Build OpenSolaris Packages Using pkgbuild/pkgtool

Author : Shivakumar GN

Document Version 0.2, 05-Aug-2007

Released under Creative Commons Attribution License

*(*http://creativecommons.org/licenses/by/3.0*)*

Table of Contents

0.About the document

The objective of this document is to enable a user of OpenSolaris based distro (eg: Belenix, Solaris Express, etc) to create software packages for their platform.

After going through this document, the users will be able create SVR4 packages by authoring spec files for the software.

Basic user level experience on a OpenSolaris system or Linux system is assumed.

The document is intended to be as simple as possbile.

For any technical queries related to pkgtool/pkgbuild, direct your queries to *desktop-discuss at opensolaris dot org*.

Any feedback about this document may be sent to the author: *shivakumar dot gn at gmail dot com*

The Appendices have supplementary details of tools/topics that are often encountered when building, creating packages (not all appendices are currently complete in the document).

Disclaimer : This documentation is for information only and the responsibility of usage lies solely with the user of the document.

Change History:

Version	Date	Description
0.1	31-July-2007	Draft version. Used in a local forum. Not released online.
0.2	05-Aug-2007	Updated with comments from Mr.Laszlo Peter (the author of pkgbuild/pkgtool)

1.Background information to pkgbuild

Package management system makes the tasks such as installation/uninstallation/upgrade of the software relatively easy to manage.

A typical source code goes through the compiling/linking stage for binaries to get created and then through packaging stage for the packages to get created. (Packages can be either for sources or for binaries)

```
Source_Code --building--> Binaries --Packaging--> Software_Packages
```

A easy mechanism to perform the above sequence of things would be nice to have. The requirements such a mechanism should satisfy are:

- Creation packages starting from pristine sources
- Package creation is repeatable in a consistent manner (success guaranteed every time)

RPM is one such packaging mechanism that has been very popular in the GNU/Linux world. Many of the linux installables are distributed in the form of rpm packages.

The rpm packages are created using the **rpmbuild** tool. To create a package, the developer authors a **.spec** file having information about source-code location, build instructions, deployment instructions, etc and then uses this spec as an input to the **rpmbuild** to create the rpm packages. Owing to rpm's popularity, many of the open source software is available along with rpm spec files.

SVR4 packaging system is the likely packaging system available as default on an OpenSolaris based distro. If a tool that can use the rpm spec files and create a SVR4 packages were to be available, it would make it possible to create SVR4 packages for the wide range of softwares that already have rpm spec files. Even for those for which rpm specs do not exist, a spec file can be easily written from scratch using the well known approach.

The **pkgbuild** is one such tool. It is a drop in replacement for rpmbuild. This tool is available as part of the JDS CBE[1] (JDS Common Build Environment). In addition to **pkgbuild** JDS_CBE also provides another tool **pkgtool** that uses pkgbuild and extends the capability

- to get sources from ftp/http sites if not available on the system (in the local repository)
- to install the packages onto the system after package creation

This document describes the use of **pkgtool** & **pkgbuild** for SVR4 package creation.

Note that while having an rpm spec file for the source code to start with is useful, it is not mandatory. One can write a spec file from scratch and use it with pkgtool/pkgbuild.

Many people have already authored many spec files (**over 400 as of date**) and these are available at http://pkgbuild.sourceforge.net/spec-files-extra. The contributed files maybe downloaded via subversion access

```
svn co https://pkgbuild.svn.sourceforge.net/svnroot/pkgbuild/spec-files-extra/trunk SFE
```

Make use of the already contributed spec files for your purpose

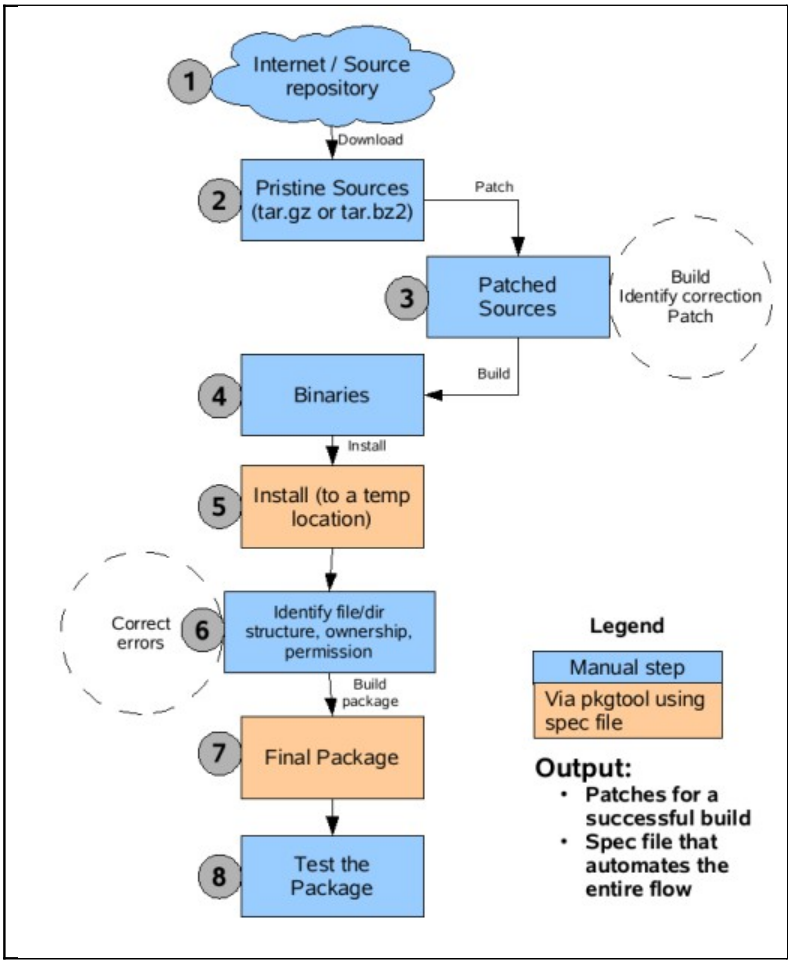
Also as and when you author spec files for your favourite softwares consider contributing them to this repository.

1 *Available at http://www.opensolaris.org/os/community/desktop/communities/jds/building*

2.Stages in Package Creation

The below diagram shows the stages involved in creating a software package.

A developer needs to perform these stages manually. By the end of stage (6) the developer should be having 2 outputs available, the set of patches required for successfully building and a spec file with instructions that can help create a package from the sources.



- Identify the source code location (http/ ftp)
- Download the sources
- Attempt building the source. If build fails, make necessary corrections and repeat build till success. Create patches based on the code change required for successful build
- Author the spec file that can perform stages (1) to (4)
- Based on instructions to install the software, include install instructions also into the spec file (stage 5)
- Use the pkgtool with the spec to perform stages (1) to (5)
- Go through the directory structure of the installed dir, and identify the final set of files/dirs along with required ownership. Include these instructions into the spec file
- Using the spec file, create a package.
- Test the package by installing it and verify the files/dirs/permissions etc after installation.
- Incase of any errors re-visit stage 6 and repeat the exercise

The stages that take up the effort is shown in the form of 2 circles in the figure above

- Creating binaries (stage 2 to stage 4): One may encounter problems due to various reasons while building the application. Suitable code corrections need to be made and build is to be re-attempted. This cycle of “identifying corrections - patching – building” takes up bulk of the effort among all the stages
- Identifying files/dirs & ownership/permissions: Here again there is a small iteration possible if the spec author doesn’t get the details right the first time.

The above diagram shows the steps involved in creating a software package.

3.SPEC file structure

The spec file is the artifact that has all the necessary information for pkgtool to create a binary package from the sources.

The example spec file below provides the information about the structure of a spec file. It is for the Geany software. It has annotations marked in blue color. Note the point below before going through

the spec file structure.

Note :

The below example is a monolithic spec file to illustrate the structure of the spec file only.

This is not to be considered to represent a good way to organize a spec file. Refer to the appendix section “Spec file authoring – conventions & good practices” for some pointers

For an example of a good spec file, take a look at the ones at the SFE repository (https://pkgbuild.svn.sourceforge.net/svnroot/pkgbuild/spec-files-extra/trunk)

```
# ============================================================
#                     Spec File for Geany
# ============================================================

# ------------------------------------------------------------
# Custom variable definitions
# ------------------------------------------------------------
%define  _basedir   /usr/local
%define  _prefix    %{_basedir}
%define  _bindir    %{_prefix}/bin

%define  name       geany
%define  version     0.11
%define  release     1

%include Solaris.inc

# %{_topdir} is by default set to RPM_BUILD_ROOT
# Default path for RPM_BUILD_ROOT is /var/tmp/pkgbuild-{username}
# Install the software here as part of package building steps

# ============================================================
#                     SVR4 required tag definitions
# ============================================================

SUNW_BaseDir:     %{_basedir}
SUNW_Pkg:         SFEgeany
SUNW_ProdName:    SUNW_Geany
SUNW_ProdVers:    0.11
SUNW_Category:    Development,Editors,IDE
SUNW_HotLine:     geany-subscribe@uvena.de
SUNW_MaxInst:     1
SUNW_PkgList:     0
SUNW_Rev:         1
#SUNW_Loc
#SUNW_Copyright:
#SUNW_PkgType:    usr

# ------------------------------------------------------------
# Preamble
# ------------------------------------------------------------
Name:         %{name}
Summary:      Geany - Light weight text editor/IDE
Version:      %{version}
Release:      %{release}
License:      GPLv2
Group:        Development/Tools
Source:       http://files.uvena.de/geany/%{name}-%{version}.tar.bz2
Patch:        geany-0.11-01-solaris.sunstudio12.diff
Vendor:       Enrico Troger,Nick Treleaven,Frank Lanitz
URL:          http://geany.uvena.de/
Packager:     Enrico Troger
BuildRoot:    %{_tmppath}/%{name}-%{version}-build
```

Comments: Lines starting with “#” symbol
 # This is a comment

Variables:
To be used in the subsequent part of the spec file. **These are typically from a Solaris.inc include file** rather than putting directly in the spec file

Include files:
%include files may contain variable/tag definitions include files are picked from <PKG_ROOT_DIR>/include directory

Tags:
<TagName>:<TagValue>
Name: geany

Some tags are exclusive to SVR4 and not present in rpm. SVR4 specific tags are as a convention made part of **Solaris.inc** include file instead of directly including in the spec fil

```
#Requires:   <-- Provide package dependency information (for installation)
#BuildRequires:

%description                <-- multiline description tag
Geany is a small and fast editor with basic features of an
integrated development environment.

# stage 1 relevant details end here
# ---------------------------------------------------------------------
# Prep-Section
# ---------------------------------------------------------------------
%prep        <--%prep makes copying of tarball, unzip/untar possible
%setup -q
./configure

%patch0 -p 1
# stage 3 relevant details end here

# ---------------------------------------------------------------------
# Build-Section
# ---------------------------------------------------------------------
%build
make
# stage 4 relevant details end here
# ---------------------------------------------------------------------
# Install-Section
# ---------------------------------------------------------------------
%install
make install DESTDIR=$RPM_BUILD_ROOT
# stage 5 relevant details end here
# clean = cleanup earlier files before repeating operation
%clean
rm -rf $RPM_BUILD_ROOT

%files
%defattr(-,root,bin)
%dir %attr (0755, root, bin) %{_bindir}
%{_bindir}/*

%dir %attr (0755, root, sys) %{_datadir}
%{_datadir}/doc
%{_datadir}/geany
%{_datadir}/locale
%{_datadir}/man
%{_datadir}/pixmaps

%dir %attr (0755, root, other) %{_datadir}/applications
%{_datadir}/applications/*

# stage 6 relevant details end here
%changelog
* 2007.06.25 - <shivakumar dot gn at gmail dot com>
- Initial spec

# ====================================================================
# CONGRATS : SPEC FILE IS NOW READY
# ====================================================================
```

%prep makes copying of tarball and unzip/untar possible

%setup is to perform any steps required before the actual compile/make

%patch<n> is to apply the set of patches

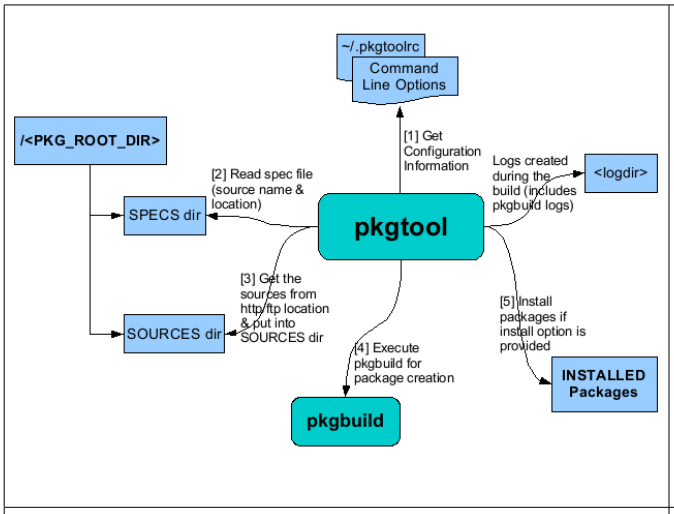
%build as the name suggests is for compiling/linking, creating binaries

%install is for installing the software in a temp location specified by RPM_BUILD_ROOT

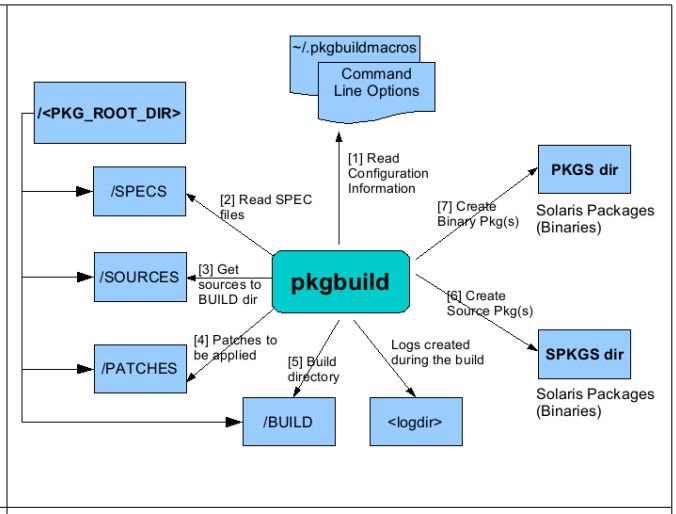
%files section identifies the dirs/files to be packaged, their final ownership/permissions and destination directories

%changelog section is for maintaining the change log history information

4.Working of pkgtool/pkgbuild



4.1 Working of pkgtool



4.2 Working of pkgbuild

The **working steps of pkgtool** are numbered [1] to [5] in the order of occurance in figure 4.1

1. Reads the configuration rc file (~/.pkgtoolrc file) for the pkgtool environment settings. Command line parameters maybe specified to over ride the entries in the rc file
2. Reads the provided spec file (to get the name of the source tarball and the http/ftp location)
3. Checks for the existence of source tarballs (tar.gz or tar.bz2) in the local sources directory, if not present, downloads the sources from the http/ftp location as specified in spec file (uses wget internally for the download). Verifies existence of the patches. Invokes pkgbuild tool once sources are copied. Subsequent steps are performed by pkgbuild tool.
4. Executes pkgbuild tool. The pkgbuild utility creates the source and binary packages. Creates the log file at /tmp
5. If the option to install was specified when invoking pkgtool, then the tool installs the created binary package(s)

Throughout the above steps, pkgtool maintains a log file in logdir (as in .pkgtoolrc file or /tmp if not specified). After completion of pkgbuild execution, pkgtool moves the pkgbuild log contents into its own log file in the logdir (from /tmp to logdir/). If an error were to be encountered for any of the above steps, the log file should have the necessary details to help identify the problem and troubleshoot.

The **working steps of pkgbuild** are numbered [1] to [7] in the order of occurance in figure 4.2

1. Reads the configuration information from the ~/.pkgbuildmacros file for the pkgbuild environment settings. Command line parameters maybe specified to over ride the entries in the configuration file
2. Read the spec file
3. Based on the instructions in the spec file (%prep macro) the sources are copied from the sources directory to build directory and unzipped/untarred
4. The patches are applied to the sources (based on the %patch<n> tag in the spec file). Preparatory steps prior to the actual build are done (as per %setup macro)
5. The sources are built as per instructions in the %build macro
6. A source package is also created. This package contains the original sources, the patchset

and the spec files.

7. A temporary installation is done (%install macro). Using the temporary installation files and the inputs regarding the ownership, permission, destination-dirs etc from the spec file (%files list), a package containing the binaries is created

Throughout the above steps, a log file having the details of the steps performed by pkgbuild is maintained under the /tmp directory. If an error were to be encountered for any of the above steps, the log file should have the necessary details to help identify the problem and troubleshoot.

5.How to : Step-by-Step guide to your first package

The following are used for providing example details

```
The following is used for providing examples
      Username for creating the packages : pkguser
      User's home directory : /export/home/pkguser
      User's default shell  : bash
      Software that is built: geany V0.11 (http://sourceforge.net/projects/geany)
                              (Geany is an excellent light weight text editor/IDE)
```

5.1.Installing the required packaging tools

a)Create a non-root user for building packaging

Note : It is recommended not to use root user for building packages.

1. Create a user on the system that will be used for package building.

```
$ useradd -m -d /export/home/pkguser -s /bin/bash -c "Packing user" pkguser
```

2. Provide Software Installation privileges to the user by making it part of “Software Installation” profile. This is done by adding the below line to the end of the file /etc/user_attr
<user_name>::::profiles=Software Installation
Ex: For the above created user, the line to be added is
pkguser::::profiles=Software Installation

```
$ echo “pkguser::::profiles=Software Installation” >> /etc/user_attr
```

3. Set the password for the new user and switch to the new user.

Note: All the steps provided below are performed using this(**pkguser**) user. Substitute the username suitably for your case.

b)Install the JDS Common Build Environment(JDS CBE)

1. Download the JDS CBE. Latest JDS CBE may be downloaded from
http://opensolaris.org/os/project/jds/contributing/building/#jds-cbe
http://dlc.sun.com/osol/jds/downloads/cbe
JDS CBE includes

Apache ant	GNU autoconf	GNU automake
GNU bison	CVS	GNU diffutils
GNU fileutiles	GNU flex	GNU gettext

GNU libtool	GNU m4	GNU make
pkgbuild/pkgtool	rsync	subversion

2. Install the CBE as per the instructions in the README file of JDS CBE. Choose the default options for the installation. The tools get installed at /opt/jdsbld
   - Note: Incase your SXDE version is >=B66, then edit the installation script cbe-install to remove the SUNWsmbaS entry (at ~line 76) and then invoke the script.

Note:

- /opt/jdsbld/bin/env.sh is to be sourced as pkguser for the tools to be available with appropriate environment settings. To make the settings available automatically on login, source the env.sh file in the .profile file

5.2.Setup the configuration for the packaging tools

a)Create the configuration file for the packaging tools

Create a file .pkgtoolrc in the home directory of pkguser.
This file is used to set different options such as directories for SPECS, SOURCES, PATCHES, LOGS, final package format, etc.
A file template maybe created using pkgtool as

$ pkgtool –dumprc > [pkguser_homedir]/.pkgtoolrc file

Modify **[pkguser_homedir]/.pkgtoolrc** file suitably as per the comments in the file
Parameters set for the example are as below (only uncommented lines are shown)

```
date_format:    %y%m%d
tarballdirs:    /export/home/pkguser/packages/SOURCES
logdir:         /export/home/pkguser/packages/LOGS
halt_on_errors: 0
patchdirs:      /export/home/pkguser/packages/PATCHES
specdirs:       /export/home/pkguser/packages:/export/home/pkguser/packages/SPECS
nightly:        1
debug:          1
maintainers:    /export/home/pkguser/packages/SOURCES/maintainers.txt
download_to:    /export/home/pkguser/packages/REPOS
pkgformat:      datastream
```

The output of the **pkgtool --dumprc** command is nicely annotated with comments for each of the variables. Use these comments to suitable modify any additional variables or to customize the ones mentioned in the above table.

5.3.Author the spec file

Author the spec file **in stages** as detailed in the Section SPEC file structure.

At the end of each stage try out the spec file using the pkgtool using the below command:

```
bash-3.00$ pkgtool -download build-only SFEgeany.spec
```

This command tells pkgtool to download the sources if not already present in SOURCES dir and then proceed to create the packages.

The log file generated in the logdir(as in rc file) provides the details of success/errors. Go through the log file to get a clear understanding of the sequence of operations performed.

Note: The SPEC file and the patches that are created should go into locations as specified in the

[pkguser_homedir]/.pkgtoolrc file

At the end of this exercise, one should have a working spec using which packages can be created. More importantly, one should have the basic understanding of how pkgtool/pkgbuild work.

5.4.Test the output package

a)Outputs created by pkgtool/pkgbuild

Outputs of pkgbuild that are of interest to the user are:

- Source package – for the example it is created under the /export/home/pkguser/packages/SPKGS as SFEgeany-src.pkg
- Binary package - for the example it is created under the /export/home/pkguser/packages/PKGS as SFEgeany.pkg
- Logs file – A logs file is created under the logdir (default it /tmp) with details of operations performed by pkgtool/pkgbuild. This is quite useful incase there is a need for trouble shooting. For the example it is created as /export/home/pkguser/packages/LOGS/SFEgeany.log

Note: The source & binary packages created will be in the form of a single .pkg file or a directory containing the packaged files depending on the pkgformat parameter value specified in the rc file.

Additional outputs that are created for internal usage of pkgtool/pkgbuild

- Build directory (/export/home/pkguser/packages/BUILD) – location for untarring the sources, patching, compiling
- PKGMAPS (/export/home/pkguser/packages/PKGMAPS) – temporary files created based on the %files section of the spec file. The pkgmap files are required for the creation of SVR4 packages

b)Install the package & use the software

Install the package that was created using the pkgadd command. For the software that was built in the examples above,

```
Incase the package is in datastream format (.pkg file):
bash-3.00$ pkgadd -d SFEgeany.pkg

Incase the package is in filesystem format (directory):
bash-3.00$ pkgadd -d . SFEgeany
```

The pkgtool itself supports installation and un-installation options. See the “Useful pkgtool commands” section for details.

6.References

[1] http://pkgbuild.sourceforge.net

[2] http://www.opensolaris.org/os/project/jds

[3] http://www.opensolaris.org/os/project/jds/contributing/building

[4] http://www.rpm.org/RPM-HOWTO

7.APPENDICES

7.1.Useful pkgtool commands

Build the entire software stack using the available spec files. The pkgtool reads all spec files to determine the build order, then builds and installs them one by one.

```
bash-3.00$ pkgtool --download build -v *.spec
```

Print out the build order of the packages.

```
bash-3.00$ pkgtool -q build-order *.spec
```

Download all community source tarballs needed to build SFEgeany package.

```
bash-3.00$ pkgtool download SFEgeany.spec
```

Download all the community source tarballs needed to build the SUNWgnome-panel package (one needs to set the environmental variables http_proxy and ftp_proxy if necessary).

```
bash-3.00$ pkgtool download SUNWgnome-panel.spec
```

Download, build, and install all the packages in the correct order (one needs to set the environmental variables http_proxy and ftp_proxy if necessary).

```
bash-3.00$ pkgtool –download build-install *.spec
```

Install packages built using SFEgeany.spec (from PKGS directory)

```
bash-3.00$ pkgtool install-pkgs SFEgeany.spec
```

Uninstall packages SUNWgnome-utils and SUNWgnome-config-editor

```
bash-3.00$ pkgtool uninstall-pkgs SUNWgnome-utils.spec \
           SUNWgnome-config-editor.spec
```

Build the SUNWgnome-img-viewer package, but don't install it (this is the option that has been used in the step by step how-to)

```
bash-3.00$ pkgtool build-only SUNWgnome-img-viewer.spec
```

7.2.Useful pkgbuild commands

Execute only the prep section of the spec file

```
bash-3.00$ pkgbuild –-short-circuit -bp SFEgeany.spec
```

Execute only the prep & compile sections of the spec file (patched sources should exist)

```
bash-3.00$ pkgbuild --short-circuit -bc SFEgeany.spec
```

Execute only the install section of the spec file (compile should have completed successfully)

```
bash-3.00$ pkgbuild --short-circuit -bi SFEgeany.spec
```

Create packages using the installed files (files already installed in $RPM_BUILD_ROOT should exist). Note that on successful package creation, the command cleans up the temporarily installed files in $RPM_BUILD_ROOT.

```
bash-3.00$ pkgbuild --short-circuit -ba SFEgeany.spec
```

7.3.Spec file authoring – conventions & good practices

Below are some conventions and good practices to follow while creating spec files:

- Setting the commonly used variables (_basedir, _prefix, _bindir, etc) in a common Solaris.inc and include this in all spec files (refer Solaris.inc).

  Refer to include files in the include directory at https://pkgbuild.svn.sourceforge.net/svnroot/pkgbuild/spec-files-extra/trunk
- SVR4 package specific tag definitions (SUNW_BaseDir, SUNW_ProdName, SUNW_HotLine, etc) also needs to be done in the include file instead of the main spec file
- Patches follow the naming convention **%{name}-nn-description.diff** where nn indicates the number(order) of the patch (01, 02, 03, etc), description is a meaningful word, the suffix for a patch is always chosen to be **.diff**
- In the %files section,
  - %defattr should be (0755, root, bin)
  - The default permissions of the %{_datadir}'s is set to (0755, root, sys) except for %{_datadir}/applications which is set to (0755, root, other)
  - %{_datadir}/applications/* (files under applications) will continue to have %defattr permissions

7.4.Topics not covered in required details

In an attempt to keep the document simple and minimal, what has been provided are basic details.

Some of the topics not covered in this document (second level bullet points provide some pointers)

- Details about package dependency, creating multiple packages using single spec file as the entry point (Ex: foo.pkg, foo-devel.pkg & foo-man.pkg)
  - “Requires”, “BuildRequires” tags
  - “%package” tag and “%files <subpackagename>” for sub-package creation
- Command line usage of pkgtool, pkgbuild (this is necessary)
  - See the tool help output for details
  - See the section listing the useful pkgtool and pkgbuild commands
- Conventions/Guidelines to use macro definitions in include files

    - Have a look at the include files in SFE repository
- Taking care of cross platform issues during spec authoring
    - Have a look at the include files in SFE repository

---- TODO: These details should get addressed if and when the document is updated next ---

7.5.Creating & Applying patches

Unified diff output from GNU diff utility (/opt/jdsbld/bin/diff) is to be used for creating a patch

```
bash-3.00$ diff -uNr foo-dir.orig  foo-dir.patched > foo-01-sol12port.diff
```

This patch may be applied to the unmodified sources using the GNU's patch utility (/usr/bin/gpatch) in the location where foo-dir.orig

```
bash-3.00$ cd foo-dir.orig
bash-3.00$ gpatch -p1 < path-to-patchfile/foo-01-sol12port.diff
```

7.6.GNU Auto Tools

Some useful pointers for introductory level information

- http://developers.sun.com/solaris/articles/gnu.html - A introductory article on GNU auto tools
- http://developer.novell.com/wiki/index.php/Overview_of_GNU_Autotools – Overview of GNU auto tools (have a look at the “Building Your Application” section)

For those who want to get into the complete details

- http://sourceware.org/autobook/autobook/autobook_toc.html – The Autobook
- http://www.gnu.org/manual/manual.html – The GNU Manuals

---------- TODO: Some basic details to be made part of this document ----------

7.7.Build Issues on OpenSolaris platform

---------- TODO: Some background information about reasons for failure and broad hints as to where to look for help to be added ----------

7.8.SVR4 packaging details

---------- TODO: SVR4 packaging related details go here ----------

7.9.Specs for pkgbuild v/s RPM Specs

---------- The differences and reasons for the same goes here ----------

~ End of Document ~