value, Value ...

    SetENV EnvVar, Value

    AppRun BinaryPath

    AppExec BinaryPath

    $Pack, $PackDir, $PackVar

DESCRIPTION
    This module provides a number of functions for creating
    SEPP/start.pl wrappers.

    PreENV EnvVar, Value, Value ...
              Prepends the Value to the current contents
              of EnvVar, using the ':' as a separator.
              'PreENV "PATH", "/usr/sbin"'

    SetENV EnvVar, Value
              Set EnvVar to Value

    AppRun BinaryPath
```

Run the application specified through the contents of $0. The argument must point to the directory where the application binaries are installed inside the pack. This command should be used at the end of every start.pl script to launch the actual application. It will also write an entry to the sepp syslog facility local_4 when the application terminates. Giving details on runtime and exitcode of the application.

AppExec BinaryPath
Works the same as AppRun, but the application is started via exec. This takes less memory, but the log entry will neither contain the runtime nor the exitcode of the application.

$Pack, $PackDir, $PackVar
These three variables can be used in the start.pl file to simplify to make the wrappers portable ...

BUGS
No Idea ... But if you tell me I'll fix 'em.

AUTHOR
Tobias Oetiker <oetiker@ee.ethz.ch>