```
# Example 1: Hello World! (thanks to Mauricio Vergara Ereche)

Name:       helloworld
Version:    1.1
Release:    3
Summary:    An application that prints “Hello World!”
License:    GPLv2+
Group:      System Environment/Base
Source0:    http://helloworld.com/helloworld-1.1.tar.gz
Patch0:     fixtypo.patch
BuildRoot:  %{_tmppath}/%{name}-%{version}-%{release}-root-%(%{__id_u} -n)
BuildArch:  noarch

%description

This program prints hello world on the screen to avoid the “programmers curse”.
The Programmmers Curse states that unless your first example is “Hello World!”,
then you will be cursed, and never able to use that tool.

%prep

%setup -q
%patch0 -p1

%build
%configure
make %{?_smp_mflags}

%install
rm -rf %{buildroot}
mkdir -p %{buildroot}%{_bindir}
cp helloworld.sh %{buildroot}%{_bindir}/helloworld

%clean
rm -rf %{buildroot}

%files
%defattr(-,root,root,-)
%attr(0755,gold,fish) %{_bindir}/helloworld


%changelog
* Mon Jun 2 2008 Tom “spot” Callaway <tcallawa@redhat.com> 1.1-3
- minor example changes

* Mon Apr 16 2007 Tom “spot” Callaway <tcallawa@redhat.com> 1.1-2
- update example package

* Sun May 14 2006 Tom “spot” Callaway <tcallawa@redhat.com> 1.1-1
- initial package for Red Hat Summit Presentation
```

```
# Example 2: With Comments!

# This is a heavily commented version of our helloworld spec file in Example 1,
# but it is otherwise identical.
# Comments start with a # sign, but be careful, as macros in comments ARE parsed.
# To avoid having macros be parsed, double up on the %%.

# This is the package name. All spec files need a name field.
Name:       helloworld

# This is the package version. All spec files need a version field.
Version:    1.1

# This is the package release, a build counter. Increment this value when you make
# a change to the spec file. All spec files need a release field.
Release:    3

# This is the package summary. It is a short string (less than 80 characters,
# please) that describes the package functionality. Try not to use the package
# name here. This field should tell me at a glance what this package does.
Summary:    An application that prints “Hello World!”

# This is the license of the package. All spec files need a license field.
# Fedora packages use this list: http://fedoraproject.org/wiki/Licensing
License:    GPLv2+

# This is the group of the package. All spec files need a group field.
# You can find a list of Red Hat groups here:
# http://fedoraproject.org/wiki/RPMGroups
Group:      System Environment/Base

# This is the source code that we want to use to build our package. You can have
# multiple source files. All spec files need at least one source file to be
# useful. Whenever possible, this should be a URL (you still need to download it
# locally).
Source0:    http://helloworld.com/helloworld-1.1.tar.gz

# This is a patch file. It fixes a minor typo in our imaginary helloworld
# script.
Patch0:     fixtypo.patch

# This is the build root. You NEED to set a buildroot in the spec file for safety.
BuildRoot:  %{_tmppath}/%{name}-%{version}-%{release}-root-%(%{__id_u} -n)

# This is the build architecture. You usually only need to set this if the package
# is explicitly not for any specific architecture, like a text file or a script
# package. You won't need this for normal binary packages.
BuildArch:  noarch

# This is the description section, where you put a thorough, but concise
# description of what this package does. This should be more in depth than the
# Summary section.
%description

This program prints hello world on the screen to avoid the “programmers curse”.
The Programmmers Curse states that unless your first example is “Hello World!”,
then you will be cursed, and never able to use that tool.
```

```
# The %%prep macro signals that it is time to get down to business. All spec files
# need to start with %%prep.
%prep

# The %%setup macro prepares the tree. By default, it unpacks Source0. The -q
# option causes the operations to be performed quietly, so we don't have to watch
# it untar the source file(s). All spec files need a %%setup section.
%setup -q

# Here, we apply our Patch. To apply a patch file, just put a percent sign in
# front of it, and pass the patch level. In this case, we're using -p1. For more
# information on how the “patch” command works in Linux, run: man patch
%patch0 -p1

# The %%build macro is where we perform the “build” tasks.
# This is a dummy example, so we'll pretend that we're doing normal tasks.
# Most UNIX/Linux applications use “configure” and “make”, so we'll do that.
# Red Hat provides a special macro to handle configure, called %%configure. When
# you use %%configure, it passes the common options to configure to ensure that
# files get installed in the right places and that compiler optimizations are
# used. The %%{?_smp_mflags} macro allows you to pass SMP flags to make, if
# %%{_smp_mflags} is set (thats what the ? means). If not set, it evaluates to
# null.
%build
%configure
make %{?_smp_mflags}

# The %%install macro is where we perform the installation of the built files. All
# spec files will need an %%install section.
# You will want to delete the buildroot before installing files into it. This
# could be risky, but you set a safe buildroot at the top of the spec file, so
# you're fine.
# We're also making a target directory in the buildroot for our helloworld binary,
# then copying it in. Any files or directories in the buildroot will reflect what
# they will look like when the package is installed - except without the
# buildroot.
%install
# This line uses the variable for BuildRoot. You can either use $RPM_BUILD_ROOT or
# %%{buildroot}. They're the same thing. But be consistent, only use one or the
# other throughout the spec.
# %%{_bindir} is an rpm provided macro for /usr/bin. While its not likely that
# Red Hat will stop putting binaries in /usr/bin, by using the macro, we're
# prepared.
rm -rf %{buildroot}
mkdir -p %{buildroot}%{_bindir}
cp helloworld.sh %{buildroot}%{_bindir}/helloworld

# The clean macro runs after we're done making the packages. We don't need our
# buildroot anymore, so we get rid of it here. You should do this in all your
# packages.
%clean
rm -rf %{buildroot}
```

```
# The files macro lists the files in our BuildRoot that we want to include in the
# binary package. If it is not listed here, it won't go in the package. But be
# aware that rpm will complain if files are left unpackaged in the BuildRoot!
# The defattr macro sets the default attributes for the files.
# The attr macro sets specific attributes for a specific file (or filemask).
# Here we're setting the default attributes for the files section to root user and
# group ownership, but we're setting /usr/bin/helloworld to be owned by user gold,
# and group fish. Remember that %%{_bindir} is just /usr/bin.
%files
%defattr(-,root,root,-)
%attr(0755,gold,fish) %{_bindir}/helloworld

# The changelog macro is where we keep a log of all the changes that we've made to
# the package. Keeping this current is vital to making a good, maintainable
# rpm package. The changelog syntax is:
# * DayName Month Day Year NameOfPackager <email@dot.com> Version-Release
# - short description of what changes we made for the package
# DO NOT put copies of the source code changelog here, keep in mind that this is
# going into the RPM database upon installation, so we don't want to inflate the
# database. If that Changelog is important, include it in the package itself.
%changelog
* Mon Jun  2 2008 Tom “spot” Callaway <tcallawa@redhat.com> 1.1-3
- minor example changes

* Mon Apr 16 2007 Tom “spot” Callaway <tcallawa@redhat.com> 1.1-2
- update example package

* Sun May 14 2006 Tom “spot” Callaway <tcallawa@redhat.com> 1.1-1
- initial package for Red Hat Summit Presentation
```

```
# Example 3: One of my Fedora Packages

Name:           kscope
Summary:        KDE front-end to Cscope
Version:        1.6.1
Release:        4%{?dist}
License:        BSD
Group:          Development/Tools
Source0:        http://download.sourceforge.net/kscope/%{name}-%{version}.tar.gz
URL:            http://kscope.sourceforge.net/
BuildRoot:      %{_tmppath}/%{name}-%{version}-%{release}-root-%(%{__id_u} -n)
BuildRequires:  desktop-file-utils, bison, flex, kdelibs3-devel
BuildRequires:  libjpeg-devel, libpng-devel, gettext, kdebase3-devel
Requires:       cscope, ctags, graphviz

%description
KScope is a KDE front-end to Cscope. It provides a source-editing
environment for large C projects, such as the Linux kernel.

KScope is by no means intended to be a replacement to any of the leading
Linux/KDE IDEs, such as KDevelop. First of all, it is not an Integrated
Development Environment: it does not provide the usual write/compile/debug
cycle supported by most IDE's. Instead, KScope is focused on source
editing and analysis.

%prep
%setup -q

%build
%configure –disable-rpath
make %{?_smp_mflags}

%install
rm -rf $RPM_BUILD_ROOT
make DESTDIR=$RPM_BUILD_ROOT install

# Clean up dangling symlink
rm -rf $RPM_BUILD_ROOT%{_defaultdocdir}/HTML/en/kscope/common
pushd $RPM_BUILD_ROOT%{_defaultdocdir}/HTML/en/kscope/
ln -s ../common common
popd

desktop-file-install --vendor="" --delete-original \
  --dir $RPM_BUILD_ROOT%{_datadir}/applications/kde \
  $RPM_BUILD_ROOT%{_datadir}/applnk/Development/kscope.desktop

%find_lang %{name}

%clean
rm -rf $RPM_BUILD_ROOT
```

```
%post
touch --no-create %{_datadir}/icons/hicolor
if [ -x %{_bindir}/gtk-update-icon-cache ]; then
  %{_bindir}/gtk-update-icon-cache --quiet %{_datadir}/icons/hicolor || :
fi

%postun
touch --no-create %{_datadir}/icons/hicolor
if [ -x %{_bindir}/gtk-update-icon-cache ]; then
  %{_bindir}/gtk-update-icon-cache --quiet %{_datadir}/icons/hicolor || :
fi

%files -f %{name}.lang
%defattr(-,root,root,-)
%doc AUTHORS ChangeLog COPYING
%{_bindir}/kscope
%{_datadir}/applications/kde/*kscope.desktop
%{_datadir}/apps/%{name}/
%{_defaultdocdir}/HTML/en/%{name}/
%{_datadir}/icons/hicolor/*/apps/%{name}.png
%{_datadir}/icons/locolor/*/apps/%{name}.png

%changelog
* Tue Feb 19 2008 Fedora Release Engineering <rel-eng@fedoraproject.org> - 1.6.1-4
- Autorebuild for GCC 4.3

* Sun Jan 20 2008 Tom "spot" Callaway <tcallawa@redhat.com> 1.6.1-3
- drop vendor

* Sun Jan 20 2008 Tom "spot" Callaway <tcallawa@redhat.com> 1.6.1-2
- cleanups from review

* Fri Jan 18 2008 Tom "spot" Callaway <tcallawa@redhat.com> 1.6.1-1
- Initial package for Fedora
```

```
# Example 4: kscope, digested.
# You can go and get this package from Fedora right now.
# Remember that macros in comments use %% instead of just one percentage symbol to
# avoid being interpreted by rpm.

# This package is named kscope.
Name:           kscope
# This summary is short, but concise.
Summary:        KDE front-end to Cscope
# It is currently at version 1.6.1
Version:        1.6.1
# This is the fourth package release for version 1.6.1. Lets talk about the
# %%{?dist} macro. Fedora uses this macro to identify the distribution that this
# package is being built for. When this package is built in the Fedora buildsystem
# for Fedora 9, the %%{dist} macro is set to .fc9. This makes the Release: 1.fc9.
# But if you build this package on your local machine, the %%{dist} macro may not
# be set. This is why we use the question mark. If the macro is unset, it is set
# to null, and ignored. So, when you build it, the Release is just 1. Nifty, huh?
# If you'd like to read all about the Fedora dist tag macro magic, go to:
# http://fedoraproject.org/wiki/DistTag
Release:        4%{?dist}
# Kscope is released under the BSD license.
License:        BSD
# It is a development tool, so, I put it in the Development/Tools group.
Group:          Development/Tools
Source0:        http://download.sourceforge.net/kscope/%{name}-%{version}.tar.gz
# This field identifies the URL for the upstream software website.
URL:            http://kscope.sourceforge.net/
# This is where we define the BuildRoot. All of the examples so far have used the
# most common Fedora buildroot. Let's take it apart and explain the pieces:
# %%{_tmppath} is an external macro. External macros can usually be identified by
# the fact that they begin with an underscore. You can get a list of all the
# external macros that rpm knows about by running: rpm --showrc
# On Red Hat/Fedora systems, %%{_tmppath} is usually set to /var/tmp.
# The %%{name}, %%{version}, and %%{release} macros should be obvious to you now.
# %%(%%{__id_u} -n) is an executed callout macro. Anything inside the parenthesis
# is evaluated, then executed. The results take the place of the macro. This is
# very powerful, but you should be VERY CAREFUL with this. For example, never call
# out to rpm inside one of these macros, as you could confuse rpm badly.
# By looking in the --showrc output, we can determine that %%{__id_u} is usually
# set to /usr/bin/id -u. So, if I was logged in as “spot”, then the executed
# callout macro would evaluate to: spot
# When we put all these pieces together, this is what the BuildRoot evaluates to:
# /var/tmp/kscope-1.6.1-4.fc9-root-spot
# Fedora uses this BuildRoot to ensure that multiple builds can have their
# own BuildRoot, as well as multiple users.
BuildRoot:      %{_tmppath}/%{name}-%{version}-%{release}-root-%(%{__id_u} -n)
# This is where we list BuildRequires. BuildRequires are packages that need to be
# installed on the build system in order for the package to build successfully (or
# with specific functionality). You can have multiple packages on a BuildRequires
# line, and you can have multiple BuildRequires lines. It is important that you be
# thorough when filling in these values, otherwise, your program may not build at
# all, or without specific functionality that you intended it to have. Watching
# the output from the “configure” script execution will help to identify libraries
# and applications that the program will need to build.
# You also need to keep in mind that any programs that you need to run in the
# any of the below sections will also need to be listed as BuildRequires. The
# exception to this is any packages that are included in RHEL or Fedora's Base
# group. There is no need to list coreutils or bash here, as no Red Hat/Fedora
# system will be missing these packages. A list of items that you can ignore as
# BuildRequires can be found here:
# http://fedoraproject.org/wiki/Packaging/Guidelines#Exceptions
# One more thing: If kdelibs3-devel requires glibc-devel to be installed, you
# don't need to list glibc-devel as a BuildRequires. Technically, you do need them
# both, but since kdelibs3-devel can't be present without glibc-devel, you cover
# them both with kdelibs3-devel.
BuildRequires:  desktop-file-utils, bison, flex, kdelibs3-devel
BuildRequires:  libjpeg-devel, libpng-devel, gettext, kdebase3-devel
# This is where we list manual Requires for this package. RPM automatically adds
# Requires for items that the built binaries are linked against, and shell
# interpreters, but it doesn't pickup on applications which are executed from
# inside kscope, which is why cscope, ctags, and graphviz are explicitly listed
# here as Requires.
Requires:       cscope, ctags, graphviz

# A slightly longer program description. Remember, you don't want to put the
# README here, just enough for people to know what the program is about.
%description
KScope is a KDE front-end to Cscope. It provides a source-editing
environment for large C projects, such as the Linux kernel.

KScope is by no means intended to be a replacement to any of the leading
Linux/KDE IDEs, such as KDevelop. First of all, it is not an Integrated
Development Environment: it does not provide the usual write/compile/debug
cycle supported by most IDE's. Instead, KScope is focused on source
editing and analysis.

# Now, we start the process!
%prep
# The setup macro unpacks the first source file by default. It is capable of
# handling multiple source files, but we're just going to call them directly later
# in the spec. The -q flag keeps it quiet while it unpacks, you may want to remove
# this flag if you hit problems here.
%setup -q

# Now, we have an unpacked source tree. It's time to build!
%build
# The %%configure macro is a system macro which executes ./configure along with
# all the options to enable the proper locations for files, configurations, and
# optimization flags. You can add any additional flags to the end, like I've done
# to disable rpath.
%configure --disable-rpath
# The make command is a UNIX/Linux standard. We're using it to build this
# application. If we needed to set some environment variables for the build, we
# could do that here, but kscope does not need this.
# There is also a macro in use here:
# %%{_smp_mflags} is a macro that Fedora creates and uses to set the “-jN”
# option on SMP systems, where N is the number of CPUs. This enables make to split
# the build process across multiple CPUs to speed up the build. We also use it on
# non-SMP systems to test and see if the sourcecode can be built simultaneously
# across multiple CPUs (some code cannot). Since it has a ? in front of it, if it
# is unset, it is ignored.
make %{?_smp_mflags}
```

```
# Now, we install the files into the temporary buildroot.
%install
# First, we delete the buildroot, so that we know we are getting the freshly built
# files and not older files from a previous build.
rm -rf $RPM_BUILD_ROOT
# Here, we run make install, but we also set the DESTDIR environment variable to
# the RPM buildroot. Otherwise, these files would end up in /.
make DESTDIR=$RPM_BUILD_ROOT install

# Clean up dangling symlink
# The pushd/popd utilities allow us to cleanly make a symlink.
rm -rf $RPM_BUILD_ROOT%{_defaultdocdir}/HTML/en/kscope/common
pushd $RPM_BUILD_ROOT%{_defaultdocdir}/HTML/en/kscope/
ln -s ../common common
popd

# Since this package is a GUI application, it comes with a .desktop file. This
# enables it to appear in the main menu. Fedora uses the desktop-file-install
# command to install and verify this file.
desktop-file-install --vendor="" --delete-original \
  --dir $RPM_BUILD_ROOT%{_datadir}/applications/kde \
  $RPM_BUILD_ROOT%{_datadir}/applnk/Development/kscope.desktop

# The strings in this application have been translated! We want to package them,
# but if we were to do it by hand, it would be rather tedious (each translation
# lang file would need to be marked in the %%files list with its locale). Fedora
# includes a macro to make this much simpler, %%find_lang. Most translated
# applications for Linux have a standard naming scheme for their translated lang
# files, and it is almost always the name of the application. We just need to pass
# that common name to %%find_lang, and it will generate a list of these files for
# us to use in our %%files section.
%find_lang %{name}

# Don't forget to clean up! No one likes a messy packager.
%clean
rm -rf $RPM_BUILD_ROOT

# This is a post scriptlet, it will run after this package is installed.
# For kscope, it refreshes the icon cache, so that the menu icons for this
# application appear immediately.
%post
touch --no-create %{_datadir}/icons/hicolor
if [ -x %{_bindir}/gtk-update-icon-cache ]; then
  %{_bindir}/gtk-update-icon-cache --quiet %{_datadir}/icons/hicolor || :
fi

# This is a postun scriptlet, it will run after this package is uninstalled.
# Just like the post scriptlet, it will refresh the icon cache so that the
# kscope icons do not show up when this package is removed.
%postun
touch --no-create %{_datadir}/icons/hicolor
if [ -x %{_bindir}/gtk-update-icon-cache ]; then
  %{_bindir}/gtk-update-icon-cache --quiet %{_datadir}/icons/hicolor || :
fi
```

```
# This is where you list the files in the package. They will all get the default
# ownership by root.root. This is good, because we don't want regular users to be
# able to make changes to our game components at will. Note that this is different
# from SUID root, which is much more dangerous. Also, we're appending a
# -f %%{name}.lang. %%{name}.lang is the file list that %%find_lang made for us
# during %%install. By passing -f, %%files will also read in the items in that
# file, and include them in this package.
%files -f %{name}.lang
%defattr(-,root,root,-)
# The %%doc macro is special for %%files. Unlike all the other entries in
# %%files, which have to be present in the buildroot, the %%doc files are pulled
# from the source tree, and put into /usr/share/doc/%%{name}-%%{version}. RPM also
# flags these files as documentation, so it is useful to use the %%doc macro here.
%doc AUTHORS ChangeLog COPYING
%{_bindir}/kscope
%{_datadir}/applications/kde/*kscope.desktop
# When you create a new directory for your package, like
# %%{_datadir}/apps/%%{name}/, you need to make sure that the RPM owns the
# directory as well. This ensures that when your package is removed, all the
# directories it made are cleaned out with it. You do not need to own any of the
# system directories created by the “filesystem” package, in fact, doing so tends
# to clutter up the RPM database. When you want to own a directory and all the
# files below it, just list the directory by name, as we have done here.
%{_datadir}/apps/%{name}/
%{_defaultdocdir}/HTML/en/%{name}/
%{_datadir}/icons/hicolor/*/apps/%{name}.png
%{_datadir}/icons/locolor/*/apps/%{name}.png

# Last but not least, the changelog. This package has had four releases.
# Note that the date, the individual doing the rebuild, and the full
# version-release are on the first line, while the comments follow on additional
# lines.
%changelog
* Tue Feb 19 2008 Fedora Release Engineering <rel-eng@fedoraproject.org> - 1.6.1-4
- Autorebuild for GCC 4.3

* Sun Jan 20 2008 Tom "spot" Callaway <tcallawa@redhat.com> 1.6.1-3
- drop vendor

* Sun Jan 20 2008 Tom "spot" Callaway <tcallawa@redhat.com> 1.6.1-2
- cleanups from review

* Fri Jan 18 2008 Tom "spot" Callaway <tcallawa@redhat.com> 1.6.1-1
- Initial package for Fedora
```

```
# Example 5: A package with no source code, just binaries
# Lets suppose that you have a tarball that contains the binary files for an ISV
# application that your boss purchased. We'll call it the “Prophet” database.
# This one is going to be a lot more complicated, but we'll explain it.
# Some lines are wrapped, look for the \ to indicate line wrap.

# It is good practice to define custom macros at the top of the spec file. As long
# as it isn't already an external macro, you can make it a macro with the %%define
# option. Prophet uses several different variants on versioning, it has a long
# string, a major string, and a short version. We'll use macros to define these
# variables early on.

# This defines the %%{prophet_version} macro to 9.2.0.6.0. Normally, we wouldn't
# need to make a special macro for the version, but we want to use it very early
# on.
%define prophet_version 9.2.0.6.0

# This defines the %%{dotted_sv} macro by executing a callout macro. In this case,
# the result sets %%{dotted_sv} to 9.2.0.
%define dotted_sv %(echo %{prophet_version} | awk -F. '{printf("%s.%s.%s", $1, $2,
\$3)}')

# This defines the %%{short_version} macro by taking the %%{dotted_sv} value and
# executing a callout macro to remove the dots. This sets the macro to 920.
%define short_version %(echo %{dotted_sv} | tr --delete .)

# Like many third party applications, Prophet likes to live in /opt. Here, we'll
# make macros to define its top level directory, home directory, config directory,
# and data directory.
%define prophet_base /opt/apps/prophet
%define prophet_home %{prophet_base}/web/product/%{dotted_sv}
%define prophet_config %{prophet_base}/config/%{dotted_sv}
%define prophet_data %{prophet_base}/data/%{dotted_sv}

# Note that we are already using some of these macros that we defined. Normally,
# setting a custom macro to hold the version is pointless, but we want to use the
# version in the Name field, and in a variety of formats later in the spec.
Name:       prophet-server-%{short_version}
Summary:    Prophet 9 Database Server Enterprise Edition
Version:    %{prophet_version}
Release:    2
Source0:    prophet-server-%{version}.tar.bz2
Patch0:     prophet-shell.patch
License:    Prophet License (Proprietary)
Group:      Applications/Databases
Buildroot:  %{_tmppath}/%{name}-%{version}-%{release}-root-%(%{__id_u} -n)

# We used awk above, so we better have it as BuildRequires. tr is in coreutils, so
# we can safely use it. Much later in the script, we'll use useradd/groupadd and
# /sbin/service, but those packages (shadow-utils, initscripts) are in the Base
# group, and it is safe to assume that they will be on the system.
# Grep and findutils are also used in the spec file, so we'll list them here to be
# safe.
BuildRequires: gawk, grep, findutils
```

```
# RPM has a field for setting Provides. These are items or variables that this
# package provides. These can be files or names that we want the RPM database to
# know that we own, above and beyond what is in the %%files list. For the Prophet
# package, we want it to know that we also provide prophet-server for this
# package, with the full verson.
Provides: prophet-server = %{version}

# RPM also has a field for hardcoding Requires. It does a good job of
# automatically detecting Requires for libraries and binaries, but has no way to
# know that this application will not run without its script files. Thus, we
# hardcode a Requires for the prophet-server-scripts package that matches our
# version.
Requires: prophet-server-scripts-%{short_version} = %{version}

# A description for the main package, short and sweet.
%description
This package contains the Prophet 9 Database Server Enterprise Edition,
preinstalled and packaged for easy deployment.

# What is this? Another package in the same spec file? Sure! We can call %%package
# multiple times to create subpackages made from the same set of sources. Here,
# we're defining the settings for the prophet-relink subpackage. Note that we
# called it relink, but when it is built, it will be called prophet-relink.
# A subpackage requires a Summary, a License, a Group, and any extra Requires that
# are specific to the subpackage. Keep in mind that most subpackages will need to
# Require the main package.
%package relink
Summary: Prophet 9 Database Server Object files for relinking
License: Prophet License (Proprietary)
Group: Applications/Databases
Requires: prophet-server-%{short_version} = %{version}-%{release}

# Subpackages also need their own description.
%description relink
This package contains the static object files (.a libraries, o files and
.makefiles) necessary for rebuilding/relinking the prophet server when patches
and/or removing/adding components.

# Now we'll define the headers for prophet-pui, prophet-doc, prophet-extra,
# prophet-java, and prophet-em subpackages.

%package pui
Summary: Prophet 9 Database Server installer inventory files
License: Prophet License (Proprietary)
Group: Applications/Databases
Requires: prophet-server-%{short_version} = %{version}-%{release}
Requires: prophet-server-%{short_version}-relink = %{version}-%{release}

%description pui
This package contains the installer inventory files that resulted after the
installation and setup of the prophet-server package using the Prophet Universal
Installer.

%package doc
Summary: Prophet 9 server documentation files
License: Prophet License (Proprietary)
Group: Applications/Databases
Requires: prophet-server-%{short_version} = %{version}-%{release}
```

```
%description doc
This package includes documentation files from the Prophet 9
distribution archives. In this package you will find miscellaneous
documentation bits, samples, demos, quick tours and various
help files which are not normally required for the Prophet Server to
function properly (ie, answer to SQL queries)

%package extra
Summary: Prophet 9 server Optional Components
License: Prophet License (Proprietary)
Group: Applications/Databases
Requires: prophet-server-%{short_version} = %{version}-%{release}
Requires: prophet-server-%{short_version}-java = %{version}-%{release}

%description extra
This package includes the optional components of the Prophet 9
Enterprise Edition. Currently these include the UltraSearch, various OEM
webstage plugins and Prophet Assistants (database Creation Assistant,
Database Management Assistant)

%package java
Summary: Prophet 9 server Java and Apache JSP support
License: Prophet License (Proprietary)
Group: Applications/Databases
Requires: oracle-server-%{short_version} = %{version}-%{release}

%description java
This package includes the JAVA-related components from the Prophet 9
Enterprise Edition. Currently these include the Prophet HTTP Server (based
on Apache), various Jave development kits, XML-Java development kit,
SQLJ and various .jsp pages to be used with the Prophet HTTP server.

%package em
Summary: Prophet 9 Server Enterprise Manager Files
License: Prophet License (Proprietary)
Group: Applications/Databases
Requires: prophet-server-%{short_version} = %{version}-%{release}
Requires: prophet-server-%{short_version}-java = %{version}-%{release}

%description em
This package includes the Prophet 9 Enterprise Manager console and
administration files (also known as the MANSYS console).
In the current implementation it requires most of the other java stuff
(available in the prophet-server-java package) to be installed as well.

# Ok, lets get down to business.
%prep

# We don't need setup to open the tarball, since it doesn't contain a nice
# directory tree, just a “bunch of files”. We use -c to create a dummy toplevel
# source directory, and we use -T to tell %%setup not to try to unpack Source0.
# You can learn all about %%setup's various options here:
# http://www.rpm.org/max-rpm-snapshot/s1-rpm-inside-macros.html
%setup -c -T

# There is no %%build section, we don't need it.
```

```
%install
rm -rf $RPM_BUILD_ROOT
mkdir -p $RPM_BUILD_ROOT

# Since we know we've got all the files built in the tarball, we'll just untar
# them directly into the buildroot, then apply our patch. We could have let %setup
# do it, and copy it over, but why waste the time/space?
tar xjf %{SOURCE0} -C $RPM_BUILD_ROOT
patch -p0 -d $RPM_BUILD_ROOT < %{PATCH0}

# This line looks for .so library files and makes them executable. This is
# important, because the debuginfo mechanisms need them to be executable.
# Red Hat implemented a set of macros that takes packages during the final stages,
# identifies the binaries and libraries (by the fact that they are set to be
# executable), then separates the debugging symbols out into their own subpackage,
# called %%{name}-debuginfo. This package, when installed, puts the symbols on the
# system where gdb and similar utilities can find them and use them to improve
# debugging output. This is all done automatically at build time, so you
# just need to make sure that your binaries and libraries are set executable.
find $RPM_BUILD_ROOT -type f -name \*.so -exec chmod +x {} \;

install -d -m 755 $RPM_BUILD_ROOT%{prophet_data}

# Replace dbs with a symbolic link to config
# Keep in mind that $() is the same as %%()
install -d -m 755 $(dirname  $RPM_BUILD_ROOT%{prophet_config})
cp -ar $RPM_BUILD_ROOT%{prophet_home}/dbs $RPM_BUILD_ROOT%{prophet_config}
rm -rf $RPM_BUILD_ROOT%{prophet_home}/dbs
ln -s ../../../config/$(basename %{prophet_config}) \$RPM_BUILD_ROOT%
{prophet_home}/dbs

# Here, we get rid of some unneeded files.
# Keep in mind that you need to delete any files in the buildroot that you do not
# intend to package, or rpm will complain about unpackaged files.
rm -rfv $RPM_BUILD_ROOT%{prophet_home}/lib/stubs
rm -fv $RPM_BUILD_ROOT%{prophet_home}/root.sh.old
rm -fv $RPM_BUILD_ROOT/etc/protab
rm -f $RPM_BUILD_ROOT%{prophet_home}/lib/*O
rm -f $RPM_BUILD_ROOT%{prophet_home}/bin/*O
rm -f $RPM_BUILD_ROOT%{prophet_home}/ctx/bin/*O
rm -f $RPM_BUILD_ROOT%{prophet_home}/bin/*.orig
rm -f $RPM_BUILD_ROOT%{prophet_home}/bin/*.bak*
rm -f $RPM_BUILD_ROOT%{prophet_home}/lib/*so0
rm -f $RPM_BUILD_ROOT%{prophet_home}/bin/*[a-zA-Z]0
# rtspro has weird dependencies; remove it
rm -f $RPM_BUILD_ROOT%{prophet_home}/bin/rtspro

# build the file lists
# Instead of manually listing out all the files in the %%files section, you can
# generate file lists and use them. We'll use find and grep to make these lists.
{
find $RPM_BUILD_ROOT%{prophet_base} -type d -mindepth 1 | \
        sed -e "s|^$RPM_BUILD_ROOT|%dir |"
find $RPM_BUILD_ROOT%{prophet_base} -type f -o -type l | \
        sed -e "s|^$RPM_BUILD_ROOT||"
} | egrep -v "/[0-9]*_patch(/|$)" > files.list

# pui
```

```
gs="/(diagnostics|inventory|proInst.loc|proInventory|install|install.platform|
pui|.patch_storage)(/|$)"
egrep $gs files.list > install.list || true
egrep -v $gs files.list > files.new && mv -f files.new files.list


# extract the enterprise manager stuff
gs="/(mansys|em|EM)(/|$)"
egrep $gs files.list > files-em.list || true
egrep -v $gs files.list > files.new && mv -f files.new files.list

# extract extra stuff
gs="/(assistants|cwmlite|dm|ds|hs|ldap|md|mgw|oem_webstage|prd|plap|slax|
syndication|ultrasearch|wwg|(lib|lib32)/(hsdb_.*|libhs.*)|bin/(pcm|pdisrvreg|
pdisrv|ptt|schemasync)|ldap/install/pdisrv.prt)(/|$)"
egrep $gs files.list > files-extra.list || true
egrep -v $gs files.list > files.new && mv -f files.new files.list

# extract the java stuff
gs="/(Apache|BC4J|JRE|classes|jar|javavm|jdbc|jdk|jlib|jre|jsp|pcs4j|pwm|soap|
sqlj|weboamlib|xdk|network/agent)(/|$)"
egrep $gs files.list > files-java.list || true
egrep -v $gs files.list > files.new && mv -f files.new files.list

# extract documentation
gs="/(doc|demo|help|qtour|relnotes|sample)(/|$)"
egrep $gs files.list > files-doc.list || true
egrep -v $gs files.list > files.new && mv -f files.new files.list

gs="\.(p|a|mk|def)$"
egrep $gs files.list > files-relink.list || true
egrep -v $gs files.list > files.new && mv -f files.new files.list

# Phew. That was a lot of fun regexing and searching, but now we have file lists
# for each of our packages. These lists need to be stored in the source dir, which
# is why we created it with -c at %%setup.

# Keep clean!
%clean
rm -rf $RPM_BUILD_ROOT

# This is a new section, %%pre gets executed before the package is installed.
# This is useful in this package because it lets us create the custom users that
# we need for this application to run.
%pre
# Add the user prophet in the new group dba.
if [ -z "$(grep '^dba:' /etc/group)" ] ; then
    /usr/sbin/groupadd -r dba >/dev/null 2>&1 || true
fi
if [ -z "$(grep '^prophet:' /etc/passwd)" ] ; then
    /usr/sbin/useradd -G dba -c "Prophet Server" \
        -r -d %{prophet_base} prophet >/dev/null 2>&1 || true
fi
if [ -f /etc/rc.d/init.d/nscd ] ; then
    /sbin/service nscd status >/dev/null 2>&1 && \
        /sbin/service nscd restart >/dev/null 2>&1
fi
```

```
# Make sure that the prophet account has a home directory
[ -d ~prophet ] || mkdir -p ~prophet

# Here, we'll make a .bash_profile for our prophet user with the environment
# variables that Prophet needs to function. Keep in mind that rpm macros will be
# evaluated before being put into the file, so you'll see “created by the prophet-
# server-920 package”.
if [ ! -f ~prophet/.bash_profile ] ; then
    cat >~prophet/.bash_profile <<EOF
#!/bin/bash
#
# bash profile automatically created by the %{name} package
EOF
fi

if [ -z "$(grep PROPHET_HOME ~prophet/.bash_profile 2>/dev/null)" ] ; then
    cat <<EOF >>~prophet/.bash_profile
# start entries added by the %{name} install script
PROPHET_HOME=%{prophet_home}
export PROPHET_HOME

PATH=\$PROPHET_HOME/bin:\$PATH
export PATH
# end entries added by the %{name} install script
EOF
fi
exit 0

# The %%preun section runs before we uninstall. It is good for doing early
# cleanups, for bits that might be generated by the package, but not packaged.
# There are also %%post and %%postun sections that you can use, but this package
# doesn't need them.
%preun
# clean up various logs left behind
if [ $1 = 0 ] ; then
    rm -f %{prophet_home}/network/log/* 2>/dev/null
    rm -f %{prophet_home}/rdbms/audit/* 2>/dev/null
fi
exit 0

# Now, we just need to use the files.lists here. All we have to do is pass the
# file list name after -f.
# We still want to set the default attributes for these files, so that they are
# owned by our user and group. Lastly, we want to make sure we own the directories
# that we create. Since the files are taken care of by the lists, we use the %%dir
# macro. This allows us to only own the dir, and not the files inside of it. If we
# didn't use %%dir, we'd have duplicate file ownership, which, while not fatal, is
# very messy.
%files -f files.list
%defattr(-,prophet,dba)
%dir %{prophet_base}
%dir %{prophet_base}/jre

# To make file sections for subpackages, just put the subpackage name after
# %%files.
%files relink -f files-relink.list
%defattr(-,prophet,dba)
```

```
%files pui -f install.list
%defattr(-,prophet,dba)
# We excluded this file from the file list on purpose, so we could call it out
# here. This file is a configuration file, rpm will let us mark those files with
# %%config. It handles those files gracefully on upgrades and removals, so that
# config files are preserved. Here, we're calling %%config(noreplace), because we
# don't want this config file to be replaced by later packages.
# It also has different ownership and perms from the default attributes.
%attr(644,root,root) %config(noreplace) /etc/proInst.loc

%files doc -f files-doc.list
%defattr(-,prophet,dba)

%files extra -f files-extra.list
%defattr(-,prophet,dba)

%files java -f files-java.list
%defattr(-,prophet,dba)

%files em -f files-em.list
%defattr(-,prophet,dba)

# A common mistake in the %%changelog is to use macros to fill in the version and
# release. You don't want to do this, because it will evaluate all the version-
# releases as the same. :)
%changelog
* Monday May 15 2006 Tom “spot” Callaway <tcallawa@redhat.com> 9.2.0.6.0-2
- Updated to 9.2.0.6.0

* Sunday May 14 2006 Tom “spot” Callaway <tcallawa@redhat.com> 9.2.0.6.0-1
- initial package
```