

Red Hat Developer Toolset 1.1

Software Collections Guide

A guide to Software Collections for Red Hat Enterprise Linux

Edition 0



Petr Kovář

Red Hat Engineering Content Services

pkovar@redhat.com

Legal Notice

Copyright © 2013 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at <http://creativecommons.org/licenses/by-sa/3.0/>. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, MetaMatrix, Fedora, the Infinity Logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle and/or its affiliates.

XFS® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

All other trademarks are the property of their respective owners.

1801 Varsity Drive
Raleigh, NC 27606-2072 USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701

Abstract

The Software Collections Guide provides an explanation of Software Collections and details how to build and package them. Developers and system administrators who have a basic understanding of software packaging with RPM packages, but who are new to the concept of Software Collections, can use this Guide to get started with Software Collections.

Table of Contents

Preface

1. Document Conventions
 - 1.1. Typographic Conventions
 - 1.2. Pull-quote Conventions
 - 1.3. Notes and Warnings
2. Getting Help and Giving Feedback
 - 2.1. Do You Need Help?
 - 2.2. We Need Feedback!
3. Acknowledgments

1. Introducing Software Collections

- 1.1. Why Package Software with RPM?
- 1.2. What Are Software Collections?
- 1.3. Enabling Support for Software Collections
- 1.4. The File System Hierarchy
- 1.5. Software Collection Scriptlets
- 1.6. Package Layout
 - 1.6.1. Meta Package
 - 1.6.2. Packages in the Software Collection Namespace
- 1.7. Enabling a Software Collection
 - 1.7.1. Running an Application Directly
 - 1.7.2. Running a Shell with Multiple Software Collections Enabled
 - 1.7.3. Running Commands Stored in a File

2. Packaging Software Collections

- 2.1. Building a Software Collection Locally
- 2.2. Software Collection Macros
 - 2.2.1. Macros Specific to a Software Collection
 - 2.2.2. Macros Not Specific to a Software Collection
- 2.3. Converting a Conventional Spec File
- 2.4. Using a Software Collection in Your Application
- 2.5. Installing a Software Collection
- 2.6. Listing Installed Software Collections
- 2.7. Software Collection Initscript Support
- 2.8. Software Collection Library Support
 - 2.8.1. Using a Library Outside of the Software Collection
 - 2.8.2. Prefixing the Library Major soname with the Software Collection Name
- 2.9. Software Collection .pc Files Support
- 2.10. Software Collection manpath Support
- 2.11. Software Collection SystemTap Support
- 2.12. Software Collection cronjob Support
- 2.13. Software Collection logrotate Support
- 2.14. Software Collection Lock File Support
- 2.15. Software Collection Configuration Files Support
- 2.16. Software Collection Kernel Module Support
- 2.17. Software Collection SELinux Support

3. Getting More Information

- 3.1. Red Hat Enterprise Linux Developer Program
- 3.2. Installed Documentation

3.3. Accessing Red Hat Documentation

A. Revision History

Preface

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the [Liberation Fonts](#) set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

Mono-spaced Bold

Used to highlight system input, including shell commands, file names and paths. Also used to highlight keys and key combinations. For example:

To see the contents of the file **my_next_bestselling_novel** in your current working directory, enter the **cat my_next_bestselling_novel** command at the shell prompt and press **Enter** to execute the command.

The above includes a file name, a shell command and a key, all presented in mono-spaced bold and all distinguishable thanks to context.

Key combinations can be distinguished from an individual key by the plus sign that connects each part of a key combination. For example:

Press **Enter** to execute the command.

Press **Ctrl+Alt+F2** to switch to a virtual terminal.

The first example highlights a particular key to press. The second example highlights a key combination: a set of three keys pressed simultaneously.

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in **mono-spaced bold**. For example:

File-related classes include **filesystem** for file systems, **file** for files, and **dir** for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialog box text; labeled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

Choose **System** → **Preferences** → **Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).

To insert a special character into a **gedit** file, choose **Applications** → **Accessories** → **Character Map** from the main menu bar. Next, choose **Search** → **Find...** from the

Character Map menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit** → **Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in proportional bold and all distinguishable by context.

Mono-spaced Bold Italic or *Proportional Bold Italic*

Whether mono-spaced bold or proportional bold, the addition of italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

To connect to a remote machine using ssh, type **ssh *username@domain.name*** at a shell prompt. If the remote machine is **example.com** and your username on that machine is john, type **ssh john@example.com**.

The **mount -o remount *file-system*** command remounts the named file system. For example, to remount the **/home** file system, the command is **mount -o remount /home**.

To see the version of a currently installed package, use the **rpm -q *package*** command. It will return a result as follows: ***package-version-release***.

Note the words in bold italics above — username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

Publican is a *DocBook* publishing system.

1.2. Pull-quote Conventions

Terminal output and source code listings are set off visually from the surrounding text.

Output sent to a terminal is set in **mono-spaced roman** and presented thus:

```
books      Desktop  documentation  drafts  mss    photos  stuff  svn
books_tests Desktop1  downloads      images  notes  scripts svgs
```

Source-code listings are also set in **mono-spaced roman** but add syntax highlighting as follows:

```

package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
    public static void main(String args[])
        throws Exception
    {
        InitialContext iniCtx = new InitialContext();
        Object          ref    = iniCtx.lookup("EchoBean");
        EchoHome        home   = (EchoHome) ref;
        Echo             echo   = home.create();

        System.out.println("Created Echo");

        System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
    }
}

```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.



Note

Notes are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.



Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled 'Important' will not cause data loss but may cause irritation and frustration.



Warning

Warnings should not be ignored. Ignoring warnings will most likely cause data loss.

2. Getting Help and Giving Feedback

2.1. Do You Need Help?

If you experience difficulty with a procedure described in this documentation, visit the Red Hat Customer Portal at <http://access.redhat.com>. Through the customer portal, you can:

- ▶ search or browse through a knowledgebase of technical support articles about Red Hat products.
- ▶ submit a support case to Red Hat Global Support Services (GSS).
- ▶ access other product documentation.

Red Hat also hosts a large number of electronic mailing lists for discussion of Red Hat software and

technology. You can find a list of publicly available mailing lists at <https://www.redhat.com/mailman/listinfo>. Click on the name of any mailing list to subscribe to that list or to access the list archives.

2.2. We Need Feedback!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in Bugzilla: <http://bugzilla.redhat.com/> against the product **Red Hat Developer Toolset**.

When submitting a bug report, be sure to mention the manual's identifier: *doc-Software_Collections_Guide*

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

3. Acknowledgments

The author of this book would like to thank the following people for their valuable contributions: Jindřich Nový, Marcela Mašláňová, Bohuslav Kabrda, Honza Horák, Jan Zelený, Langdon White, Florian Nadge, Stephen Wadeley, Douglas Silas, and Vít Ondruch, among many others.

Chapter 1. Introducing Software Collections

This chapter introduces you to the concept of Software Collections or SCLs for short.

1.1. Why Package Software with RPM?

The RPM Package Manager (RPM) is a package management system that runs on Red Hat Enterprise Linux. RPM makes it easier for you to distribute, manage, and update software that you create for Red Hat Enterprise Linux. Many software vendors distribute their software via a conventional archive file (such as a tarball). However, there are several advantages in packaging software into RPM packages. These advantages are outlined below.

With RPM, you can:

Install, reinstall, remove, upgrade and verify packages.

Users can use standard package management tools (for example **Yum** or **PackageKit**) to install, reinstall, remove, upgrade and verify your RPM packages.

Use a database of installed packages to query and verify packages.

Because RPM maintains a database of installed packages and their files, users can easily query and verify packages on their system.

Use metadata to describe packages, their installation instructions, and so on.

Each RPM package includes metadata that describes the package's components, version, release, size, project URL, installation instructions, and so on.

Package pristine software sources into source and binary packages.

RPM allows you to take pristine software sources and package them into source and binary packages for your users. In source packages, you have the pristine sources along with any patches that were used, plus complete build instructions. This design eases the maintenance of the packages as new versions of your software are released.

Add packages to Yum repositories.

You can add your package to a **Yum** repository that enables clients to easily find and deploy your software.

Digitally sign your packages.

Using a GPG signing key, you can digitally sign your package so that users are able to verify the authenticity of the package.

For in-depth information on what is RPM and how to use it, refer to the [Red Hat Enterprise Linux 6 Deployment Guide](#) or the [Red Hat Enterprise Linux 5 Deployment Guide](#).

1.2. What Are Software Collections?

With Software Collections, you can build and concurrently install multiple versions of the same software

components on your system. Software Collections have no impact on the system versions of the packages installed by any of the conventional RPM package management utilities.

Software Collections:

Do not overwrite system files

Software Collections are distributed as a set of several components, which provide their full functionality without overwriting system files.

Are designed to avoid conflicts with system files

Software Collections make use of a special file system hierarchy to avoid possible conflicts between a single Software Collection and the base system installation.

Require no changes to the RPM package manager

Software Collections require no changes to the RPM package manager present on the host system.

Need only minor changes to the spec file

To convert a conventional package to a single Software Collection, you only need to make minor changes to the package spec file.

Allow you to build a conventional package and a Software Collection package with a single spec file

With a single spec file, you can build both the conventional package and the Software Collection package.

Uniquely name all included packages

With Software Collection's namespace, all packages included in the Software Collection are uniquely named.

Do not conflict with updated packages

Software Collection's namespace ensures that updating packages on your system causes no conflicts.

Can depend on other Software Collections

Because one Software Collection can depend on another, you can define multiple levels of dependencies.

1.3. Enabling Support for Software Collections

To enable support for Software Collections on your system so that you can enable and build Software Collections, you need to have installed the packages *scl-utils* and *scl-utils-build*.

If the packages *scl-utils* and *scl-utils-build* are not already installed on your system, you can install them

by typing the following at a shell prompt as root:

```
yum install -y scl-utils scl-utils-build
```

The *scl-utils* package provides the **scl** tool that lets you enable Software Collections on your system. For more information on enabling Software Collections, refer to [Section 1.7, “Enabling a Software Collection”](#).

The *scl-utils-build* package provides macros that are essential for building Software Collections. For more information on building Software Collections, refer to [Section 2.1, “Building a Software Collection Locally”](#).

1.4. The File System Hierarchy

The root directory of Software Collections is normally located in the **/opt/** directory to avoid possible conflicts between Software Collections and the base system installation. The use of the **/opt/** directory is recommended by the Filesystem Hierarchy Standard (FHS).

You can change the location of the root directory by setting the **%_scl_prefix** macro in the spec file, as in the following example:

```
%_scl_prefix /opt/provider
```

where **provider** is the provider (vendor) name registered with the Linux Foundation and the subordinated Linux Assigned Names and Numbers Authority (LANANA), in conformance with the Filesystem Hierarchy Standard.

Each organization or project that builds and distributes Software Collections should use its own provider name, which conforms to the Filesystem Hierarchy Standard (FHS) and avoids possible conflicts between Software Collections and the base system installation.

You are advised to make the file system hierarchy conform to the following layout:

```
/opt/provider/application-version/
```

For more information on the Filesystem Hierarchy Standard, refer to <http://www.pathname.com/fhs/>.

For more information on the Linux Assigned Names and Numbers Authority, refer to <http://www.lanana.org/>.

Below is an example of the file system hierarchy layout with two Software Collections, **Software Collection 1** and **Software Collection 2**:

```
opt
`-- provider
    |-- Software Collection 1
    |   |-- Software Collection root directory
    |   `-- Software Collection scriptlets
    |
    `-- Software Collection 2
        |-- Software Collection root directory
        `-- Software Collection scriptlets
```

As you can see in the example above, each of the Software Collections directories contains two subdirectories: the Software Collection root directory and a directory containing the Software Collection scriptlets. For more information on the Software Collection scriptlets, refer to [Section 1.5, “Software Collection Scriptlets”](#).

1.5. Software Collection Scriptlets

The Software Collection scriptlets are simple shell scripts that change the current system environment so that the group of packages in the Software Collection is preferred over the corresponding group of conventional packages installed on the system.

To utilize the Software Collection scriptlets, use the **scl** tool. For more information on **scl**, refer to [Section 1.7, “Enabling a Software Collection”](#).

1.6. Package Layout

Each Software Collection's layout consists of the meta package, which installs a subset of other packages, and a number of the Software Collection's packages, which are installed within the Software Collection namespace.

1.6.1. Meta Package

Each Software Collection includes a meta package, which installs a minimal subset of essential packages. For example, the essential packages can provide the Perl language interpreter, but no Perl extension modules. The meta package contains a basic file system hierarchy and delivers a number of the Software Collection's scriptlets.

The purpose of the meta package is to make sure that all essential packages in the Software Collection are properly installed and that it is possible to enable the Software Collection.

The meta package produces the following packages that are also part of the Software Collection:

The main package: %scl

The main package in the Software Collection contains dependencies of the base packages, which are included in the Software Collection. The main package does not contain any files.

For example, if the name of the Software Collection is **ruby-1.8.7**, then the main package macro is expanded to:

```
ruby-1.8.7
```

The runtime subpackage: *name*-runtime

The runtime subpackage in the Software Collection owns the Software Collection's file system and delivers the Software Collection's scriptlets.

For example, if the name of the Software Collection is **ruby-1.8.7**, then the runtime subpackage macro is expanded to:

```
ruby-1.8.7-runtime
```

The build subpackage: *name*-build

The build subpackage in the Software Collection delivers the Software Collection's build configuration. The build subpackage is optional and can be excluded from the Software Collection.

For example, if the name of the Software Collection is **ruby-1.8.7**, then the build subpackage

macro is expanded to:

```
ruby-1.8.7-build
```

Example of the Meta Package

To get an idea of what a typical Software Collection meta package looks like, see the following example:

```

%{!?scl:%global scl example}
%scl_package %scl

Summary: Package that installs %scl
Name: %scl_name
Version: 1
Release: 1%{?dist}
BuildArch: noarch
License: GPLv2+
Requires: %{scl_prefix}less
BuildRequires: scl-utils-build

%description
This is the main package for %scl Software Collection.

%package runtime
Summary: Package that handles %scl Software Collection.
Requires: scl-utils

%description runtime
Package shipping essential scripts to work with %scl Software Collection.

%package build
Summary: Package shipping basic build configuration

%description build
Package shipping essential configuration macros to build %scl Software Collection.

%prep
%setup -c -T

%install
rm -rf %{buildroot}
mkdir -p %{buildroot}%{_scl_scripts}/root
cat >> %{buildroot}%{_scl_scripts}/enable << EOF
export PATH=%{_bindir}:\$PATH
EOF
%scl_install

%files
%files runtime
%scl_files

%files build
%{_root_sysconfdir}/rpm/macros.%{scl}-config

%changelog
* Thu Jan 07 2012 John Doe <jdoe@example.com> 1-1
- Initial package

```

1.6.2. Packages in the Software Collection Namespace

Besides the minimal subset of packages installed by the meta package, the Software Collection also installs a number of packages that are part of the package collection itself.

Software Collection package: *name-pkgname*

The Software Collection can contain a number of packages, which are included in the Software Collection namespace and relocated to the Software Collection's file system hierarchy.

For example, if the name of the Software Collection is **ruby-1.8.7**, then the Software Collection's package name for the *ruby-1.8.7.352-2.1.el6.x86_64.rpm* package is:

```
ruby-1.8.7-ruby-1.8.7.352-2.1.el6.x86_64.rpm
```

1.7. Enabling a Software Collection

The **scl** tool is used to enable a Software Collection and to run applications in the Software Collection environment.

General usage of the **scl** tool can be described using the following syntax:

```
scl action software_collection_1 software_collection_2 command
```

1.7.1. Running an Application Directly

For example, to directly run **Perl** with the **--version** option in the Software Collection named **software_collection_1**, execute the following command:

```
scl enable software_collection_1 'perl --version'
```

1.7.2. Running a Shell with Multiple Software Collections Enabled

To run the **Bash** shell in the environment with multiple Software Collections enabled, execute the following command:

```
scl enable software_collection_1 software_collection_2 bash
```

The command above enables two Software Collections named **software_collection_1** and **software_collection_2**.

1.7.3. Running Commands Stored in a File

To execute a number of commands, which are stored in a file, in the Software Collection environment, run the following command:

```
cat cmd | scl enable software_collection_1 -
```

The command above executes commands, which are stored in the **cmd** file, in the environment of the Software Collection named **software_collection_1**.

Chapter 2. Packaging Software Collections

This chapter introduces you to packaging Software Collections.

2.1. Building a Software Collection Locally

To build a Software Collection on your system, run the following command:

```
rpmbuild -ba package.spec --define 'scl name'
```

The difference between the command shown above and the standard command to build conventional packages (**rpmbuild -bb package.spec**) is that you have to append the **--define** option to the **rpmbuild** command when building a Software Collection.

The **--define** option defines the **scl** macro, which uses the Software Collection configured in the Software Collection spec file.

2.2. Software Collection Macros

The Software Collection packaging macro **scl** relocates the file structure, which is part of the Software Collection, to a file system that is used exclusively by the Software Collection.

The **scl** macro also defines files ownership for the Software Collection's meta package and provides additional packaging macros to use in the Software Collection environment.

When using Software Collection macros in the spec file, you must prefix them with **%{?scl:macro}**, as in the following example:

```
%{?scl:Requires:%scl_runtime}
```

In the example above, the **%scl_runtime** macro is the value of the **Requires** tag. Both the macro and the tag use the **%{?scl:}** prefix.

2.2.1. Macros Specific to a Software Collection

The table below shows a list of all macros specific to a particular Software Collection.

Table 2.1. Software Collection Specific Macros

Macro	Description	Example value
%scl_name	name of the Software Collection	software_collection_1
%pkg_name	name of the original package	perl
_%scl_prefix	root of the Software Collection (not package's root)	/opt/provider/
_%scl_scripts	location of Software Collection's scriptlets	/opt/provider/software_collection_1/
_%scl_root	installation root (install-root) of the package	/opt/provider/software_collection_1/root/
_%scl_require_package	depend on a particular package from the Software Collection	software_collection_1-package_2

2.2.2. Macros Not Specific to a Software Collection

The table below shows a list of macros that are not specific to a particular Software Collection. Because these macros are not relocated and do not point to the Software Collection file system, they allow you to point to the system root file system. These macros use `_root` as a prefix.

Table 2.2. Software Collection Non-Specific Macros

Macro	Description	Relocated	Example value
<code>%_root_prefix</code>	Software Collection's <code>%_prefix</code> macro	no	<code>/usr/</code>
<code>%_root_exec_prefix</code>	Software Collection's <code>%_exec_prefix</code> macro	no	<code>/usr/</code>
<code>%_root_bindir</code>	Software Collection's <code>%_bindir</code> macro	no	<code>/usr/bin/</code>
<code>%_root_sbindir</code>	Software Collection's <code>%_sbindir</code> macro	no	<code>/usr/sbin/</code>
<code>%_root_datadir</code>	Software Collection's <code>%_datadir</code> macro	no	<code>/usr/share/</code>
<code>%_root_sysconfdir</code>	Software Collection's <code>%_sysconfdir</code> macro	no	<code>/etc/</code>
<code>%_root_libexecdir</code>	Software Collection's <code>%_libexecdir</code> macro	no	<code>/usr/libexec/</code>
<code>%_root_sharedstatedir</code>	Software Collection's <code>%_sharedstatedir</code> macro	no	<code>/usr/com/</code>
<code>%_root_localstatedir</code>	Software Collection's <code>%_localstatedir</code> macro	no	<code>/usr/var/</code>
<code>%_root_includedir</code>	Software Collection's <code>%_includedir</code> macro	no	<code>/usr/include/</code>
<code>%_root_infodir</code>	Software Collection's <code>%_infodir</code> macro	no	<code>/usr/share/info/</code>
<code>%_root_mandir</code>	Software Collection's <code>%_mandir</code> macro	no	<code>/usr/share/man/</code>
<code>%_root_initddir</code>	Software Collection's <code>%_initddir</code> macro	no	<code>/etc/rc.d/init.d/</code>
<code>%_root_libdir</code>	Software Collection's <code>%_libdir</code> macro, this macro does not work if Software Collection's meta package is platform-independent	no	<code>/usr/lib/</code>

2.3. Converting a Conventional Spec File

The following steps show how to convert a conventional spec file into a Software Collection spec file so that the Software Collection spec file that you can use in both the conventional package and the Software Collection.

Procedure 2.1. Converting a Conventional Spec File into a Software Collection Spec File

1. Add the `%scl_package` macro to the spec file. Place the macro in front of the spec file preamble as follows:

```
%{?scl:%scl_package package_name}
```

2. You are advised to define the `%pkg_name` macro in the spec file in case the package is not built for the Software Collection:

```
%{!?scl:%global pkg_name %{name}}
```

Consequently, you can use the `%pkg_name` macro to define the original name of the package wherever it is needed in the spec file that you can then use for building both the conventional package and the Software Collection.

3. Change the **Name** tag in the spec file preamble as follows:

```
Name: %{?scl_prefix}package_name
```

4. To check that all essential Software Collection's packages are dependencies of the main meta package, add the following macro after the **BuildRequires** or **Requires** tags in the spec file:

```
%{?scl:Requires: %scl_runtime}
```

5. Edit the `%setup` macro in the `%prep` section of the spec file so that the macro can deal with a different package name in the Software Collection environment:

```
%setup -q -n %{pkg_name}-%{version}
```

Example of the Converted Spec File

To see what the diff file comparing a conventional spec file with a converted spec file looks like, see the following example:

```

--- a/less.spec
+++ b/less.spec
@@ -1,10 +1,13 @@
+{%?scl:%scl_package less}
+{%!?!scl:%global pkg_name %{name}}
+
+  Summary: A text file browser similar to more, but better
-Name: less
+Name: {%?scl_prefix}less
+  Version: 444
+  Release: 7{%?dist}
+  License: GPLv3+
+  Group: Applications/Text
-Source: http://www.greenwoodsoftware.com/less/{%name}-{%version}.tar.gz
+Source: http://www.greenwoodsoftware.com/less/{%pkg_name}-{%version}.tar.gz
+  Source1: lesspipe.sh
+  Source2: less.sh
+  Source3: less.csh
@@ -19,6 +22,7 @@ URL: http://www.greenwoodsoftware.com/less/
+  Requires: groff
+  BuildRequires: ncurses-devel
+  BuildRequires: autoconf automake libtool
+{%?scl:Requires: %scl_runtime}

%description
The less utility is a text file browser that resembles more, but has
@@ -31,7 +35,7 @@ You should install less because it is a basic utility for
viewing text
files, and you'll use it frequently.

%prep
-%setup -q
+%setup -q -n {%pkg_name}-{%version}
+  %patch1 -p1 -b .Foption
+  %patch2 -p1 -b .search
+  %patch4 -p1 -b .time
@@ -51,16 +55,16 @@ make CC="gcc $RPM_OPT_FLAGS -D_GNU_SOURCE -
D_LARGEFILE_SOURCE -D_LARGEFILE64_SOU
%install
rm -rf $RPM_BUILD_ROOT
make DESTDIR=$RPM_BUILD_ROOT install
-mkdir -p $RPM_BUILD_ROOT/etc/profile.d
+mkdir -p $RPM_BUILD_ROOT{%_sysconfdir}/profile.d
+  install -p -c -m 755 {%SOURCE1} $RPM_BUILD_ROOT/{%_bindir}
-  install -p -c -m 644 {%SOURCE2} $RPM_BUILD_ROOT/etc/profile.d
-  install -p -c -m 644 {%SOURCE3} $RPM_BUILD_ROOT/etc/profile.d
-  ls -la $RPM_BUILD_ROOT/etc/profile.d
+  install -p -c -m 644 {%SOURCE2} $RPM_BUILD_ROOT{%_sysconfdir}/profile.d
+  install -p -c -m 644 {%SOURCE3} $RPM_BUILD_ROOT{%_sysconfdir}/profile.d
+  ls -la $RPM_BUILD_ROOT{%_sysconfdir}/profile.d

%files
%defattr(-,root,root,-)
%doc LICENSE
-/etc/profile.d/*
+{%_sysconfdir}/profile.d/*
+{%_bindir}/*
+{%_mandir}/man1/*

```

2.4. Using a Software Collection in Your Application

To use a Software Collection in your application, you need to adjust the **BuildRequires** and **Requires** tags in your application's spec file so that these tags properly define dependencies on Software Collections.

For example, to define dependencies on two Software Collections named **software_collection_1** and **software_collection_2**, add the following three lines to your application's spec file:

```
BuildRequires: scl-utils-build
Requires: %scl_require software_collection_1
Requires: %scl_require software_collection_2
```

2.5. Installing a Software Collection

To ensure that a Software Collection is on your system, install the main meta package of the Software Collection. You can use conventional tools like **Yum** or **PackageKit** for this task because Software Collections are fully compatible with the RPM Package Manager. For example, to install a Software Collection named **software_collection_1** with **Yum**, run the following command:

```
yum install -y software_collection_1
```

This command will automatically install all the packages that are part of the Software Collection. Also, if you install an application that depends on a Software Collection, the Software Collection will be installed along with the rest of the application's dependencies.

For detailed information on **Yum** and **PackageKit** usage, refer to the [Red Hat Enterprise Linux 6 Deployment Guide](#).

2.6. Listing Installed Software Collections

To get a list of Software Collections that are currently installed on the system, run the following command:

```
scl --list
```

2.7. Software Collection Initscript Support

Ensure that users can directly manage any services provided by the Software Collection or one of the associated applications with the system default tools, like **service** or **chkconfig**.

To avoid possible name conflicts with the system versions of the services that are part of the Software Collection, make sure to adjust the **%install** section of the spec file as follows:

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT{%?scl:%_root_sysconfdir}%{!?
scl:%_sysconfdir}/rc.d/%{?scl_prefix}service_name
```

With this configuration in place, you can then refer to the version of the service included in the Software Collection as follows:

```
%{?scl_prefix}service_name
```

2.8. Software Collection Library Support

In case you distribute libraries that you intend to use only in the Software Collection environment or in addition to the libraries available on the system, adjust the **LD_LIBRARY_PATH** environment variable in the spec file as follows:

```
export LD_LIBRARY_PATH=%{_libdir}:$LD_LIBRARY_PATH
```

This configuration ensures that the version of the library in the Software Collection is preferred over the version of the library available on the system if the Software Collection is enabled.



Note

In case you distribute a private shared library in the Software Collection, consider using the **DT_RUNPATH** attribute instead of the **LD_LIBRARY_PATH** environment variable to make the private shared library accessible in the Software Collection environment.

2.8.1. Using a Library Outside of the Software Collection

If you distribute libraries that you intend to use outside of the Software Collection environment, you can use the directory **/etc/ld.so.conf.d/** for this purpose.



Warning

Do not use **/etc/ld.so.conf.d/** for libraries already available on the system. Using **/etc/ld.so.conf.d/** is only recommended for a library that is not available on the system, as otherwise the version of the library in the Software Collection might get preference over the system version of the library. That could lead to undesired behavior of the system versions of the applications, including unexpected termination and data loss.

Procedure 2.2. Using **/etc/ld.so.conf.d/** for libraries in the Software Collection

1. Create a file named **libs.conf** and adjust the spec file configuration accordingly:

```
SOURCE2: %{?scl_prefix}libs.conf
```

2. In the **libs.conf** file, include a list of directories where the versions of the libraries associated with the Software Collection are located. For example:

```
/opt/provider/software_collection_1/root/usr/lib64/
```

In the example above, the **/usr/lib64/** directory that is part of the Software Collection **software_collection_1** is included in the list.

3. Edit the **%install** section of the spec file, so the **libs.conf** file is installed as follows:

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?scl:%_sysconfdir}/ld.so.conf.d/
```

2.8.2. Prefixing the Library Major soname with the Software Collection Name

When using libraries included in the Software Collection, always remember that a library with the same

major soname can already be available on the system as a part of the base system installation. It is thus important not to forget to use the **scl enable** command when building an application against a library included in the Software Collection. Failing to do so may result in the application being executed in an incorrect environment, linked against the incorrect system version of the library.



Warning

Keep in mind that executing your application in an incorrect environment (for example in the system environment instead of the Software Collection environment) as well as linking your application against an incorrect library can lead to undesired behavior of your application, including unexpected termination and data loss.

To ensure that your application is not linked against an incorrect library even if the **LD_LIBRARY_PATH** environment variable has not been set properly, change the major soname of the library included in the Software Collection. The recommended way to change the major soname is to prefix the major soname version number with the Software Collection name.

Below is an example of the MySQL client library with the **mysql155-** prefix:

```
$ rpm -ql mysql155-mysql-libs | grep 'lib.*so'
/usr/lib64/mysql/libmysqlclient.so.mysql155-18
/usr/lib64/mysql/libmysqlclient.so.mysql155-18.0.0
```

On the same system, the system version of the MySQL client library is listed below:

```
$ rpm -ql mysql-libs | grep 'lib.*so'
/usr/lib64/mysql/libmysqlclient.so.18
/usr/lib64/mysql/libmysqlclient.so.18.0.0
```

2.9. Software Collection .pc Files Support

The .pc files are special metadata files used by the **pkg-config** program to store information about libraries available on the system. In case you distribute .pc files that you intend to use only in the Software Collection environment or in addition to the .pc files installed on the system, adjust the **PKG_CONFIG_PATH** environment variable. Depending on what is defined in your .pc files, adjust the **PKG_CONFIG_PATH** environment variable for the **%{_libdir}** macro (which expands to the library directory, typically **/usr/lib/** or **/usr/lib64/**), or for the **%{_datadir}** macro (which expands to the share directory, typically **/usr/share/**).

If the library directory is defined in your .pc files, adjust the **PKG_CONFIG_PATH** environment variable in the spec file as follows:

```
export PKG_CONFIG_PATH=%{_libdir}/pkgconfig:\$PKG_CONFIG_PATH
```

If the share directory is defined in your .pc files, adjust the **PKG_CONFIG_PATH** environment variable in the spec file as follows:

```
export PKG_CONFIG_PATH=%{_datadir}/pkgconfig:\$PKG_CONFIG_PATH
```

This configuration ensures that the .pc files in the Software Collection are preferred over the .pc files available on the system if the Software Collection is enabled.

The Software Collection can provide a wrapper script that is visible to the system to enable the Software

Collection, for example in the `/usr/bin/` directory. In this case, ensure that the `.pc` files are visible to the system even if the Software Collection is disabled.

To allow your system to use `.pc` files from the disabled Software Collection, update the **PKG_CONFIG_PATH** environment variable with the paths to the `.pc` files associated with the Software Collection. Depending on what is defined in your `.pc` files, adjust the **PKG_CONFIG_PATH** environment variable for the `%{_libdir}` macro (which expands to the library directory), or for the `%{_datadir}` macro (which expands to the share directory).

Procedure 2.3. Updating the **PKG_CONFIG_PATH** environment variable for `%{_libdir}`

1. To update the **PKG_CONFIG_PATH** environment variable for the `%{_libdir}` macro, create a custom script `/etc/profile.d/name.sh`. The script is preloaded when a shell is started on the system.

For example, create the following file:

```
%{?scl_prefix}pc-libdir.sh
```

2. Use the `pc-libdir.sh` short script that modifies the **PKG_CONFIG_PATH** variable to refer to your `.pc` files:

```
export
PKG_CONFIG_PATH=%{_libdir}/pkgconfig:/opt/provider/software_collection/path/
to/your/pc_files
```

3. Add the file to your Software Collection package's spec file:

```
SOURCE2: %{?scl_prefix}pc-libdir.sh
```

4. Install this file into the system `/etc/profile.d/` directory by adjusting the `%install` section of the Software Collection package's spec file:

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!}?
scl:%_sysconfdir}/profile.d/
```

Procedure 2.4. Updating the **PKG_CONFIG_PATH** environment variable for `%{_datadir}`

1. To update the **PKG_CONFIG_PATH** environment variable for the `%{_datadir}` macro, create a custom script `/etc/profile.d/name.sh`. The script is preloaded when a shell is started on the system.

For example, create the following file:

```
%{?scl_prefix}pc-datadir.sh
```

2. Use the `pc-datadir.sh` short script that modifies the **PKG_CONFIG_PATH** variable to refer to your `.pc` files:

```
export
PKG_CONFIG_PATH=%{_datadir}/pkgconfig:/opt/provider/software_collection/path/
to/your/pc_files
```

3. Add the file to your Software Collection package's spec file:

```
SOURCE2: %{?scl_prefix}pc-datadir.sh
```

4. Install this file into the system `/etc/profile.d/` directory by adjusting the `%install` section of the Software Collection package's spec file:

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?
scl:%_sysconfdir}/profile.d/
```

2.10. Software Collection manpath Support

To allow the `man` command on the system to display manual pages from the enabled Software Collection, update the `MANPATH` environment variable with the paths to the manual pages that are associated with the Software Collection.

To update the `MANPATH` environment variable, add the following line to the spec file:

```
export MANPATH=${MANPATH}:%{_mandir}
```

This update relocates the `%{_mandir}` macro to the Software Collection path. So that the manual pages associated with the Software Collection are not visible as long as the Software Collection is not enabled.

The Software Collection can provide a wrapper script that is visible to the system to enable the Software Collection, for example in the `/usr/bin/` directory. In this case, ensure that the manual pages are visible to the system even if the Software Collection is disabled.

To allow the `man` command on the system to display manual pages from the disabled Software Collection, update the `MANPATH` environment variable with the paths to the manual pages associated with the Software Collection.

Procedure 2.5. Updating the `MANPATH` environment variable for the disabled Software Collection

1. To update the `MANPATH` environment variable, create a custom script `/etc/profile.d/name.sh`. The script is preloaded when a shell is started on the system. For example, create the following file:

```
%{?scl_prefix}manpage.sh
```

2. Use the `manpage.sh` short script that modifies the `MANPATH` variable to refer to your man path directory:

```
export
MANPATH=${MANPATH}:/opt/provider/software_collection/path/to/your/man_pages
```

3. Add the file to your Software Collection package's spec file:

```
SOURCE2: %{?scl_prefix}manpage.sh
```

4. Install this file into the system `/etc/profile.d/` directory by adjusting the `%install` section of the Software Collection package's spec file:

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?
scl:%_sysconfdir}/profile.d/
```


2.11. Software Collection SystemTap Support

SystemTap is a useful instrumentation platform for probing running processes and kernel activity on the Linux system. In case you distribute the SystemTap data directory for cached SystemTap data files that you intend to use only in the Software Collection environment or in addition to the SystemTap data files available on the system, adjust the **XDG_DATA_DIRS** environment variable in the spec file as follows:

```
export XDG_DATA_DIRS=%{_datadir}:\$XDG_DATA_DIRS
```

This configuration ensures that the SystemTap data files in the Software Collection are preferred over the SystemTap data files available on the system if the Software Collection is enabled.

The Software Collection can provide a wrapper script that is visible to the system to enable the Software Collection, for example in the **/usr/bin/** directory. In this case, ensure that the SystemTap data files are visible to the system even if the Software Collection is disabled.

To allow your system to use SystemTap data files from the disabled Software Collection, update the **XDG_DATA_DIRS** environment variable with the paths to the SystemTap data files associated with the Software Collection.

Procedure 2.6. Updating the XDG_DATA_DIRS environment variable for the disabled Software Collection

1. To update the **XDG_DATA_DIRS** environment variable, create a custom script **/etc/profile.d/name.sh**. The script is preloaded when a shell is started on the system.

For example, create the following file:

```
%{?scl_prefix}systemtap.sh
```

2. Use the **systemtap.sh** short script that modifies the **XDG_DATA_DIRS** variable to refer to your SystemTap data files directory:

```
export
XDG_DATA_DIRS=${XDG_DATA_DIRS}:/opt/provider/software_collection/path/to/your/systemtap_data_files
```

3. Add the file to your Software Collection package's spec file:

```
SOURCE2: %{?scl_prefix}systemtap.sh
```

4. Install this file into the system **/etc/profile.d/** directory by adjusting the **%install** section of the Software Collection package's spec file:

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{! ?
scl:%_sysconfdir}/profile.d/
```

2.12. Software Collection cronjob Support

With your Software Collection, you can run regular tasks on the system either with a dedicated service or with cronjobs. If you intend to use a dedicated service, refer to [Section 2.7, “Software Collection Initscript Support”](#) on how to work with initscripts in the Software Collection environment.

Procedure 2.7. Running regular tasks with cronjobs

1. To use cronjobs for running regular tasks, place a **crontab** file for your Software Collection in the **/etc/cron.d/** directory with the Software Collection's name.

For example, create the following file:

```
#{?scl_prefix}crontab
```

2. Ensure that the contents of the **crontab** file follow the standard **crontab** file format, as in the following example:

```
0 1 * * Sun root
/opt/provider/software_collection/architecture/usr/bin/cron_job_name
```

3. Add the file to your spec file of the Software Collection package:

```
SOURCE2: #{?scl_prefix}crontab
```

4. Install the file into the system directory **/etc/cron.d/** by adjusting the **%install** section of the Software Collection package's spec file:

```
%install
install -p -c -m 644 #{SOURCE2} $RPM_BUILD_ROOT#{?scl:%_root_sysconfdir}%{!?
scl:%_sysconfdir}/cron.d/
```

2.13. Software Collection logrotate Support

With your Software Collection or an application associated with your Software Collection, you can manage log files with the **logrotate** program.

Procedure 2.8. Managing log files with logrotate

1. To manage your log files with **logrotate**, place a custom **logrotate** file for your Software Collection in the system directory for the **logrotate** jobs **/etc/logrotate.d/**.

For example, create the following file:

```
#{?scl_prefix}logrotate
```

2. Ensure that the contents of the **logrotate** file follow the standard **logrotate** file format as follows:

```
/opt/provider/software_collection/var/log/your_application_name.log {
    missingok
    notifempty
    size 30k
    yearly
    create 0600 root root
}
```

3. Add the file to your spec file of the Software Collection package:

```
SOURCE2: #{?scl_prefix}logrotate
```

4. Install the file into the system directory **/etc/logrotate.d/** by adjusting the **%install** section of the Software Collection package's spec file:

```
%install
install -p -c -m 644 %{SOURCE2} $RPM_BUILD_ROOT%{?scl:%_root_sysconfdir}%{!?\
scl:%_sysconfdir}/logrotate.d/
```

2.14. Software Collection Lock File Support

If you store your Software Collection's lock files within the `/opt/provider/software_collection/` file system hierarchy, you can avoid any possible conflicts with the system versions of the applications or services that can be on the system.

If you want to prevent Software Collection's applications or services from running while the system version of the respective application or service is running, make sure that your applications or services, which require a lock, write the lock to the system directory `/var/lock/` instead of the Software Collection's directory `/opt/provider/software_collection/var/lock/`. In this way, your applications or services' lock file will not be overwritten. The lock file will not be renamed and the name stays the same as the system version.

If you want your Software Collection's version of the application or service to run concurrently with the system version (when the Software Collection version's resources will not conflict with the system version's resources), ensure that the applications or services write the lock to the Software Collection's directory `/opt/provider/software_collection/var/lock/`.

2.15. Software Collection Configuration Files Support

If you store your Software Collection's configuration files within the `/opt/provider/software_collection/` file system hierarchy, you can avoid any possible conflicts with the system versions of the configuration files that can be present on the system.

If you cannot store the configuration files within `/opt/provider/software_collection/`, then ensure that you properly configure an alternative location for the configuration files. For many programs, this can be usually done at build or installation time.

2.16. Software Collection Kernel Module Support

Because Linux kernel modules are normally tied to a particular version of the Linux kernel, you must be careful when you package kernel modules into a Software Collection. This is because the package management system on Red Hat Enterprise Linux does not automatically update or install an updated version of the kernel module if an updated version of the Linux kernel is installed. To make packaging the kernel modules into the Software Collection easier, see the following recommendations. Ensure that:

1. the name of your kernel module package includes the kernel version,
2. the tag **Requires**, which can be found in your kernel module spec file, includes the kernel version and revision (in the format **kernel-version-revision**).

2.17. Software Collection SELinux Support

Because Software Collections are designed to install the Software Collection packages in an alternate directory, set up the necessary SELinux labels so that SELinux is aware of the alternate directory.

If the file system hierarchy of your Software Collection package imitates the file system hierarchy of the corresponding conventional package, you can run the **semanage fcontext** and **restorecon** commands to set up the SELinux labels.

For example, if the `/opt/provider/software_collection_1/x86_64/root/usr/` directory in your Software Collection package imitates the `/usr/` directory of your conventional package, set up the SELinux labels as follows:

```
semanage fcontext -a -e /usr
/opt/provider/software_collection_1/x86_64/root/usr
```

```
restorecon -R -v /opt/provider/software_collection_1/x86_64/root/usr
```

The commands above ensure that all directories and files in the `/opt/provider/software_collection_1/x86_64/root/usr/` directory are labeled by SELinux as if they were located in the `/usr/` directory.

To use the `semanage fcontext` and `restorecon` commands in your Software Collection, add the commands to the `%post` post-installation scriptlet in the Software Collection spec file.



Important

Keep in mind that the `semanage -e` command, which substitutes the source path for the destination path during labeling, is not supported in Red Hat Enterprise Linux 5.

Chapter 3. Getting More Information

For more information on Software Collection packaging, Red Hat Enterprise Linux Developer Program, Red Hat Developer Toolset, and Red Hat Enterprise Linux, refer to the resources listed below.

3.1. Red Hat Enterprise Linux Developer Program

- ▶ [Red Hat Enterprise Linux Developer Program](#) – The *Red Hat Enterprise Linux Developer Program* delivers industry-leading developer tools, instructional resources, and an ecosystem of experts to help Linux programmers maximize productivity in building Linux applications.
- ▶ [Red Hat Enterprise Linux Developer Program Group](#) (requires Red Hat Login) – The *Red Hat Enterprise Linux Developer Program Group* contains developer-related information for the development tools available for Red Hat Enterprise Linux. In addition, users can find there developer-related papers and videos on topics that are of interest to developers, for example RPM building, threaded programming, performance tuning, debugging, and so on.

3.2. Installed Documentation

- ▶ **scl(1)** – The manual page for the **scl** tool for enabling Software Collections and running programs in Software Collection's environment.
- ▶ **scl --help** – General usage information for the **scl** tool for enabling Software Collections and running programs in Software Collection's environment.
- ▶ **rpmbuild(8)** – The manual page for the **rpmbuild** utility for building both binary and source packages.

3.3. Accessing Red Hat Documentation

The **Red Hat Documentation** portal located at <https://access.redhat.com/knowledge/docs/> serves as a central source of all product documentation. It is translated in 22 languages and for each product, it provides different kinds of books from release and technical notes to installation, user, and reference guides in HTML, PDF, and EPUB formats.

The following is a brief list of documents that are directly or indirectly relevant to this book:

- ▶ [Red Hat Developer Toolset 1.1 User Guide](#) – The *User Guide* for Red Hat Developer Toolset 1.1 contains information about Red Hat Developer Toolset, a Red Hat offering for developers on the Red Hat Enterprise Linux platform. Using Software Collections, Red Hat Developer Toolset provides current versions of the **GCC** compiler, **GDB** debugger and other binary utilities.
- ▶ [Red Hat Enterprise Linux 6 Developer Guide](#) – The *Developer Guide* for Red Hat Enterprise Linux 6 provides detailed information on the Eclipse IDE, libraries and runtime support, compiling and building, debugging, and profiling.
- ▶ [Red Hat Enterprise Linux 6 Installation Guide](#) – The *Installation Guide* for Red Hat Enterprise Linux 6 provides more details on getting, installing, and updating the system.
- ▶ [Red Hat Enterprise Linux 5 Installation Guide](#) – The *Installation Guide* for Red Hat Enterprise Linux 5 provides more details on getting, installing, and updating the system.
- ▶ [Red Hat Enterprise Linux 6 Deployment Guide](#) – The *Deployment Guide* for Red Hat Enterprise Linux 6 documents relevant information regarding the deployment, configuration, and administration of Red Hat Enterprise Linux 6.
- ▶ [Red Hat Enterprise Linux 5 Deployment Guide](#) – The *Deployment Guide* for Red Hat Enterprise Linux 5 documents relevant information regarding the deployment, configuration, and administration of Red Hat Enterprise Linux 5.

Revision History

Revision 1.0-1	Tue Jan 22 2013	Petr Kovář
Red Hat Developer Toolset 1.1 release of the Software Collections Guide.		
Revision 1.0-2	Thu Nov 08 2012	Petr Kovář
Red Hat Developer Toolset 1.1 Beta-2 release of the Software Collections Guide.		
Revision 1.0-1	Wed Oct 10 2012	Petr Kovář
Red Hat Developer Toolset 1.1 Beta-1 release of the Software Collections Guide.		
Revision 1.0-0	Tue Jun 26 2012	Petr Kovář
Red Hat Developer Toolset 1.0 release of the Software Collections Guide.		
Revision 0.0-2	Tue Apr 10 2012	Petr Kovář
Red Hat Developer Toolset 1.0 Alpha-2 release of the Software Collections Guide.		
Revision 0.0-1	Tue Mar 06 2012	Petr Kovář
Red Hat Developer Toolset 1.0 Alpha-1 release of the Software Collections Guide.		
Revision 0.0-0	Thu Feb 23 2012	Petr Kovář
Initial creation of book.		