

RPMSpecsEditor Users Guide

Table of Contents

Introduction.....	3
Overview.....	3
SPEC File.....	4
Variables.....	4
Requires Variable Entry.....	6
Variable Editor.....	6
Description.....	7
Expressions.....	7
Additional Executables.....	8
Symbolic Links.....	9
Libraries.....	9
Target Library Directories.....	10
Source Libraries.....	10
Source Library Versions.....	11
Rebuild Libraries.....	11
Requires.....	11
Change Log.....	13
Options.....	14
RPM Creation Example.....	16
RPMBuild Directory Setup.....	16
Source Files.....	16
SPEC File.....	16
Desktop Menu Items.....	16
Build RPM.....	17
RPM Signing.....	17
Troubleshooting.....	17
Example Application Launch Problem.....	17
Finding Missing Libraries.....	18
Installing Missing Libraries.....	18
Revision History.....	19

RPM Specs Editor Users Guide

Introduction

The *RPM Specs Editor* utility allows the user to create a SPEC file that is read by the *RPMBuild* utility in order to create an RPM file. An RPM file is a package format used by the RPM Package Manager, primarily for installing software on GNU/Linux distributions such as Red Hat, Fedora, SUSE Linux, and CentOS. The *RPM Specs Editor* is geared toward Qt applications containing various libraries and plugins but should be adaptable to other frameworks if needed.

Overview

The *RPM Specs Editor* utility consists of a main window with tabs for variables, expressions, libraries, dependency requirements, change log, and options.

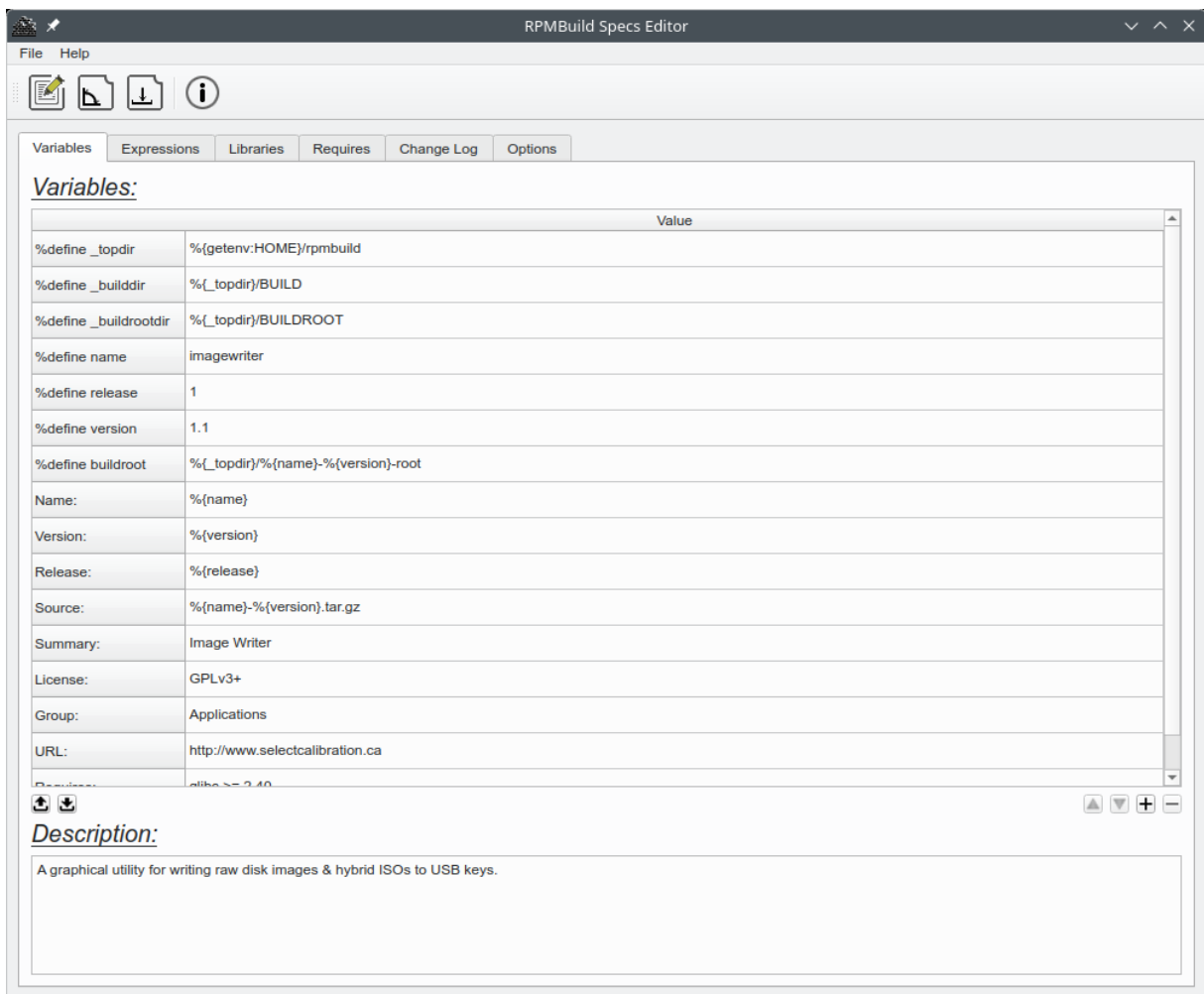


Illustration 1: RPM Specs utility main window.

RPM Specs Editor Users Guide

SPEC File

The SPEC file is used by the *RPMBuild* utility in order to create an RPM file. The SPEC file has uniquely identified sections that are used in the build process.

Table 1: RPM SPEC file partial list of sections:

Variable	Description
%description	Description of the program.
%prep	Steps necessary to prepare the software for building.
%build	Commands to build the software.
%install	Commands to install the software.
%post	Commands to be executed following the installation of the software.
%postun	Commands to be executed following the removal of the software.
%files	List of files that will be included in the package.
%changelog	Record of changes made for each release.

Variables

The SPEC file can contain variables that can be used for various purposes. For example, the variable `%{buildroot}` is defined as `%{_topdir}/%{name}-%{version}-root` and is easier and cleaner to write as opposed to the expanded version.

Examples of variables written at the beginning of a SPECs file:

```
%define _topdir                %{getenv:HOME}/MyProjects/Installers/linux/rpm
%define _builddir              %{_topdir}/BUILD
%define _buildrootdir          %{_topdir}/BUILDROOT
%define name                    measuredirect
%define release                 1
%define version                 9.0
%define buildroot               %{_topdir}/%{name}-%{version}-root
Name:                           %{name}
Version:                         %{version}
Release:                         %{release}
Source:                          %{name}-%{version}.tar.gz
Summary:                         Measure Direct
License:                         GPLv3+
Group:                           Applications
URL:                             http://www.selectcalibration.ca
Requires:                        glibc >= 2.31
Requires:                        libz1
Requires:                        fontconfig
Requires:                        libfreetype6
Requires:                        libglvnd
Requires:                        libXext6
Requires:                        libX11-6
Requires:                        libexpat1
Requires:                        libbz2-1
Requires:                        libpng16-16
Requires:                        libxcb1
Requires:                        libXau6
Requires:                        libdrm2
```

RPMSpecsEditor Users Guide

```

Requires:                libX11-xcb1
Requires:                libxcb-icccm4
Requires:                libxcb-image0
Requires:                libxcb-shm0
Requires:                libxcb-util1
Requires:                libxcb-keysyms1
Requires:                libxcb-randr0
Requires:                libxcb-render-util0
Requires:                libxcb-render0
Requires:                libxcb-shape0
Requires:                libxcb-sync1
Requires:                libxcb-xfixes0
Requires:                libxcb-xinerama0
Requires:                libxcb-xkb1
Requires:                libxcb-xinput0
Requires:                libSM6
Requires:                libICE6
Requires:                libxkbcommon-x11-0
Requires:                libxkbcommon0
Requires:                libuuid1
Requires:                libxcb-glx0

```

Variables are added to the SPEC file in the same order listed in the *RPMSpecsEditor* variables section. The order is important as variables must be defined before they can be used.

Some variables, such as `%{_topdir}`, have special meaning for the *RPMBuild* utility. Commonly used RPM macro variables are shown in the following tables.

Table 2: *RPMBuild* utility macro variables:

macro	Definition	Comment
<code>%{buildroot}</code>	<code>%{_buildrootdir}/%{name}-%{version}-%{release}.%{_arch}</code>	same as \$BUILDROOT
<code>%{_topdir}</code>	<code>%{getenv:HOME}/rpmbuild</code>	
<code>%{_builddir}</code>	<code>%{_topdir}/BUILD</code>	
<code>%{_buildrootdir}</code>	<code>%{_topdir}/BUILDROOT</code>	
<code>%{_rpmdir}</code>	<code>%{_topdir}/RPMS</code>	
<code>%{_sourcedir}</code>	<code>%{_topdir}/SOURCES</code>	
<code>%{_specdir}</code>	<code>%{_topdir}/SPECS</code>	
<code>%{_srcrpmdir}</code>	<code>%{_topdir}/SRPMS</code>	
<code>%{_buildrootdir}</code>	<code>%{_topdir}/BUILDROOT</code>	

Table 3: General macro variables:

macro	Definition	Comment
<code>%{_sysconfdir}</code>	<code>/etc</code>	
<code>%{_prefix}</code>	<code>/usr</code>	can be defined to <code>/app</code> for flatpak builds
<code>%{_exec_prefix}</code>	<code>%{_prefix}</code>	default: <code>/usr</code>
<code>%{_includedir}</code>	<code>%{_prefix}/include</code>	default: <code>/usr/include</code>
<code>%{_bindir}</code>	<code>%{_exec_prefix}/bin</code>	default: <code>/usr/bin</code>
<code>%{_libdir}</code>	<code>%{_exec_prefix}/%{_lib}</code>	default: <code>/usr/%{_lib}</code>
<code>%{_libexecdir}</code>	<code>%{_exec_prefix}/libexec</code>	default: <code>/usr/libexec</code>

RPM Specs Editor Users Guide

macro	Definition	Comment
%{_datadir}	%{_datarootdir}	default: /usr/share
%{_infodir}	%{_datarootdir}/info	default: /usr/share/info
%{_mandir}	%{_datarootdir}/man	default: /usr/share/man
%{_docdir}	%{_datadir}/doc	default: /usr/share/doc
%{_rundir}	/run	
%{_localstatedir}	/var	
%{_sharedstatedir}	/var/lib	
%{_lib}	lib64	lib on 32bit platforms

Requires Variable Entry

The *Requires* entries describe what is needed on the target system to run the application. The default list of variables include an entry for the *glibc* version which must be equal to or greater than some value so an RPM created on a system that has a current *glibc* version greater than that of the target system it will not run. The *Require* entries ensures that the minimum requirements for the application are met and will deal with this during installation instead of simply not working.

With respect to *glibc* the following command can be run to determine what is currently installed:





```
ron@linux-4o1p:~> ldd --version
ldd (GNU libc) 2.31
Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
Written by Roland McGrath and Ulrich Drepper.
```

Additional *Requires* variable entries can be automatically added to the specs file based on a scan of the selected plugin and libraries folder. This is done by default but can be disabled if necessary.

Variable Editor

Variables can be manipulated a variety of ways. In addition to adding or removing entries the order can be changed using the UP/DOWN buttons. Variable entries can also be archived and restored as base settings.

Table 4: Variable Editor Functions:

Icon	Description
	Store current variable entries and values.
	Restore the current variables and entries from previously saved values.
	Move the selected entry up.
	Move the selected entry down.

RPM Specs Editor Users Guide

	Add a new variable entry.
	Remove a selected variable entry.

Description

The description entry describes the purpose of the program and any other relevant information. The description can be quite lengthy if needed.

Expressions

The Expression section contains entries that are used in various sections of the SPEC file. For example, the entry *Build Root Path* has a default value of `%{buildroot}%{_libdir}/%{name}` and is used to define the location of the application image prior to installation.

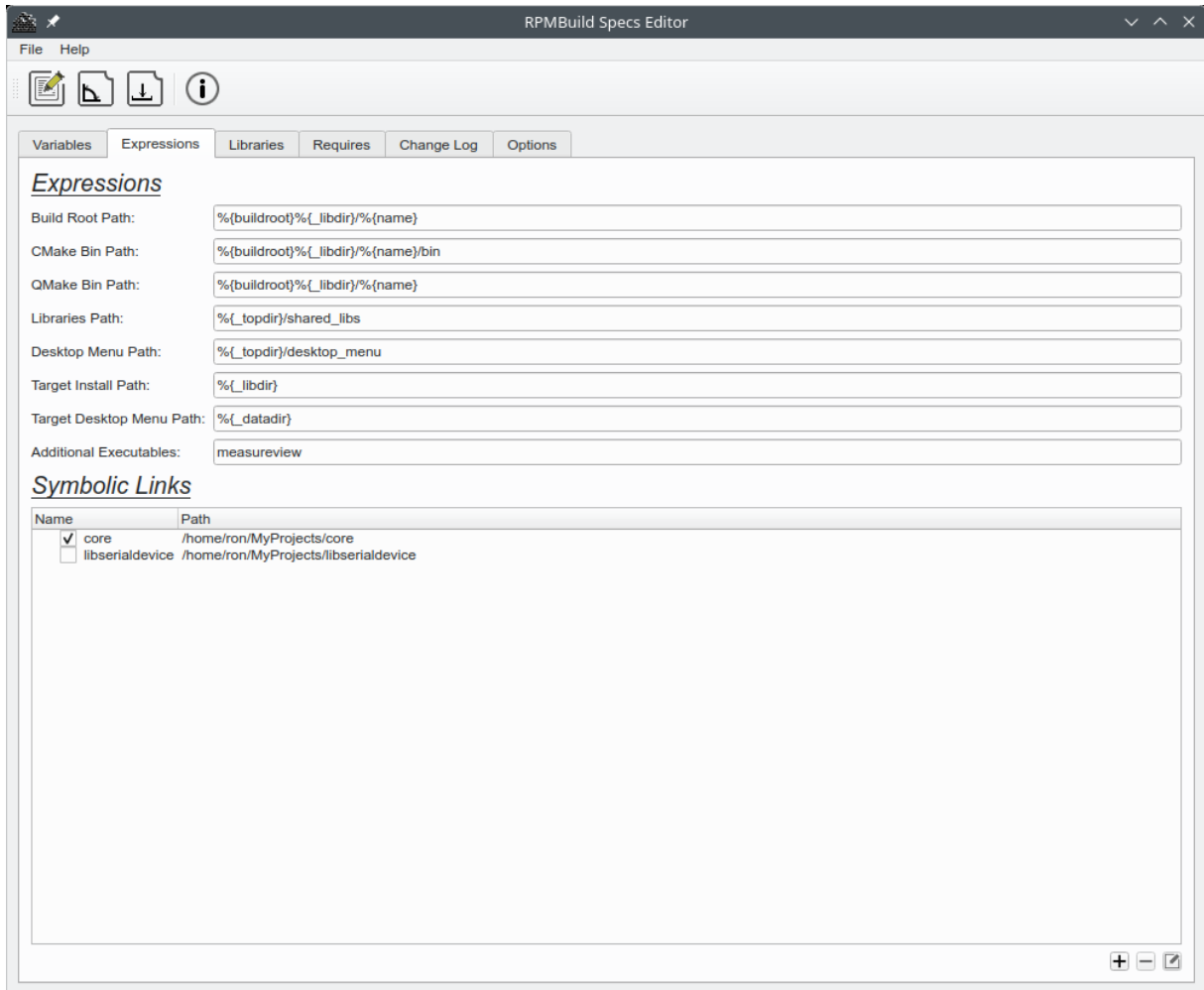


Illustration 2: Expressions view.

RPM Specs Editor Users Guide

The Expression entries have the following roles:

Table 5: Expression Entries:

Name	Description
Build Root Path	Location where the software image is created.
CMake Bin Path	Location of the compiled executable when using CMake.
QMake Bin Path	Location of the compiled executable when using QMake.
Libraries Path	Location of libraries and plugins used by the software.
Desktop Menu Path	Location of the files necessary for the desktop menu (*.desktop, *.png).
Target Install Path	Target location of the software when installed.
Target Desktop Menu Path	Target location of the menu entries when installed.
Additional Executables	List of additional executables. Some RPM packages involve more than one executable when created. The executable based on the variable package name is automatically included but other executables can be specified by entering one or more names in this section. The names are space separated and case sensitive.
Symbolic Links	Links to directories that are needed to be setup in the build location.

The following is an example of where the *Build Root Path* entry is used in the generated SPEC file:

```
...
%build

qmake PREFIX=%{buildroot}%{_libdir}/%{name}
make

%install

mkdir -p %{buildroot}%{_libdir}/%{name}
mkdir -p %{buildroot}%{_libdir}/%{name}/platforms
mkdir -p %{buildroot}%{_libdir}/%{name}/lib
mkdir -p %{buildroot}%{_libdir}/%{name}/xcbglintegrations
mkdir -p %{buildroot}%{_datadir}/applications
mkdir -p %{buildroot}%{_datadir}/pixmaps

make install

patchelf --set-rpath '$ORIGIN/lib' %{buildroot}%{_libdir}/%{name}/%{name}
...
```

Additional Executables

This entry is a single line containing one or more executable names separated with a space that the *RPMbuild* utility needs to handle. An example of where this would be necessary is when creating an RPM for the *MeasureDirect* utility. *MeasureDirect* creates two executables where one is handled automatically but the second must be manually defined (executable is called *measureview* as shown in illustration 2). The following is an example of the entries related to the additional executable in the SPEC file:

```
...
```

RPM Specs Editor Users Guide

```
make install

patchelf --set-rpath '%{_libdir}/%{name}/lib' %{buildroot}%{_libdir}/%{name}/%{name}
patchelf --set-rpath '%{_libdir}/%{name}/lib' %{buildroot}%{_libdir}/%{name}/measureview

...

%post

ln -sf %{_libdir}/%{name}/%{name} %{_bindir}/%{name}
ln -sf %{_libdir}/%{name}/%{name} %{_bindir}/measureview

...

%postun

rm -f %{_bindir}/%{name}
rm -f %{_bindir}/measureview
rm -r %{_libdir}/%{name}/lib
rm -r %{_libdir}/%{name}/platforms
rm -r %{_libdir}/%{name}/platformthemes
rm -r %{_libdir}/%{name}/xcbglintegrations
rm -r %{_libdir}/%{name}

%files

%defattr(-,root,root)
%{_libdir}/%{name}/%{name}
%{_libdir}/%{name}/measureview
%{_libdir}/%{name}/lib/*
%{_libdir}/%{name}/platforms/*
%{_libdir}/%{name}/platformthemes/*
%{_libdir}/%{name}/xcbglintegrations/*

...
```

Symbolic Links

This allows one or more symbolic links to be created that may be necessary for compilation when creating the SPECS file. Using the example from illustration 2 it is necessary to create a symbolic link to the folder 'core' and a library called 'libserialdevice' in order to compile the *MeasureDirect* example. These are automatically added and removed when no longer needed.

```
...

%build

ln -s /home/ron/MyProjects/core %{_builddir}/core
ln -s /home/ron/MyProjects/libserialdevice %{_builddir}/libserialdevice

...

make install

rm %{_builddir}/core
rm %{_builddir}/libserialdevice

...
```

Libraries

The Libraries section allows selection of folders and specific library files to be included in the RPM package. The folders can be used for Qt plugins or for dynamically linked library files.

RPM Specs Editor Users Guide

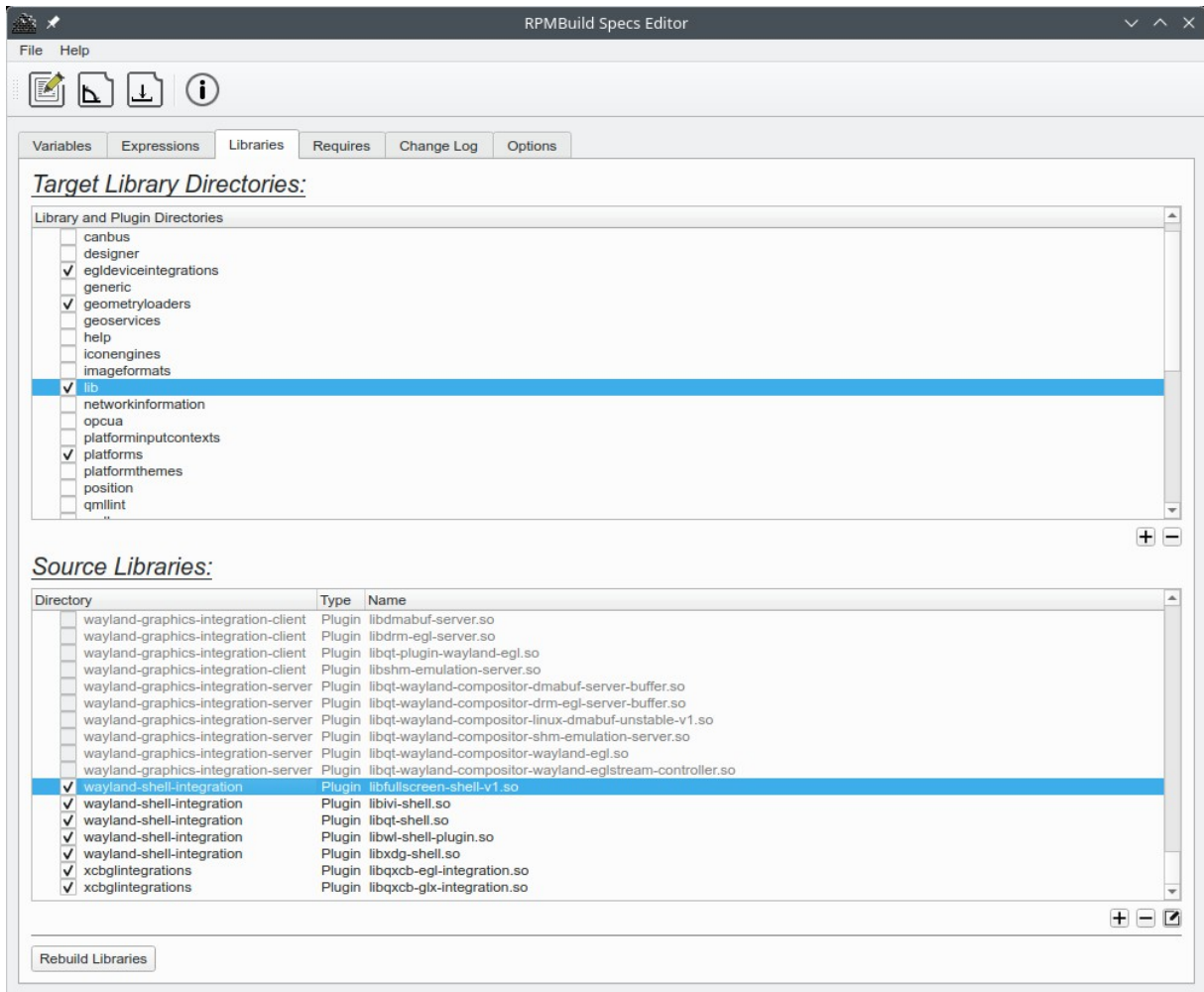


Illustration 3: Target and source libraries.

Target Library Directories

The target library directories represents the name of the directory that will be included in the RPM package. In the example shown in illustration 3 the visible target directory entries are *platforms*, *lib*, *egldeviceintegrations*, and *geometryloaders* and will be created as sub directories of the installed software.




Source Libraries

The source libraries list the individual files that will be copied to the associated target directory of the generated RPM file. For plugin files a version number is not part of the file name where dynamic libraries is expected to have the full version numbers as part of the name.

The Library entries can be manipulated with the following functions:

RPMSpecsEditor Users Guide

Table 6: Library Functions:

Icon	Description
	Add target directory or library.
	Remove target directory or library.
	Modify name of library. For multiple library selections the version numbers can be changed without having to modify each entry individually. See section Source Library Versions for details.

Source Library Versions

The source dynamic libraries are expected to have three version values in the format of *lib<some_name>.so.1.2.3*. In the event that the Qt version has changed all library files can be updated to a specific version by selecting all relevant library files and change the version numbers as a group to match what is needed.

Rebuild Libraries

This function will update the targets and target library entries based on actual file data. When there are numerous changes to target names and library file names it is easier to simply rebuild all the data than to try and change individual entries. When clicked the user needs to select the root directory containing all the *Target Libraries*.

All new entries are unselected by default when using the rebuild function.

Requires

The *Requires* section contains details of the packages that would need to be installed on the target system to provide the required libraries used by the application. This option scans the shared libraries folder, including plugin libraries, as opposed to just the executable which is done automatically when the rpm file is created.

Installing the RPM file directly should identify and automatically install all required libraries defined in the RPM file. The Debian versions created by SCI uses the Alien utility to convert the RPM to a DEB package which does not transfer information related to required libraries so manual installation of additional packages may be necessary.

RPM Specs Editor Users Guide

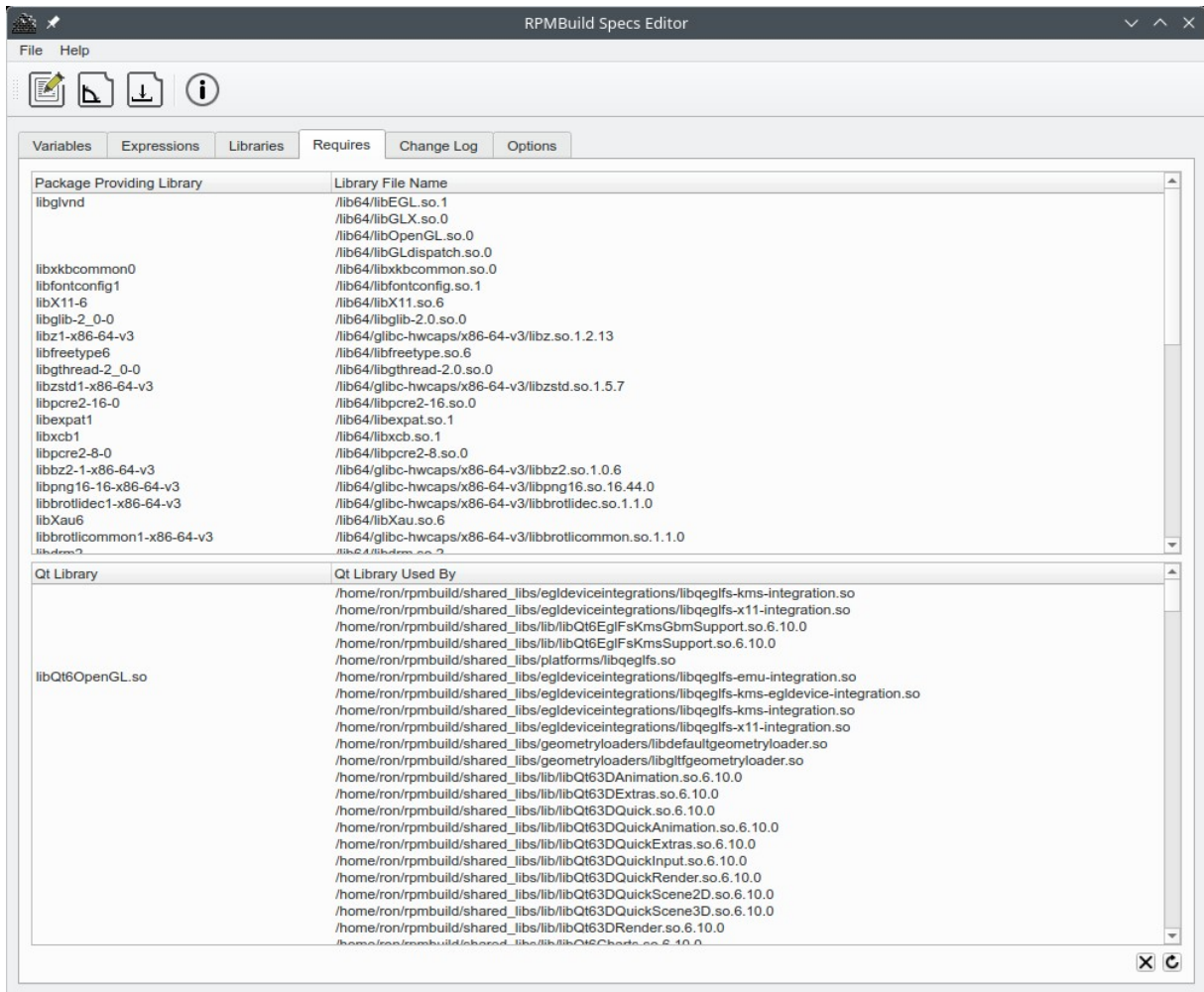




Illustration 4: Requires section.

Table 7: Requires Functions:

Icon	Description
	Update libraries and packages from the <i>shared_libs</i> path defined in <i>Options</i> section.
	Clear all listed libraries and packages.

The upper view of the Requires section shows a list of all packages that are needed to be installed along with the list of dynamic libraries they provide. As shown in illustration 4 the first entry is *libglvnd* which provides four library files needed by the application. A *Requires* entry in the spec file will be automatically created for *libglvnd* so that this package is installed if missing.

The lower view of the Requires section shows a list of all used Qt library files and which libraries require them. As shown in illustration 4 the Qt library file *libQt6OpenGL.so* is needed for the

RPMSpecsEditor Users Guide

listed library or plugin files.

The Qt library file as shown in the lower section of illustration 4 does not include the version numbers following the library extension (format of lib<name>.so.xx.yy.zz). This is defined in the library section.

Change Log

The *Change Log* section contains details of the revision history for the RPM utility. Each entry consist of two parts where the first is the date and author name followed by one or more change entries. The lines must start with either '*' or '-' to identify a new entry or change. The order the entries must be newest revisions at the top and oldest revisions at the bottom. The following shows an example of a change log with the newest entry on top:

```
* Wed Sep 3 2025 <author name>
- Fixed various bugs
- Added feature to do something better.
- Cleaned up something.
* Sun Nov 6 2016 <author name>
- Initial RPM release
```

RPM Specs Editor Users Guide

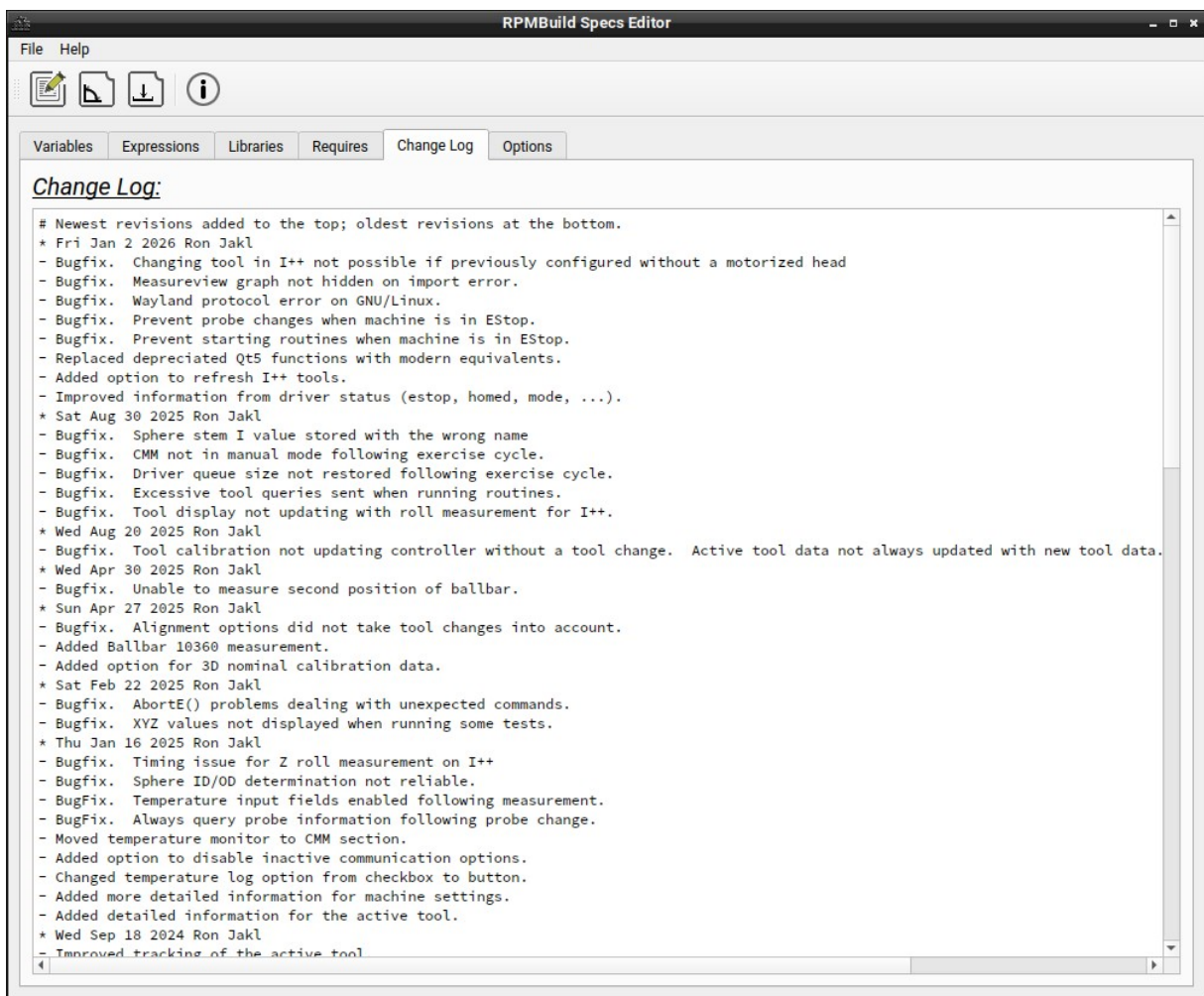


Illustration 5: Change Log section.

Options

The *Options* section contains entries that will impact the generated SPEC file.

RPM Specs Editor Users Guide

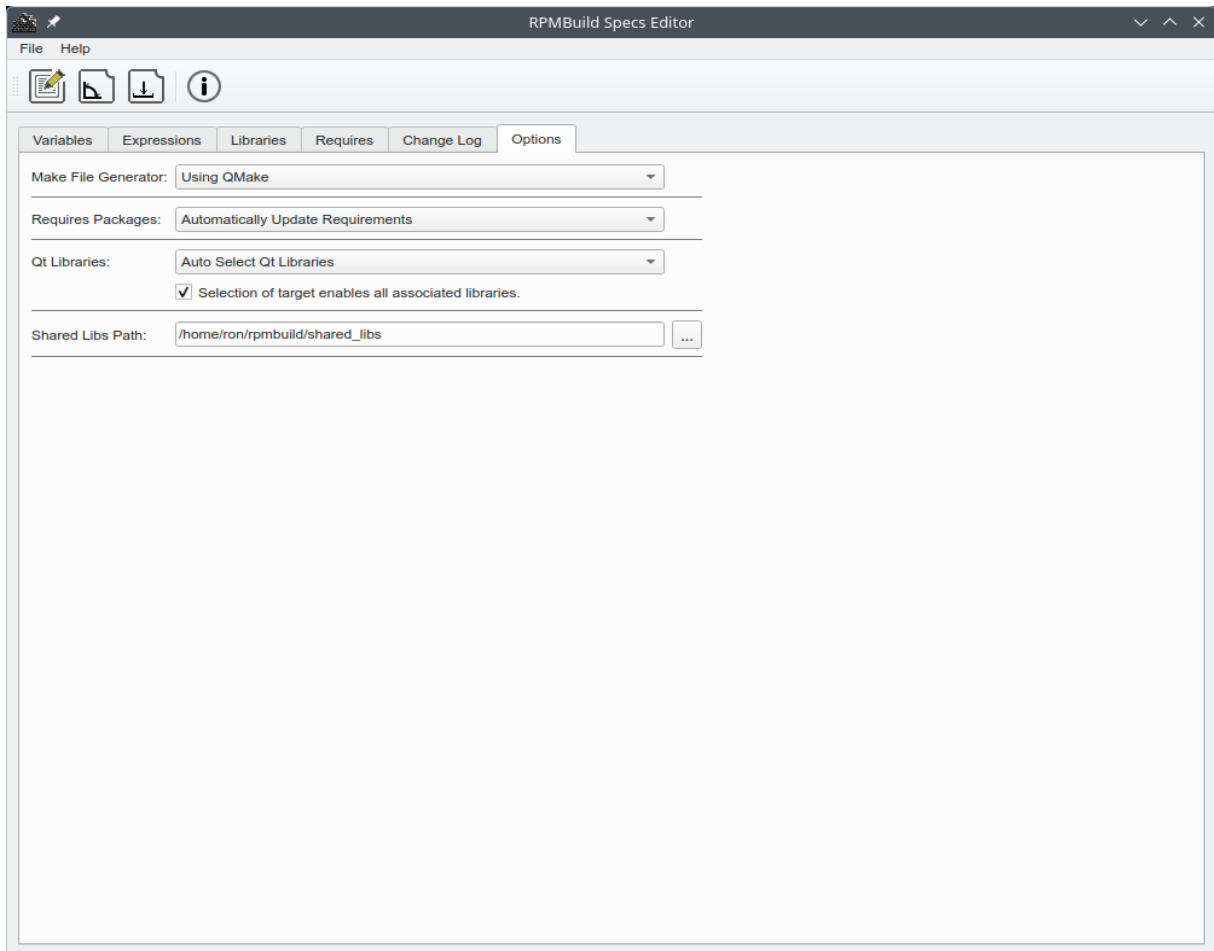


Illustration 6: Options section.

Table 8: Options:

Name	Description
Make File Generator	Selection for the type of make-file generator used in the spec file.
Requires Packages	Method for dealing with required packages used by the libraries and plugins.
Qt Libraries	Auto or manual selection of Qt library files. By default all necessary libraries must be manually selected to be included with the application but this can be done automatically. Selection behavior for target libraries. Libraries associated with a specific target can be automatically enabled when selected. For example, if there is a dozen library files associated with a specific target they will all be enabled when the target is enabled.
Shared Lib Path	Location where the shared libraries and plugins are located.

RPM Specs Editor Users Guide

RPM Creation Example

This example describes the entire process to create an RPM file from a Qt application using Qmake to create the make file. At a minimum it is assumed that a version of Qt is installed and utilities such as *GCC*, *make*, *patchelf*, and *RPMBuild* are also installed.

RPMBuild Directory Setup

A suitable directory structure must be created in order to use the *RPMBuild* utility. In this example the build folder is called *rpmbuild* and is below the users home folder.

```
rpmbuild
├── BUILD
│   └── SOURCES
├── RPMS
│   └── x86_64           ← location where created binary RPM will be put
├── SOURCES             ← location where source tar archives are stored
├── SPECS               ← location where specs files are stored
├── SRPMS
├── desktop_menu       ← location where the desktop menu file and image is stored
├── shared_libs        ← location of plugins and libraries
│   ├── bearer
│   ├── egldeviceintegrations
│   ├── generic
│   ├── imageformats
│   ├── lib
│   ├── platforminputcontexts
│   ├── platforms
│   └── platformthemes
```

The folders *desktop_menu* and *shared_libs* contain the menu data and copies of libraries and plugins that can be used to create the RPM file.

Source Files

The source file is a tar archive of the application and associated files. For Qt this will include the *PRO* file or *CmakeLists.txt* file, source files, resource files, and anything else related to the application. The naming of the archive (internal and external) is important. For example, an application with the name *MyApp* version 1.0 should be in a directory called *MyApp-1.0* and then compressed with the tar command and saved in the *SOURCES* folder with the same naming convention.

```
tar -czf MyApp-1.0.tar.gz MyApp-1.0
```

SPEC File

The SPEC file is created by the *PRMSpecsEditor* utility and saved in the *SPECS* folder. The variable names for program and version numbers must be updated to match the name and version of the application.

Desktop Menu Items

The desktop menu entries describe the application in some detail and will be added to the operating system launcher when installed. There are two files where the first is a text file with the extension of *.desktop* and the second is a PNG image file for the application icon. Both will have the name of the application (i.e. *MyApp.desktop* and *MyApp.png*).

Typical Desktop Entry:

RPMsSpecsEditor Users Guide

```
[Desktop Entry]
Categories=Utility;
Encoding=UTF-8
Name=Description Of MyApp
GenericName=My Application
Comment=My Application
Exec=MyApp
Icon=MyApp.png
Terminal=false
StartupNotify=true
Type=Application
```

Build RPM

The build process is done from the root of the *RPMBuild* directory. If using the terminal the command would be something like this:

```
rpmbuild -v -bb --clean SPECS/MyApp.spec
```

where:

```
-v          = verbose output
-bb        = build only binary packages
--clean    = remove build tree after packages are made
SPECS/xxx  = name of specs file
```

If source file RPM'S are needed then use the appropriate command line option. In this example only an executable is required.

If creation is successful there will be an RPM file created in the RPMS folder with the name MyApp-1.0-1.x86_64.rpm (assuming the target is x86_64).

RPM Signing

The RPM files can be signed to show that the application was created by a credible source and that modifications have not been made following the creation of the RPM. The command to do this is:

```
rpm --resign RPMS/x86_64/MyApp-1.0-1.x86_64.rpm
```

This will only work if a suitable signing certificate exists. If the goal is for large scale distribution than a certificate should be acquired allowing this software to be installed anywhere. If the RPM is unsigned or signed with a locally generated certificate then the user installing the RPM will have the option to abort installation due to not being signed or signed with an unknown certificate.

Troubleshooting

Following the installation of an application on a target system the application may not launch properly if dependencies are missing and this happens particularly with the plugins.

Dependency problems should only be an issue when installing on Debian based systems as the Debian package is created from the RPM package using the Alien utility and appears to ignore the 'Requires' entries from the RPM package. The solution is to create a similar utility for Debian systems that creates a Debian package directly (a future project idea).

Example Application Launch Problem

The *ballbarviewer* application was installed on a Debian based GNU/Linux system and did not launch properly from the application menu following the installation. When opened from a

RPM Specs Editor Users Guide

terminal window the following was returned:

```
ron@ron-VirtualBox:~$ ballbarviewer ← launch application from terminal window.

qt.qpa.plugin: Could not load the Qt platform plugin "xcb" in "" even though it was found.
This application failed to start because no Qt platform plugin could be initialized. Reinstalling the
application may fix this problem.

Available platform plugins are: eglfs, linuxfb, minimal, minimalegl, offscreen, vnc, xcb.

Aborted (core dumped)
```

To see detailed information related to plugins it is necessary to set the environment variable to show debug information for Qt plugins. This is done from the terminal window prior to trying to launch the application.

```
export QT_DEBUG_PLUGINS=1
```

Finding Missing Libraries

The *ballbarviewer* application is typically installed in the folder */usr/lib64/ballbarviewer* and all libraries and plugins will be below this directory. Since the problem was found with the *xcb platform* plugin this specific library is the one of interest.

```
ron@ron-VirtualBox:~$ ls /usr/lib64/ballbarviewer/platforms
libqeglfs.so libqlinuxfb.so libqminimalegl.so libqminimal.so libqoffscreen.so libqvnc.so
libqxcb.so
```

The missing libraries needed by *libqxcb.so* can be found using the GNU/Linux LDD command. The output was piped through *grep* looking only for lines containing the text '*not found*'.

```
ron@ron-VirtualBox:~$ ldd /usr/lib64/ballbarviewer/platforms/libqxcb.so | grep "not found"
libxcb-xinerama.so.0 => not found
libxcb-xinerama.so.0 => not found
```

Installing Missing Libraries

Once the missing libraries have been identified they can be installed. The name of the library is usually not the name of the package containing it but this can be determined by a variety of methods. In this example the package name is '*libxcb-xinerama0*'.

```
ron@ron-VirtualBox:~$ sudo apt-get install libxcb-xinerama0
[sudo] password for ron:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  libxcb-xinerama0
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 0 B/5,260 B of archives.
After this operation, 37.9 kB of additional disk space will be used.
Selecting previously unselected package libxcb-xinerama0:amd64.
(Reading database ... 184296 files and directories currently installed.)
Preparing to unpack .../libxcb-xinerama0_1.14-2_amd64.deb ...
Unpacking libxcb-xinerama0:amd64 (1.14-2) ...
Setting up libxcb-xinerama0:amd64 (1.14-2) ...
Processing triggers for libc-bin (2.31-0ubuntu9.18) ...
```

```
ron@ron-VirtualBox:~$ ballbarviewer ← worked this time
```

RPMSpecsEditor Users Guide

Revision History

<i>Date</i>	<i>Version</i>	<i>Changes</i>
Sep 4, 2025	1.0	New Program
Oct 2, 2025	1.1	[bug fix] Writing specs file no longer appears to hang the save dialog. Added additional information for required Qt libraries. Added warning if required Qt libraries not included. Added completion message when writing specs file.
Jan 7 2026	1.2	[bug fix] Open specs data dialog caption name is 'save'. [bug fix] Adding or removing variables not updating variables data. Added default entries for <code>_builddir</code> and <code>_buildrootdir</code> variables. Added option for multiple executables. Removed plugin library hint from libraries view. Added symbolic links option. Added warning messages when removing various entries. Updated documentation.
Feb 25, 2026	1.3	[bug fix] Direct changes to shared library path not seen. Added option to rebuild the library data. Added more detailed information about required Qt libraries. Added option to automatically select required Qt libraries.
Feb 27, 2026	1.4	Added option to automatically select libraries under a specific target. Added progress dialog when creating a specs file.