Revisor 2.1

Reference

Revisor Complete Installation, Configuration and Tweaking Reference

Publican

BOOK PUBLISHING TOOL

Jeroen van Meeuwen, RHCE

Revisor 2.1 Reference
Revisor Complete Installation, Configuration and Tweaking Reference
Edition 0

Author Jeroen van Meeuwen, RHCE *kanarip@fedoraunity.org*

This is Revisors upstream documentation.

Preface

This is the documentation for Revisor, a utility to create and customize your own Linux distribution based on Fedora, Red Hat Enterprise Linux or CentOS. This is also known as a Remix, or Re-Spin, or Re-Master. You may create anything you want with Revisor no matter what it's called though.

1. About the Contributors

Author

Jeroen van Meeuwen (RHCE, LPIC-2, MCP, CCNA) is currently a Senior System Engineer, specialized in Linux systems and Systems Architecture, working for Operator Groep Delft in The Netherlands. His experience with computers goes back to the early '90s, with a Philips P2000T being over a decade old, little tapes containing programs but most importantly games, and 16K memory cartridges. Since 1998, he has been involved with Red Hat Linux (5.2 at that time), and was an early adopter of Fedora Core Linux in November 2003, until his first real contributions to Free and Open Source Software were made in 2005.

As a contributor to Free and Open Source Software within the Fedora community, amongst other programs, Jeroen has developed Revisor, a Python framework to build distributions with. With regards to Configuration Management, Jeroen currently maintains or co-maintains -amongst other packages- the entire stack of packages related to Puppet

Contributors

Jonathan Steffan is a community volunteer based in Colorado, USA, and has a long standing record within Fedora for packaging Zope (Web Application Server), Plone (Open Source Content Management System), providing compat-python2.4 packages through Livna[1] and RPMFusion[2] ever since Fedora 7, and voluntarily administering the Fedora Unity servers, Zope and Plone instances, creating and further developing Revisor and pyJigdo, amongst many other things.

2. About Fedora Unity

The Fedora Unity Project consists of a group of concerned peers from within the Fedora community that strive to bring the best possible solutions to the community, in a consistent manner. This, amongst other things, resulted in extensive documentation on various topics often referred to on the Web, published under the Open Documentation License v1.0.

The Fedora Unity Project is a different project then the official Fedora Project. The people behind the Fedora Unity Project often contribute to the Fedora Project directly as well, but there's little to no bureaucracy in the Fedora Unity Project. Why do you think these Re-Spins are not released by the Fedora Project itself?

3. About this Document

This document is licensed under the Open Publication License version 1.0, which is available at *http://www.opencontent.org/openpub/*. You can get the latest version from *http://kanarip.fedorapeople.org/*

[1] Livna (*http://livna.org*) merged into RPMFusion after having offered numerous free and non-free packages through a third-party repository
[2] RPMFusion (*http://rpmfusion.org*) is the best & largest third-party addon repository to Fedora and Enterprise Linux, with free and nonfree packages

Revisor_Reference_Manual/en-US/pdf/Revisor_Reference_Manual.pdf (PDF), and it's sources live at *http://git.fedorahosted.org/git/?p=revisor;a=tree;f=doc*.

4. 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*[3] 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.

4.1. 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 keycaps 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 keycap, all presented in mono-spaced bold and all distinguishable thanks to context.

Key combinations can be distinguished from keycaps by the hyphen connecting each part of a key combination. For example:

> Press **`Enter`** to execute the command.
>
> Press **`Ctrl`**+**`Alt`**+**`F2`** to switch to the first virtual terminal. Press **`Ctrl`**+**`Alt`**+**`F1`** to return to your X-Windows session.

The first paragraph highlights the particular keycap to press. The second highlights two key combinations (each a set of three keycaps with each set 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

[3] https://fedorahosted.org/liberation-fonts/

> **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.

4.2. 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"));
   }
}
```

4.3. 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.

5. We Need Feedback!

You should over ride this by creating your own local Feedback.xml file.

Introduction

Revisor is a community product by Fedora Unity. Amongst other features, it allows the creation of installation media and live media in the easiest possible manner, through a click-and-go GUI. This chapter gives some insight on how and why Revisor was born, and how the product evolved since.

1.1. History of Revisor

Revisor development started in December 2006, during the Fedora 7 development cycle, in which -as you might recall- the Fedora Core repository, maintained by Red Hat, and Fedora Extras repository, mostly maintained by the community, were merged into one large repository being maintained by both community members as well as Red Hat employees -which are mostly community members who just so happen to be hired by Red Hat, so community altogether. Before then Red Hat employees maintained Fedora Core -as the set of packages upstream for Red Hat's Enterprise product- and the community maintained a repository with additional software; Fedora Extras. Red Hat composed the Fedora distribution every once in a while, but the merge introduced the possibility for packages that were in Fedora Extras to be included in the main distribution, and for the community to also (co-)maintain the (former) Fedora Core packages that originally made up the distribution.

In addition to this huge merge of packages, Red Hat employees were also able to release the entire build process to the community, meaning that from the moment the source is committed up and until the release is announced, the entire process is open. Not that is was all behind closed doors or proprietary or anything, the community just couldn't really play with it as much. We now have koji (build system), mash (repository compose from build system products), bodhi (updates release system), livecd-tools (compose tool for live media) and pungi (compose tool for installation media).

Composing the distribution's media

Composing media was an obscure process up and until the moment these tools exposed the best way to compose (a set of) installation media. Fedora Unity had been building and releasing so-called Re-Spins1 regularly, but they were built using a not-so-very intelligent bash script. Like hundreds if not thousands of other parties that needed to build their own media one way or the other, the entire process was based on the best educated guess of what should happen. Luckily, in the FOSS world an educated guess is often a very good guess, despite the fact that one keeps learning even years after the original engagement.

When in December 2006 the compose tools hit a stage in which they were released to the public, Fedora Unity was eager to get these tools and study them and use them for composing their Re-Spins. Up and until then, Re-Spins were composed with the aforementioned bash script that didn't do much but trigger the appropriate commands in a sequence; There wasn't any dependency resolving between the packages included nor did we know exactly how a release was supposed to be composed -it was our educated guess of how it could happen. Although it often led to success, we've had many, many failed Re-Spins as well. With a handful of volunteers, you can imagine the amount of frustration that might give. Fedora Unity was eager to improve their Re-Spin process.

Fedora Unity's engagement

So, early February 2007, a number of Fedora Unity members attended “FUDCon 2007” in Boston, and presented a working GUI front-end to both livecd-tools and pungi enabling regular users to also re-compose or re-spin the installation media and live media they had been getting from the Fedora Project. Revisor at this point just made it “as easy as possible”. Besides the possibilities of pungi and livecd-tools themselves, the wizard Revisor had apparently was very, very useful to mere mortals. From that point on, things took off.

Fedora Unity decided Revisor could accomplish more then just being a front-end to existing compose tools and enable someone to tweak a lot of settings as well. In March 2007, Revisor was rebuilt from the ground up in order to allow a more flexible process, neing more dependent on the configuration directives it was given from configuration files, command-line parameters and the graphical user interface, and less so on the processes of the existing tools. When in San Diego at the Red Hat Summit (early May 2007), Robert 'Bob' Jensen and Jonathan Steffan gave a presentation on “Customizing Fedora”, the responses were amazing. Since then Revisor has evolved from a front-end to existing tools to the complete compose tool it is today, with lots of configuration options for specific use-cases.

For users, Revisor is particularly useful because it has a GUI front-end wizard, which, with the defaults settings, will just succeed in getting a user the media he/she wants. If a user decides he needs little adjustment of the media, the GUI allows for selecting the most common options. If a user decides he needs some less common adjustments, the configuration options gives him very granular control - and as long as the documentation on all the options is sufficient, users will be able to make those less common adjustments.

For administrators on the other hand, Revisor is the tool that gives so much granular control over what happens, that it can serve almost every specific use-case. In this aspect, Revisor could potentially replace the compose tools administrators have been developing themselves with a consistent and flexible program flow.

This document should enable you to study the process of composing installation and live media, and comprehend the logic Revisor adds to that process.

1.2. The Installation Media Challenge

When Fedora Unity first started doing these so-called Re-Spins, the challenge ahead could maybe be explained like this:

When a user downloads a Fedora release and installs the distribution, the user will need to download and install a number of updates. The “older” the release becomes, the more updates will be available, and the greater the total download size of these updates the user will need to download on top of the download size of the original release media.

“older” is in quotes on purpose, because really for an operating system -or “distribution” if you will- being released every 6 months, “old” is quite a relative concept. The number of updates available however, at any given time during the release cycle, may range from 0 right after the release (which has never happened before), to the total amount of packages installed on the user's system (often over 2000). You can imagine the size of these updates ranging from 0 MB to an astonishing 2GB(!), only 6 months after the initial release. A single Fedora release has a support cycle of 13 months.

Some of us do not have the bandwidth capacity or enough data transfer quota to download this many extra, rather useless bits. In addition, some of us do not have an Internet connection at all, and those benefit from getting the updates through Re-Spins as well.

The use of updates in Re-Spins has several more beneficial side-effects, which we'll explain in more detail later on in this document. Long story short; If for some reason the software used to compose the media (the release) with does not work for your hardware or your specific needs, updated software incorporated in the composed installer images might resolve that problem.

This is the original challenge the Fedora Unity team resolved a long time ago, and is at the base of what Revisor does nowadays.

1.3. The Live Media Challenge

Back in the day Fedora Core 5 was the most recent release, Fedora Unity created so-called live media using Kadischi, then actively developed by Jasper Hartline, another valued Fedora Unity member. In those days, live media could only have a read-only root file system and was not as feature-rich as live media is today. However, just before Revisor came to life, two applications were developed; pungi for creating installation media, and livecd-tools for creating live media. These two applications did their work well; The media composed for a genuine Fedora release, including many different custom live media spins were, and still are, created with these tools, not to mention the many downstream consumers that use these tools to do the exact same thing. Immediately, the Revisor developers set themselves a target to provide a single interface to both of those tools, and give the user more flexibility so that the user could bend the rules without breaking them.

Features

Revisor allows you to build and customize your own Remix, Re-Spin, Spin or even your own distribution, based on Fedora and derivative distributions such as Red Hat Enterprise Linux and CentOS.

Revisor builds installation media, live media, installation trees, cobbler distro's and profiles, virtualization images and more. We'll now briefly explain what each of these terms mean and what Revisor can (or cannot) do for you.

2.1. Installation Media

Installation media is what you normally use to install a system with. The installation media composed will allow you to go through the installation process, answering a number of questions (either manually or through kickstart), and ends up in a system running the distribution you install.

Composing installation media using the Revisor GUI allows you to choose the media (CD, or DVD), the packages to be included on the media (also called *RPM payload*).

Using the command-line interface, Revisor also allows you to choose DVD Duallayer and single- or dual-layer Bluray.

2.2. Installation Trees

Installation trees are typically used in environments where a distribution needs to be deployed over multiple systems, or is very volatile. Installation trees are often made accessible through HTTP or FTP protocols, in one place, and do not have as much overhead (in creating .iso files, and burning those to optical media to distribute them).

2.3. Live Media

Live media often is a perfect showcase for an Operating system, Desktop Environment or any other thing you want to show. Also, since Live media is read-only, live media perfectly allows for a kiosk system, a system that may change while it's running, but restores all original settings when rebooted.

Live media is also installable. You start out with a system and boot it from live media, then choose to install the live media. This however is inferior to real installation media, but is convenient if you happen to like what you see when running from live media.

2.4. Reproducibility

Media composed with Revisor is extremely reproducible. Using **`kickstart_exact_nevra`**, you can even select specific versions of packages to be included on the product.

2.5. Consistency

When composing different types of media, such as CDs and DVDs, Revisor composes these discs in one run, making the different media completely consistent. **pungi** would require you to run twice, once for CDs, and once for DVDs. This is because **pungi** uses the **`part / <size>`** kickstart configuration directive to set the maximum size of the media, and has no option to override the size on the command-line, nor to compose a certain set of media (it all depends on the size).

2.6. Flexibility

Over the years, Revisor has been adopted to serve a large number of use-cases, where use-cases stretch from media being composed as efficient as possible, as robust as possible, specific deployment needs and expectations, and to match the Fedora Project Release Engineering tools' behaviour. All this allows you to configure a lot, and thus customize a lot, making Revisor more of a flexible framework.

2.7. Graphical User Interface

Revisor has a Graphical User Interface or GUI, in addition to the Command Line Interface or CLI, which makes Revisor more accessible to users then the other tools, which are CLI only. Most people only know of Revisor through the GUI, and may think there is no CLI to Revisor. Only when it comes down to many of the additional features that Revisor has, and that do not fit in a simplified GUI, one gets down with it using the CLI.

2.8. Open Development Community

Revisor has one of those old-fashioned Free and Open Source Software development communities, allowing anyone to make a contribution to Revisor. In fact, Revisor has not bounced a single patch since the project started. Therefor, it improves faster then any of the other compose tools, and is better adaptible to your needs and expectations, because unlike the other utilities, Revisor is not limited to use-cases that apply to Fedora Project Release Engineering.

2.9. Plugin System

Revisor has a plugin system so that you can easily extend Revisor. This plugin system gives you full control over the Revisor procedures, and hands you off anything Revisor knows about the compose process. There's are multiple plugins available from upstream as well. To give you an example, the ability to replace **`isolinux.cfg`** after the compose is done, is a plugin. See *Chapter 9, Plugins* for more information.

Current plugins included with Revisor include:

- *Section 9.1.1, “Cobbler Plugin”*
- *Section 9.1.6, “Isolinux Plugin”*
- *Section 9.1.9, “Rebrand Plugin”*
- *Section 9.1.10, “Reuse Installer Images Plugin”*

2.10. Extraneous Debugging

Revisor has extraneous debugging, which enables you, as well as the supporters and Revisor's developers, to trace down what happens exactly, and where anything might have gone wrong.

2.11. Smart Caching

Revisor is one of the utilities that uses the same cache over and over again, across multiple product composes. If you have downloaded something once, chances are you're going to need to download it again and again especially if you are a frequent user of the compose tools. Revisor caches all the downloaded packages in **`/var/tmp/revisor-yumcache/`** (by default), enabling you to download just once and never again.

2.12. Using YUM Configuration Files

Revisor uses YUM configuration files, where everyone else is not. With using YUM configuration files however, the control you have is nearly limitless. With all the features in YUM already, using it's configuration file format and letting YUM itself work with those allows Revisor to do a lot of cool things without doing anything itself:

1. **Excluding packages from repositories**

   Excluding packages from repositories means a great deal. Not having them exist in the *Package Sack* ensures the package will not end up in the product. This may be what you want for maybe just a few, or maybe an awful lot of packages.

   Using the alternative configuration file format, kickstart, in use by every other compose tool, and the **`repo`** configuration directive that is available with kickstart, you can exclude packages using the **`--exclude=`** parameter to the **`repo`** configuration directive. However, that parameter does not allow wildcard matches.

2. **Including only a certain (set of) package(s)**

   Including only a certain package, or certain set of packages is valuable when a lot of packages exist in the repository configured, but you only need one or two.

3. **Concurrent use of baseurl(s) and the mirrorlist**

   Like during normal YUM operations, the baseurl(s) and the mirrorlist configured for a repository are used concurrently. This is not possible with the kickstart configuration directive **`repo`**, which takes either **`--baseurl`** or **`--mirrorlist`**, but not both.

4. **Repository priorities**

   Settings available with YUM are available within Revisor as well, like repository priorities. Using repository priorities, you can have YUM decide to pull a package from the repository with a higher priority (a lower priority number) rather then a repository with a lower priority.

5. **YUM Plugins**

   YUM plugins, such as **yum-fastestmirror**, **yum-fedorakmod**, are available, giving you even more control over the behaviour of YUM.

Part I. Getting Started

Installation

This chapter contains the installation instructions for Revisor.

3.1. Packages

You can install Revisor using RPM packages from the repositories already configured on your system.

revisor

Shorthand package for the Revisor GUI.

revisor-cli

The CLI version of Revisor. This package is always installed, as it contains the Python code for Revisor's core. Installing just this package will give you the command-line version of Revisor, and prevents the graphical dependencies from the ***revisor-gui*** package to be installed as well.

revisor-gui

The GUI version of Revisor. This is the actual package containing the Graphical User Interface, as opposed to ***revisor***. Depends on ***revisor-cli***, and thus also installs the command-line version of Revisor.

3.1.1. Red Hat Enterprise Linux 5 or higher

On Red Hat Enterprise Linux 5 or higher, and derivatives, install the Extra Packages for Enterprise Linux[1] (EPEL) repository.

Then, give the following command:

```
# yum install revisor
```

3.1.2. Fedora 7 or higher

On Fedora 7 or higher, and derivatives, no additional repository configuration is required.

Give the following command:

```
# yum install revisor
```

About EOL Releases

Please bear in mind that Fedora releases that are past the point of End-Of-Life, approximatly 13 months after the initial release, are not supported anymore for use with Revisor. Also, the version of Revisor running on these EOL versions of Fedora are not supported anymore.

[1] Extra Packages for Enterprise Linux: *http://fedoraproject.org/wiki/EPEL*

3.2. The Latest and Greatest

The latest and greatest is available from GIT, at *git://git.fedorahosted.org/revisor*. To clone this repository, use:

```
$ git clone git://git.fedorahosted.org/revisor/
```

Using the GIT clone, you have the several options to start using the latest and greatest:

Running directly from the source

You can run directly from within the source tree. See *Section 14.1, “Running Revisor from Source”* for more information on how to do so.

Installed packages and running from source

Do not run Revisor from source while RPM packages have been installed. Files managed by a package will get created, moved and removed when using Revisor's source tree, and updates to the installed RPM packages will destroy these changes.

Building your own packages

You can create your own packages, so that you have all the benefits of RPM. See *Section 14.2, “Building Revisor Packages”* for more information on how to do so.

Configuration

Revisor configuration can be performed using *Section 4.1, “Configuration Files”*, or through *Section 4.5, “Command-line Options”*.

We seriously recommend you consider configuration files for products you are going to have to build more then once, or for seriously complex products that make use of Revisor functions beyond what is a simple click-and-go exercise, such as rebranding and other advanced customization. On the other hand, if you are just to try one or two simple things just for this once, command-line options might be more suitable.

4.1. Configuration Files

Revisor uses configuration files for a large part of it's operations. These files mostly reside in **`/etc/revisor/`**. There is two types of configuration files Revisor uses:

1. **Revisor Configuration Files**

   Revisor configuration files, such as **`/etc/revisor/revisor.conf`**, contain information and settings unique to Revisor. A Revisor configuration file is where you specify default options, and include information on different products you want to compose.

2. **YUM Configuration Files**

   YUM configuration files, such as the files in **`/etc/revisor/conf.d/`**, contain configuration for YUM. To be more precise, Revisor doesn't even handle the files (it let's YUM do so). The files in **`/etc/revisor/conf.d/`** practically contain the same information as **`/etc/yum.conf`** combined with the files in **`/etc/yum.repos.d/`** (but not exactly the same content!).

4.1.1. `/etc/revisor/revisor.conf`

The default Revisor configuration file is **`/etc/revisor/revisor.conf`**. This configuration file contains two sections:

1. **`[revisor]`**

   The global section. Options specified in this section apply to all the models defined in this configuration file.

   See also: *Section 4.2, “Global and Model Configuration”*

2. **`[`*`model`*`]`**

   Model configuration. You use one section per *Model*, and each section represents a *product*.

   See also: *Section 4.2, “Global and Model Configuration”*

Model sections basically define a single product. Amongst other things, the distribution name, release version, architecture for the product to be composed and what YUM configuration file to use, are (often) defined on a per-model basis. There is a large number of settings available for models, and they are all related to how the product is going to look like. The product name, the location of the RPM payload for installation media, the ISO label, the YUM configuration file to use, are all model settings.

Using models, you can reproduce the outcome of the compose process, a *product*, simply by not changing the model configuration anymore. If you want something different, you can just add another model section, and name it differently.

To see what models are available with the Revisor standard package, use:

```
# revisor --list-models
```

4.1.2. `/etc/revisor/conf.d/`

The default YUM configuration files used by Revisor. In a model configuration section, the **`main =`** setting points to one of the YUM configuration files in **`/etc/revisor/conf.d/`**.

The YUM configuration files configure the repositories available during a product compose, as well as the available architectures.

4.1.3. Updates to Configuration Files

The Revisor packages (or any packages in Fedora for that matter) are not allowed to overwrite existing configuration files in **`/etc/`**, and they should thus not do so. Users anticipate that if they change a file in **`/etc/`** these changes are persistent in that they are not destroyed when a package is updated. If an update to Revisor is installed on your system, files with the extension **`.rpmnew`** may be created --if you had changed anything in the file before applying the update. Since this world isn't perfect, configuration errors may exist in the configuration files shipped with Revisor. Please pay close attention to updates to these configuration files by searching for and examining the **`.rpmnew`** files.

You can use any file location (not just **`/etc/revisor/`**) for your own custom configuration.

4.1.4. Changing Configuration Files

If you are creating your own models off of the ones that ship with Revisor itself, please consider using an alternative configuration file (a file other then **`/etc/revisor/revisor.conf`**, or copy the original file for safekeeping. This way, you can always return to a working, sample configuration file and test whether it is Revisor causing errors, or configuration mistakes.

4.2. Global and Model Configuration

The default Revisor configuration file, **`/etc/revisor/revisor.conf`** consists of multiple sections (the file is in .INI format). One is the **`[revisor]`** global section, where you specify configuration options that apply to each other section or *Model*.

The options specified in the global and model configuration sections apply to the Revisor compose in the following order:

1. The options from the global section are read, tested and set.
2. The options from a model section are read, tested and set, regardless of whether the global section had caused the setting to be set to a certain value already.

For example, if you know all the models in a configuration file are optical live media products, the configuration sections could look like the following:

Example 4.1. Example revisor.conf global vs. model configuration

```
[revisor]
# Optical live media for all models
media_live_optical = 1

[model1]
main = /etc/revisor/conf.d/revisor-model1.conf
description = The model1 product
```

```
architecture = i386
# This is already configured in the global section of
# this configuration file and can thus be removed.
#media_live_optical = 1
```

When Running the GUI

Note that when running Revisor in Graphical User Interface mode, you can still change a lot of the settings supplied by Revisor through the configuration files loaded. When you are running Revisor in GUI mode, the configuration files supply the defaults.

4.3. YUM Repository Configuration

The files in **`/etc/revisor/conf.d/`** are YUM configuration files. Revisor directs YUM through its API to use these files during the compose process, and does not handle these files itself. This chapter explains how these files are used, how you can change them and lists a few tips and tricks.

Because these files are YUM Configuration files, you can configure anything that YUM supports. See **man yum.conf** for more details.

4.3.1. $releasever and $basearch

When configuring a repository URL, make sure you do not use *`$releasever`* or *`$basearch`* variables. Since Revisor allows cross-composing distributions between different versions of the operating system, as well as different architectures, these variables need to be expanded to the value intended. Ergo, if a repository configuration has *http://server/repo/$releasever/$basearch/*, for a Fedora 12 i386 YUM configuration file you would need to change such in *http://server/repo/12/i386/*.

See also *Section 4.3.7, “Testing & Troubleshooting the YUM Configuration”*

4.3.2. Using a Local Mirror

If you have a local mirror of Fedora, you can use the **`baseurl`** configuration directive in each repository configuration section to tell YUM to use the local mirror.

Optionally, you can also disable the **`mirrorlist`**, preferably by outcommenting it, so that YUM will only use the local mirror.

The default **`baseurl`** uses **`http://download.fedoraproject.org/`**. This location may or may not be suitable for you. If you have a local mirror, you might want to change this setting here, or add your mirror to Fedora Project's Mirrorlist.

Adding your local mirror to the Mirrorlist

You can add your local mirror to the Mirrorlist, so that the Fedora Project mirrorlist redirects you to your local mirror. Additionally, systems in your local network(s) will be redirected to the local mirror. The local mirror does not have to be a public mirror in order to do so. See *http://admin.fedoraproject.org/mirrormanager/* for more details.

Alternatively, you can set each **`baseurl`** directive to the location of the repository on the local mirror.

See also *Section 4.3.7, “Testing & Troubleshooting the YUM Configuration”*

4.3.3. Using Local Files

If you have the repositories on your local filesystem, configure a **`baseurl`** of **`file:///`*`path/to/repository/`***.

Make sure to supply the correct path

Make sure to supply the correct path to the repository. **`file://`** is the "*protocol*" for the location, and the location is ***`/path/to/repository/`***. Put together, you have *three* slashes.

See also *Section 4.3.7, "Testing & Troubleshooting the YUM Configuration"*

4.3.4. Using a DVD

A DVD does not contain enough packages to rebuild the installer images. If you are using a DVD and you want to rebuild the installer images, you will need to have a network connection and a mirror you can reach.

If you have no need for rebuilding the installer images, make sure you configure Revisor to not rebuild the installer. There's a HOWTO on the subject in *Section 6.4, "HOWTO: Re-use Installer Images"*.

There is a list of required packages, but since the packages change per release and may change in the middle of the release cycle as well, we cannot hand you a list that just works. If you really want to know though, the list of packages are in Revisor's **`cfg.py`**. Maybe you can write a plugin with a command-line option that spits out the required packages per model, and optionally download all of them, and create a repository out of the downloaded packages. If you do so, please let us know at the Revisor Development mailing-list at *https://fedorahosted.org/mailman/listinfo/revisor-devel*, and we'll be glad to help you out or maintain the plugin upstream.

See also *Section 4.3.7, "Testing & Troubleshooting the YUM Configuration"*

4.3.5. Adding Third Party Repositories

When adding a third party repository, make sure you add the correct release version as well as architecture to the Revisor YUM configuration file. Verify the location for the **`baseurl`** and/or **`mirrorlist`** you configure manually or through YUM. Make sure you expand any **`$releasever`**, **`$basearch`** and **`$arch`** variables.

There's a HOWTO on adding third party repositories included in this document at *Section 6.1, "HOWTO: Add a Third Party Repository"*.

See also *Section 4.3.7, "Testing & Troubleshooting the YUM Configuration"*

4.3.6. Creating Your Own Repository

Creating your own repository is relatively simple. You take a directory, dump some RPM packages in it, and run **createrepo**. See **`man createrepo`** for more information.

People often wonder how Revisor handles comps.xml group files.

When you create your own repository, essentially a third party repository, follow the directions in *Section 6.1, "HOWTO: Add a Third Party Repository"* to add the repository configuration to Revisor's YUM configuration, since your own repository is a third party repository as well.

See also *Section 4.3.7, "Testing & Troubleshooting the YUM Configuration"*

4.3.7. Testing & Troubleshooting the YUM Configuration

Before you use the (modified) configuration file, take it for a test run to see if you have configured YUM properly.

Procedure 4.1. Testing a YUM configuration file

1.
   ```
   # yum -c /path/to/configuration/file \
       clean all
   ```

2.
   ```
   # yum -c /path/to/configuration/file \
       list kernel
   ```

Should this show errors, or fail otherwise, add `-d 9` to the command line in order to have YUM spit out more detailed debugging output. This detailed output should leave you a clue or two about what might be wrong with the configuration.

4.4. Configuring A Proxy Server

Configuring a Proxy server to be used can be done through YUM, using the `proxy=` configuration directive.

4.5. Command-line Options

With the command-line options available, you can configure options that either override what is in the configuration file or have simply not been configured using the configuration file. With the default configuration files that come with the **revisor-cli** package for example, no media products have been pre-configured in the global configuration section. In the **revisor-unity** package however, some default configuration has been applied so that Fedora Unity Re-Spins actually create CD, DVD and Rescue ISO images as well as the Installation Tree, and include the sources.

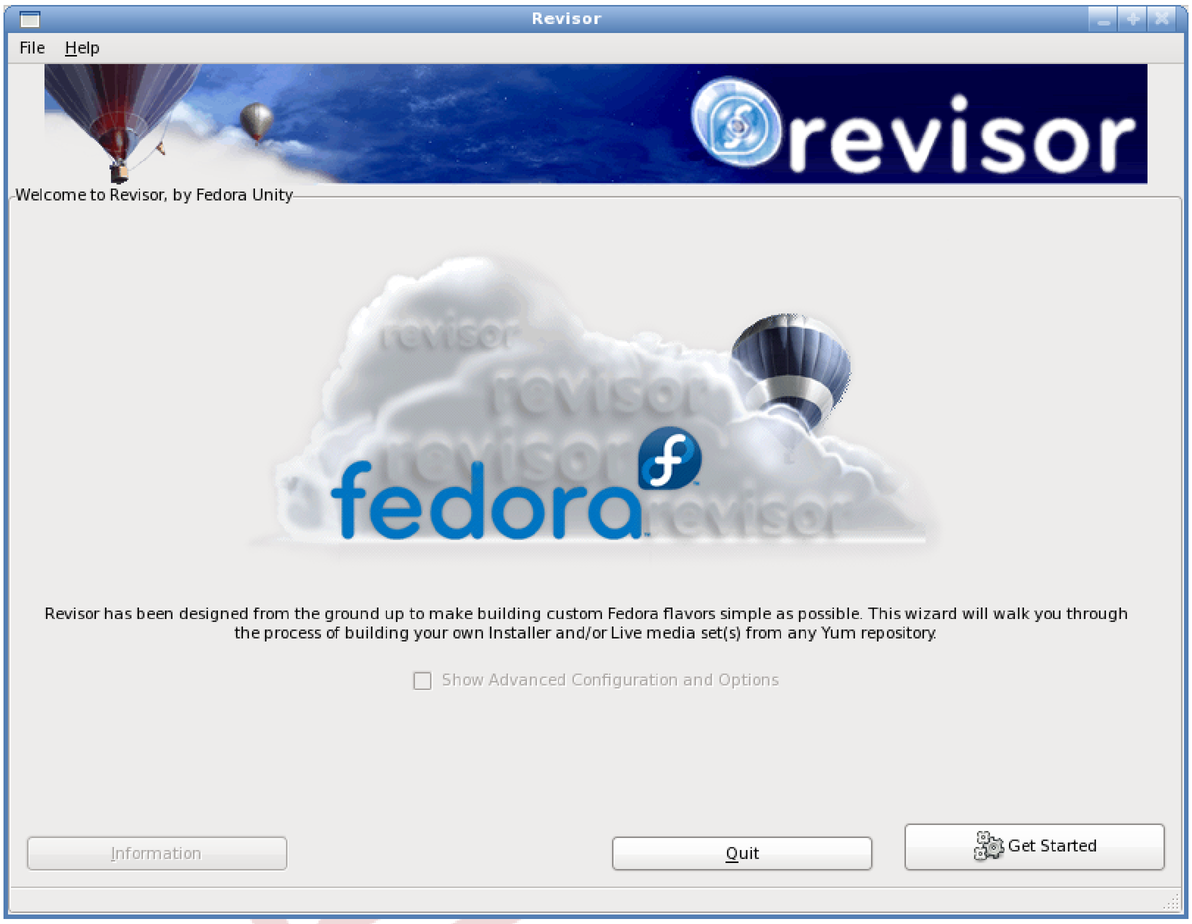
Only some configuration options have CLI parameters. Use **revisor --help** to see a complete list of configuration options you can supply on the command line.

Note that command-line options always override the configuration settings supplied in the configuration files. Additionally, when using the graphical user interface, command-line options can be overriden by GUI interaction.

Quick Start

This is a quick start guide to get you on your feet using Revisor really quickly, without any modifications, advanced options, whatsoever.

When you execute Revisor for the first time, the following screen appears:

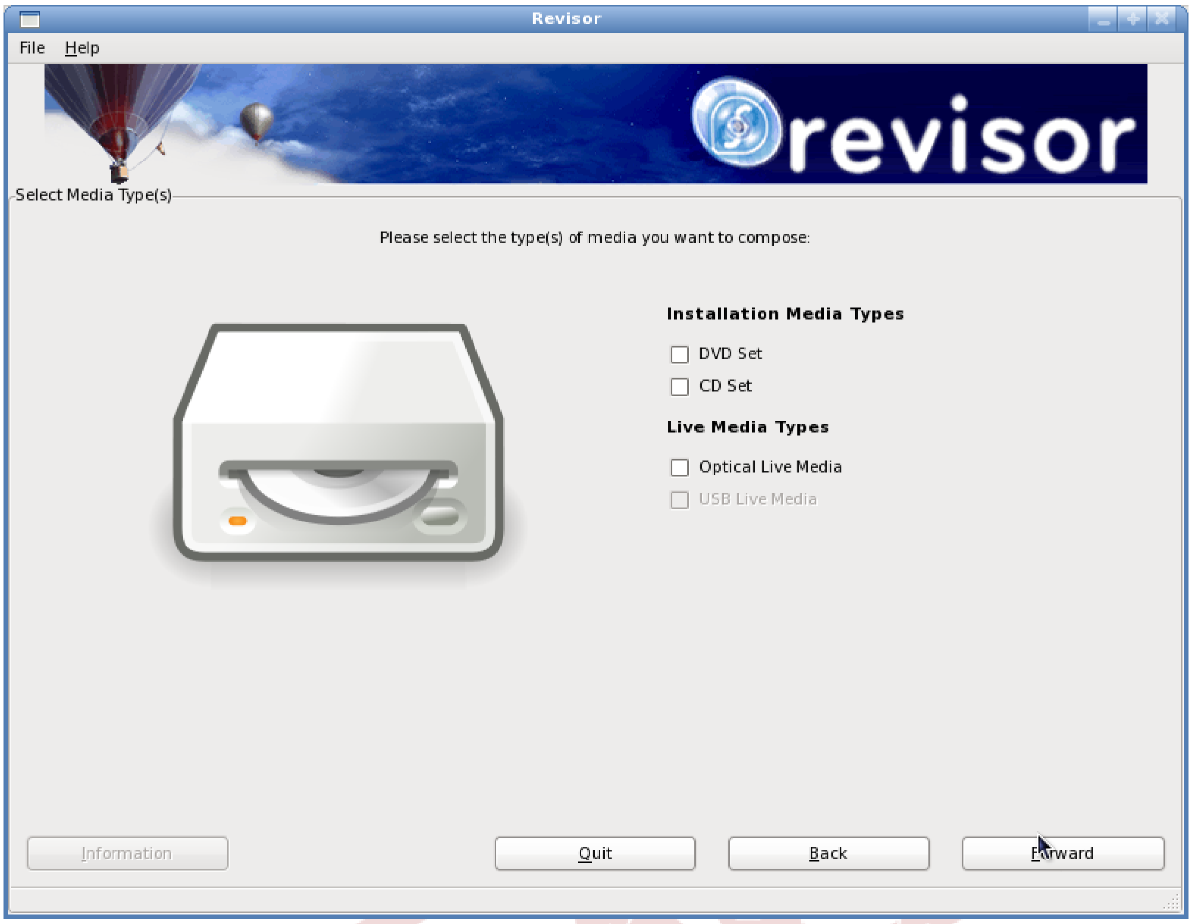

Revisor
File
Help
revisor
Welcome to Revisor, by Fedora Unity
fedora

Revisor has been designed from the ground up to make building custom Fedora flavors simple as possible. This wizard will walk you through the process of building your own Installer and/or Live media set(s) from any Yum repository.

Show Advanced Configuration and Options
Information
Quit
Get Started

Figure 5.1. Revisor Welcome Screen

The first thing you'll notice, is that the Advanced Options are disabled. Don't worry, as the advanced options are very limited at this point and do not get you anywhere right now. Press **Get Started** to get started.



Please select the type(s) of media you want to compose:

Installation Media Types

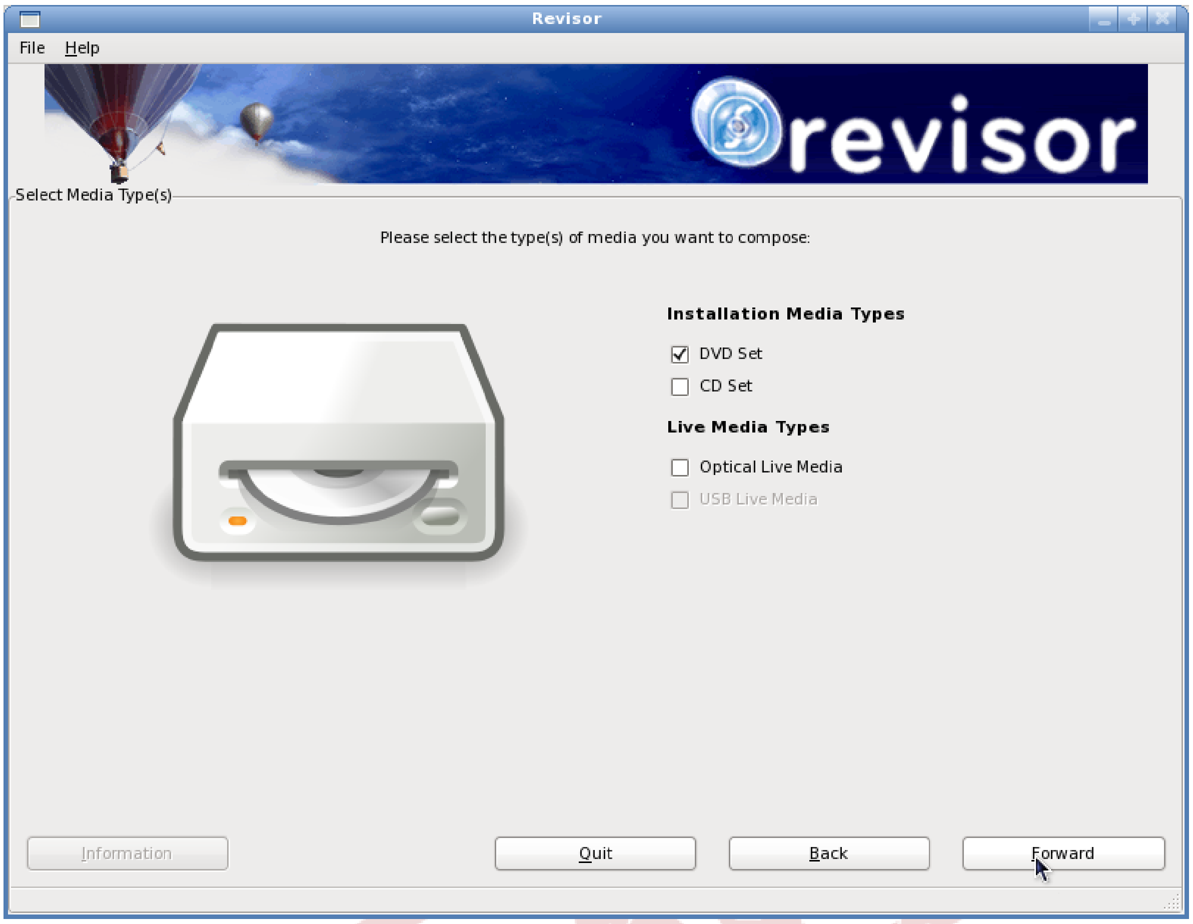
- [ ] DVD Set
- [ ] CD Set

Live Media Types

- [ ] Optical Live Media
- [ ] USB Live Media

Figure 5.2. Select Media Type(s)

Select the DVD Set to compose installation media with the maximum size of a DVD per disc.



Select Media Type(s)

Please select the type(s) of media you want to compose:

Installation Media Types

[✓] DVD Set

[ ] CD Set

Live Media Types

[ ] Optical Live Media

[ ] USB Live Media

Figure 5.3. Select DVD Set

Part II. Manual

HOWTO

This chapter includes a couple of HOWTOs on use-cases that Revisor serves.

6.1. HOWTO: Add a Third Party Repository

para

6.2. HOWTO: Create a Re-Spin

Fedora Unity is famous for creating Re-Spins, amongst other things. This, however, does not mean you need to be some kind of guru or spins master to create your own Re-Spin ;-)

This is a HOWTO on composing a Re-Spin, but does not include testing the Re-Spin. Maybe it will include testing the Re-Spin as soon as we get someone on board willing to write about it ;-)

1. First, install the **revisor-unity** package:

   ```
   # yum -y install revisor-unity
   ```

   The **revisor-unity** package includes the configuration files Fedora Unity uses to execute automatic and unattended composes for Re-Spins.

2. Execute the following for a x86_64 version of a Fedora 12 Re-Spin:

   ```
   # revisor --cli --config /etc/revisor-unity/f12-install-respin.conf \
       --model f12-x86_64-respin
   ```

6.3. HOWTO: Include folders and files on the Media

para

6.4. HOWTO: Re-use Installer Images

para

Using Kickstart

Kickstart is a configuration file format for automating installation procedures. Or at least, it was, originally. Nowadays, kickstart files are used as input to the compose tools, including Revisor.

Revisor again is unique in that it does not require a kickstart file for input. The other tools only take kickstart configuration files. Revisor though allows most of what is in a kickstart file to be configured interactively in Graphical User Interface mode.

7.1. How Kickstart Is Used

There's two cases in which a kickstart file is used differently. One is during the compose of installation media, and the other of course is during the compose of live media, or virtualization media.

7.1.1. Installation Media

In the case of installation media, the following settings are used:

- **`repo`**
  The **`repo`** command in kickstart is used when Revisor is configured to use the repositories configured in the kickstart file only. Use **`kickstart_repos = 1`** to enable this feature, or set the appropriate checkbox in the Revisor GUI.

- **`%packages`**
  The **`%packages`** section in kickstart is used to determine the RPM payload on the installation media. It can include groups and packages, and exclude packages. It accepts wildcards, both in includes and excludes of packages (but not groups).

@core and @base

By default, groups @core and @base are included in the package manifest. You can specify @base to not be included, by using **`%packages --nobase`**, but @core cannot be excluded using a kickstart package manifest.

Using **`kickstart_exact`**, you can exclude @core and @base so that you need to explicitly select them in the kickstart package manifest.

Using **`kickstart_exact_nevra`** ...

7.2. The Kickstart Package Manifest

para

Group @core

para

Group @base

para

Including groups of packages

para

Including a single package

para

Excluding a single package

para

Using wildcard matches

para

7.2.1. Using Kickstart with Package NEVRA

para

Part III. Reference

Compose Process Details

This chapter lists the details of the compose process as well as dives deep into the features of Revisor.

8.1. Overview

Of course, the compose process for installation media is a little different then the compose process for live media.

When composing, Revisor starts out doing the following:

- Revisor initiates and loads plugins, options, and defaults. At this point, Revisor has a so-called *ConfigStore* that holds all options Revisor knows about.
- Revisor reads the options from the command-line.
- Revisor reads the configuration file specified with the **`--config`** command-line parameter, or uses it's builtin default, **`/etc/revisor/revisor.conf`**.
- Revisor reads the global **`[revisor]`** section for all settings available in it's *ConfigStore* and sets those configured in the global section. Remember that if an option is not available in the *ConfigStore* but is configured in the global configuration section, it is ignored.
- If a model is specified in the configuration file's global section **`[revisor]`**, Revisor will set that model to be used and loads it.
- If a model has been specified on the command-line, with option **`--model`**, Revisor loads that model.
- When loading the model, Revisor again iterates over all the settings that are in the *ConfigStore*, checks if the setting has been configured in the model section, and adjusts the setting in the *ConfigStore* if necessary. Again remember that if the *ConfigStore* does not know about one or the other option already, that setting is ignored.
- Now that the defaults and configuration file settings have been applied to the *ConfigStore*, it is time for Revisor to look at the options specified on the command-line to see if you wanted to override anything.
- While loading each configuration setting available in the global **`[revisor]`**, model-specific sections and/or command-line, Revisor checks every settings against a function that is specifically written to check such setting. For example, the label of an ISO cannot be longer then 32 characters.
- Especially in CLI mode, these settings build up the task list for Revisor. If there's nothing to do, Revisor will throw an error explaining what's missing. If there's things to do that cannot be done in one run, Revisor will throw an error explaining that.
- In Graphical User Interface mode however, if the settings loaded so far are all OK, the GUI will start. Since you can still adjust a few settings from within the GUI, the settings loaded so far will be the defaults for configuration settings that have a dialog for you to adjust them with, throughout the rest of the process.

8.2. Installation Media

As we've explained before, composing installation media is a little different then composing live media. That's not just because installation media should start an installation procedure and live media should show you a nice, shiny, fully-functional Desktop.

For one, installation media allows split media. This means that Revisor can span the payload of the product over multiple ISO images or multiple discs, if you will. When composing installation media, Revisor basically does the following:

- Of course, Revisor goes through the loading of configuration options mentioned in the *Overview*.
- When you're done specifying options in the GUI, or when Revisor thinks it can go ahead using the options specified in CLI mode, it takes the list of packages selected from either the GUI or the kickstart **`%packages`** manifest.

  Not getting too deep into details here, yet, because some of these things are routines shared with other composing modes, but here's a few additional considerations Revisor makes when doing the package selection.

  - Normally, a kickstart **`%packages`** manifest only allows you to select package *names*. With Revisor though, you can select exact package *NEVRA* to select a certain version or architecture for the package that you want. Additionally, if a package is not available, Revisor searches the *Provides* of the available packages.

8.3. Live Media

para

8.4. Respin Mode

Revisor has a respin mode that in some aspects differs from the regular routines. It is intended to reflect behaviour of tools in use by the Fedora Project Release Engineering team as closely as possible.

Re-Spin mode only affects installation media products.

In Re-Spin mode, the way the RPM payload is determined from kickstart differs from Revisor's normal procedures. See *Chapter 7, Using Kickstart* for more details on using a kickstart package manifest.

A kickstart file's so-called *Package Manifest* usually looks like:

```
%packages
@group1
@group2 --nodefaults
@group3 --optional
package1
package2
-package3
%end
```

Which tells us the following:

- Include all mandatory and default packages from group1
- Include all mandatory packages from group2
- Include all mandatory, default and optional packages from group3
- Include package1, and package2
- Exclude package3

Depending on how you use this instructions or information, there is a slight difference in the package set that ends up on the media you compose.

8.4.1. Selecting Groups

Selecting groups has the following logic: When you load a repository you may also load the groups file (often referred to as 'comps' or 'comps.xml'). This comps file is an XML file with categories, groups (per category), and per group:

- a list of mandatory packages. If you select or include the group, these packages come with it.
- a list of default packages. If you select or include the group, these packages will come with it as a default. If you only want the mandatory, minimum set of packages for this group, in a kickstart package manifest append **`--nodefaults`** to the group line or in the Revisor GUI, right-click on the group and choose *Deselect all packages*.
- a list of optional packages. If you select a group you have not yet selected these packages. To select the optional packages of a group, in a kickstart package manifest append **`--optional`** to the group line or in the Revisor GUI, right-click on the group and choose *Select all optional packages*.
- a list of conditionals. If you select this group, these conditionals are thrown into the package sack and transaction information and include or exclude other packages. Suppose you select the '@nl-support' or “Dutch Support” group from the Languages or Localization category, you would end up with support for the Dutch language in all applications that have that kind of support.

8.4.2. Select Matching Packages

This is the logic Revisor applies when running in Re-Spin mode (on the CLI, specify **`--respin`**). It imitates the behavior pungi has, and thus enables you to copy that behavior. Note that **`--respin`** has other implications as well.

First of all, it iterates the groups in the kickstart package manifest. For each group, it appends the names of the mandatory packages to a list, and depending on the additional parameters specified with that group (**`--nodefaults`** or **`--optional`**), appends the names of the other packages in that group as well.

Then it iterates over the package names in the package manifest. These package names are appended to the same list of package names too. This includes package 'names' with some sort of wildcard (?, or *).

Then it iterates over all the excluded packages, appending each of those to the YUM configuration exclude list.

Now that Revisor has a very simple, flat list of package names, it uses YUM's internal matching logic 5 to get what packages in the repositories matched exactly (by name), matched (by wildcard) and did not match at all. Revisor then selects the exact matches and matches, adding them to the transaction.

8.5. Dependency Resolving

Dependency resolving is the area where some of the efficiency Revisor can gain for you comes from. While of course there is specific reasons to do things one way, or the other, most people I speak to about Revisor, it is not very clear why, or what Revisor does in this area. First of all, there's two ways of resolving dependencies:

1. **Inclusive Dependency Resolving**

   Iterate all packages in the transaction and list their requirements, then for each of those requirements, find all packages that provide a matching capability, add those packages to the transaction, and don't forget to add the requirements those packages have themselves, back into the pile of (unmet) requirements.

2. **Exclusive Dependency Resolving**
   Iterate all the packages and for each of the requirements found, find the best package that meets the requirement. This is also YUMs default behavior. Anaconda uses YUM during the installation, and this is the behaviour of YUM used during the installation.

8.5.1. Inclusive Dependency Resolving

Hypothetically, you could describe inclusive dependency as follows:

```
final_packages = []
more_to_do = True
while more_to_do:
more_to_do = False
for package in packages:
    if package in final_packages:
        continue

    dependencies = find_package_dependencies()
    for dependency in dependencies:
        pulled_in_package = pull_in_dependency()
        if pulled_in_package not in final_packages:
            packages.append(pulled_in_package)
            more_to_do = True
```

So, what does this mean? Basically, it means that if there is a requirement for a capability, all packages providing that capability are being pulled in. Now imagine package 'foo' requires capability 'web-client'. There's a number of packages providing that capability, right? So you get Firefox, lynx, elinks, konqueror, safari, Netscape, Internet Explorer, emacs, for free! All of those pull in their own dependencies also, of course.

Note

If you catch this before it catches you, you can prevent the packages from being pulled in during dependency resolving by not making the package available in the *Package Sack* in the first place, using the **`-firefox`** syntax in the kickstart package manifest, and setting **`kickstart_uses_pkgsack_excludes`** to 1.

Note

You may have thought of it; pulling in packages this way may give you a package set (or *RPM payload*) that has conflicting packages. Imagine package **foo** requiring capability **bar**, which is provided by two packages that conflict with one another (either on explicit **`Conflicts:`** RPM header or file level). Both will be pulled in, hence disabling you to install everything (**`'*'`** or -previously- **`@Everything`** in the kickstart package manifest).

8.5.1.1. When This Makes Sense

If you are composing a large distribution of which 3 million users in even so many different situations having so many different expectations and desires, you will want this behaviour, since you won't be able to determine which one of the packages for each capability someone in that group wants, and which one may not want. Or, in case of upgrades, what the system needs. Shipping them all on the same media is the best solution in these cases.

8.5.1.2. When This Does Not Make Sense

- When creating installation media to be installed unattended, or to be used in conjunction with deployment strategies
- When creating installation media to be upgrading PCs you have controlled from the beginning, such as in a company
- Installation for a small group of users or systems
- When creating minimal installation media, or media with a minimal RPM payload.
- When creating installation media that is to be used with installing "Everything" in the RPM payload.

8.5.2. Exclusive Dependency Resolving

Exclusive dependency resolving is what YUM does when you execute a **yum install**. Unless you've specified one of the packages satisfying any of the dependencies in the transaction, YUM is going to look up the best match for you. This results in the installation of only one package providing the requirement(s) of other packages, rather then all packages providing said requirement being installed.

As an example, imagine you install a package foo which requires capability web-client. Using exclusive dependency resolving, YUM would select one package providing the web-client capability whereas inclusive dependency resolving would include all packages providing the web-client capability.

During the installation procedure, one of the major features of installation media, anaconda is going to use YUM dependency resolving to satisfy all the dependencies.

Installation Procedure !== Upgrade Procedure

Note that an installation procedure is not the same as an upgrade procedure. With an installation procedure for example, you have control over the partitioning layout whereas with an upgrade procedure, you have none. More importantly, during an upgrade procedure, the (already installed) system has an existing package set which needs to be updated/upgraded and thus could possibly introduce dependency resolving problems, because of third party packages installed on the system, or because the media used to upgrade the system with does not contain the software packages needed to complete the upgrade RPM transaction.

8.6. Copying Arbitrary Files Onto the Media

With **`--copy-dir`**, you can specify a path Revisor should copy onto the media.

Installation Media

In the case of installation media, the path specified with **`--copy-dir`** will be copied recursively to the **`files/`** sub-directory at the root of the ISO image (or the first ISO image if you compose split media).

A few use-case examples:

- If one kickstart profile is not enough for you to deploy the product onto your systems, create a directory that holds multiple kickstart files and specify the path to that directory using **`--copy-dir`**. The kickstart files now end up available to the installation procedures as **`cdrom:/files/*.ks`**, and can thus be used by specifying them on the kernel cmdline (**`ks=cdrom:/files/profile1.ks`**), or, when used in combination with **`--isolinux-cfg`** from the *Isolinux Plugin*, can be added as an option in the isolinux menu.

- If you have files or scripts that need to be copied onto, or run on, the installed system before it attempts to reboot and operate normally, you can use **`--copy-dir`** to make these files and scripts available during the installation and copy or execute them from either **`%pre`** or **`%post`** scripts.

Live Media

In the case of live media, the path specified with **`--copy-dir`** will be copied recursively onto the root directory (**`/`**) of the live media filesystem (which is probably loop-mounted onto **`/var/tmp/revisor/`**).

If, for example, you want to copy a home directory onto the live media, and the home directory you want to copy is at **`/home/user1/`** on the composing system, you copy this directory so that the root of that new directory has a sub-directory **`home/`** which in turn contains a sub-directory **`user1/`**:

```
$ mkdir -p /tmp/something/home/
$ cp -a /home/user1 /tmp/something/home/.
$ revisor [options] --copy-dir /tmp/something/
```

8.7. Cleaning Up

Revisor tends to clean up after itself by default. If a product compose succeeds, you (probably) don't need to change this default behaviour. However, by default, Revisor tends to leave the YUM cache directories untouched. This is to prevent you from having to download all the packages a second, third or more times when you run another compose.

To change this default behaviour, Revisor has an option **`--clean-up`**. The default value for this option is **`1`**, meaning Revisor will clean up it's temporary, compose-specific files, but no files that could be re-used. Specifying **`--clean-up=0`** will cause Revisor to leave everything behind and not clean anything up at all. This is most ideal for troubleshooting purposes, where one needs to examine the temporary, compose-specific files and see what went wrong. To clean up everything however, because for example you might be low on disk-space, use **`--clean-up=2`**. Revisor will then also clean up the files that could be re-used.

8.7.1. Exception to the Rule

There's one exception to the rule of cleaning up. **`/var/tmp/revisor/`**, or put more accurately, the path specified as the **`installroot`** in the YUM configuration file configured with the model used to compose the product, will not be cleaned up afterwards. When composing live media, this directory may still be in use as a mount-point for the live media filesystem. Removing this directory recursively in these cases would not make sense.

Plugins

para

9.1. Upstream Plugins

Plugins available from upstream, maintained by upstream

9.1.1. Cobbler Plugin

The Cobbler plugin is able to put the product composed into a Cobbler environment, by handing off the built product to the existing Cobbler infrastructure as a *distro*, and creating a *profile*.

Using this module, one can automatically import the Revisor product into a Cobbler environment, and immediately use the new Cobbler *profile* to start deploying or automated testing, maybe.

9.1.2. Composer Plugin

para

9.1.3. Delta Plugin

A small change to a ISO image does not require you to download the complete ISO image if you have a copy of the old ISO image.

Only applicable to (...)

The generation of Delta ISO images is only applicable to situations in which the ISO image does not contain SquashFS images. SquashFS images are smaller, but all SquashFS images are unique. Since the Delta principle is based on similarities, and no two SquashFS images are alike, creating a Delta on two ISO images containing SquashFS images will lead to a Delta pratically the same size as the SquashFS image. For Live Media that compresses the ext3 filesystem image into a SquashFS image, since that SquashFS image is probably over 97% of the size of the ISO image, creating Delta images for compressed Live Media does not make sense. For installation media however, most RPMs would be similar as well as (potentially) the installer images.

9.1.4. GUI (Graphical User Interface) Plugin

Yes, the Graphical User Interface for Revisor is actually a plugin.

9.1.5. HUB Plugin

para

9.1.6. Isolinux Plugin

The isolinux plugin adds the **`--isolinux-cfg`** command-line option to Revisor. Specify a file here, and the original **`isolinux.cfg`** that is built as part of the compose process is replaced by the **`isolinux.cfg`** specified.

9.1.7. Jigdo Plugin

para

9.1.8. Mock Plugin

para

9.1.9. Rebrand Plugin

The rebrand plugin hooks in to Revisor at several different stages. The goal of this plugin is to ensure no trademarked packages end up on the media. Trademarked packages may include **fedora-logos**, **redhat-logos**, and so forth.

The plugin adds a **`--rebrand`** option, to which you can specify the name of your new product. When rebranding Fedora to Omega for example, specifying **`--rebrand Omega`** would be sufficient to make sure the product does not have any Fedora trademarks.

9.1.10. Reuse Installer Images Plugin

para

9.1.11. Server Plugin

para

9.1.12. Virtualization Plugin

para

9.1.13. WUI (Web-based User Interface) Plugin

para

9.2. Writing Your Own Plugins

para

Tweaking the build process

para

10.1. Reusing Existing Installer Images

para

10.2. Building The Installer Images in Mock

para

10.3. Omitting isomd5sums

para

10.4. Omitting SHA1SUMS

para

Tips and Tricks

para

11.1. The spin-kickstarts Package

para

11.2. Even More Debugging

something about using -x to buildinstall scripts

11.3. Kickstart Validator

something about using -x to buildinstall scripts

11.4. Using Mirrormanager for Mirror Redirection

Something about using Mirrormanager to redirect you to the local mirror (so you do not have to edit YUM configuration files).

11.5. Using The localrepo DNS Alias

Something about using the localrepo DNS alias to point to your local mirror (either through real DNS or through /etc/hosts), so you do not have to edit the YUM configuration files.

Frequently Asked Questions

para

How Does Revisor Handle Comps?

para

What Are Installer Images?

para

What is the relationship between Revisor and Pungi?

Where pungi builds a bunch of RPMs into ISO images and installation trees through one single procedure, perfect for Release Engineering on something like the Fedora Project, Revisor does it different entirely.

What is the relationship between Revisor and livecd-tools?

Revisor depends on livecd-tools for the composing of live media. Creating the filesystem to install the packages to, turning that image file into a SquashFS file, and applying the settings inside the chroot.

Why Rebuild Installer Images?

para

How do I create an updates.img?

para

Testing

The following test cases describe different types of testing a new Revisor release.

13.1. Simple Test Cases

This section has a few simple test cases ensuring configuration shipped with Revisor works as anticipated.

13.1.1. Packages

Install the **revisor-cli**:

```
# yum --enablerepo=updates-testing install revisor
```

Installed are all dependencies for the Revisor CLI interface. Make sure **spin-kickstarts** is installed, a package for sample kickstarts.

Starting Revisor as follows should not show any error messages related to Revisor attempting to start up it's GUI interface:

```
# revisor
```

Configuration Files

The following configuration files should exist:

- `/etc/revisor/revisor.conf`

Each section should have a configuration file listed as `main`.

And, of course, every configuration file listed in each section. In this case, the following snippet is easy enough:

```
$ i=0; \
 configfiles="`grep ^main /etc/revisor/revisor.conf | \
 sed -r -e 's/^main.*=\s*(.*)/\1/g'`"

for configfile in $configfiles; do \
 [ ! -f $file ] && i=1; \
done; \
echo $i
```

Another way to test the configuration file is to execute:

```
$ revisor --list-models >/dev/null
```

If everything is well, since `STDOUT` is redirected to `/dev/null`, you should see no messages at all.

13.1.2. Configuration Files

The main Revisor configuration file is `/etc/revisor/revisor.conf`. The file lists a series of models, each having their own YUM configuration file in `/etc/revisor/conf.d/`.

Testing

- For each model in **`/etc/revisor/revisor.conf`**, the **`main`** setting for that model should point to a valid file.
- Each YUM configuration file should work. To verify, run the following command for each configuration file:

  ```
  $ yum -c $file list kernel
  ```

Known Errors

- Revisor has baseurls in YUM repositories set to *http://localrepo*. This URL will not be retrievable for many people, but allows the developers to quickly change mirrors.
- Repositories that are unavailable at the moment of testing will throw errors Revisor can't do anything about.
- If the directories **`revisor-yumcache/`** and **`revisor/`** in **`/var/tmp/`**, the default working directory, are not writeable for the user then YUM will throw permission denied errors.

  Remove **`/var/tmp/revisor/`** and **`/var/tmp/revisor-yumcache/`** or run the command with root permissions.

13.1.3. Requirements for Compose Results

Although heavily dependent on Anaconda for this part, these are still requirements

ld-linux.so.2

In the **`initrd.img`** of the composed product, if 32-bit, **`/lib/ld-linux.so.2`** (or any other version) should link to **`/lib/ld-2.9.so`** (or any other version). If **`/lib/ld-linux.so.2`** links to itself, the media will fail to install.

How to test

In a terminal, type the following command:

```
$ lsinitrd /path/to/initrd | grep ld-linux
```

See also: *https://www.redhat.com/archives/anaconda-devel-list/2009-February/msg00115.html*

13.2. Complex Test Cases

para

13.3. Specific Test Cases

para

Development

This chapter sheds some light on development of Revisor, such as different branches and maintenance policies, versioning schemas, etcetera.

This part of the documentation relies on whether you have **sudo** set up properly. If you have not, you're on your own.

14.1. Running Revisor from Source

The latest code in GIT can be built into a RPM you can install but one of the advantages of having the complete source tree is that you can run it directly from that source tree so that when you pull in the next updates you do not have to rebuild the RPM. Note that we do not bump the version number for every little change we make, and as such the RPM built does not allow you to use **`rpm -Uvh`** or **`rpm -Uvh --oldpackage`**. Of course, Revisor's Makefiles also allow **make install**, but that leaves a number of unmanaged files on your computer you would have to track down manually in order to remove Revisor completely.

Cannot have Revisor RPMs installed

When running revisor from within the source tree, you cannot have any of the Revisor packages installed. Having Revisor RPM packages installed regardless will mess up the GIT repository or source tree.

To run Revisor from within the source tree, checkout the master branch, and run the **`./switchhere`** script:

```
$ ./switchhere
```

The **`./switchhere`** script does the following:

- Symlink **`/etc/revisor/`** to ***`$PWD`*`/conf/`** so that **`/etc/revisor/revisor.conf`**, the primary configuration file, and **`/etc/revisor/conf.d/`**, the configuration directory, are valid (the symlink causes the actual file and directory to be found in ***`$PWD`*`/conf/`**)
- Create the **`/usr/share/revisor/`** directory so that a couple of symlinks can be created from within that directory:
- In Revisor 2.1.0 (development version in branch master), this includes:
  - **`/usr/share/revisor/ui => `*`$PWD`*`/revisor/modgui/glade/`**
  - **`/usr/share/revisor/pixmaps => `*`$PWD`*`/revisor/modgui/glade/pixmaps/`**
  - **`/usr/share/revisor/comps => `*`$PWD`*`/conf/`**
- In Revisor 2.0.5 (branch F-7, F-8 or EL-5), this includes:
  - **`/usr/share/revisor/ui => `*`$PWD`*`/glade/`**
  - **`/usr/share/revisor/pixmaps => `*`$PWD`*`/glade/pixmaps/`**
  - **`/usr/share/revisor/comps => `*`$PWD`*`/conf/`**

- In Revisor 2.1.0, also create symlinks from within the appropriate `/usr/share/man/man$x/` directories to the source for these man pages in `$PWD/doc/`.

From this moment on, you should be able to run:

```
$ ./revisor.py
```

Root privileges required

Note that revisor needs root privileges to run, and that you'll need to sudo or su-c to gain those. Use here whatever you find the most convenient; Revisor though should have a nice error message when run without those privileges.

14.1.1. Installing the Required Packages

To be able to run Revisor from within the source tree, you'll need to install the required packages for each component, of course.

To get a current list of those packages, use:

```
$ rpmquery --specfile --qf="%{REQUIRES}\n" revisor.spec | sort | uniq | xargs -n 1 repoquery
 --requires --alldeps --resolve
```

14.2. Building Revisor Packages

para

14.3. Tickets

bugzilla, trac

14.4. Adding a new spin or remix

1. Add the appropriate models in the appropriate configuration file for Revisor.
2. Add the appropriate configuration file to the appropriate automake Makefile
3. Run autoreconf && ./configure && make rpm to verify the rpm building
4. Create the model's YUM configuration files with the following repositories:
   - fedora, enabled, pointing to Everything
   - fedora-source, disabled, pointing to Everything
   - fedora-updates, enabled, pointing to the updates repository
   - fedora-updates-source, disabled, pointing to the updates repository
   - anaconda-updates, enabled, pointing to the anaconda updates repository
   - anaconda-updates-source, disabled, pointing to the ananconda updates repository

14.5. Versioning Schema

para

14.6. Release Procedure

para

Appendix A. Revision History

Revision 1.0

Index

Part IV. Appendices

Appendix A. Terminology

Model

A model in Revisor describes a product.

Package Manifest

A package manifest is the list of groups and packages to include or exclude from a transaction, in a kickstart configuration file.

Remix

A Fedora Remix is a product based on Fedora, with Fedora packages and optionally, other packages as well, such as those from third-party repositories.

Re-Spin

A Fedora Re-Spin is a product that is composed for the single purpose of including updated software packages into the product. It uses the same compose procedure as the media that the Fedora Project composes and releases, but includes updates.

Fedora Unity releases Fedora Re-Spins every so often, twice or trice per release.

Spin

A Fedora Spin is a custom set of software packages, often for a specific audience. Spins include a KDE Spin, which contains KDE software packages rather then the Desktop spin, which is based around GNOME. Similarly, there are XFCE, LXDE, Sugar, Education, Games and Developer Spins.

Fedora Spins have gone through the Spins Process (*http://fedoraproject.org/wiki/Spins_Process*), and have been approved trademark usage by the Fedora Project Board.

Package Sack

When YUM creates a list of packages available from the repositories configured, including package metadata such as dependencies and provided capabilities for each package, YUM creates a PackageSack. It's basically a large bag with all Package Objects, filtered by compatible architectures for the configured architecture.

Appendix B. Configuration Reference

This is the configuration reference for Revisor. Options are listed in alphabetical order.

B.1. Configuration Options

Table B.1. Configuration Options

Configuration Options			
Configuration Directive		**CLI Option**	
	Possible Values	**Default**	**Context**
	Description		
`answer_yes`		**`-y`**, **`--yes`**	
	0, 1	0	global, model
	Answer *yes* to all questions. This enables you to run an automated, unattended compose.		
`clean_up`		**`--clean-up`**	
	0, 1, 2	1	global, model
	Should Revisor not clean up at all (0), clean up it's temporary build data (1), or everything (2). Note that *everything* includes the yum cache.		
`copy_dir`		**`--copy-dir`**	
	`[dir]`	False	global, model
	A directory tree to copy onto the media created. In the case of installation media, the contents of the directory specified are copied onto **`cdrom:/files/`**. In the case of live media, the contents of the directory specified are copied onto the root filesystem of the live system. See also *Section 6.3, “HOWTO: Include folders and files on the Media”*.		
`copy_local`		**`--copy-local`**	
		False	global, model
	Tell Revisor to copy files, even when they are local. This applies to relative corner-cases where the repositories or the **`destination_directory`** is mounted over NFS, and some actions cannot be performed.		
`debuglevel`		**`-d`**, **`--debug`**	
	0 - 9	0	global, model
	The level of debugging. 0 is the lowest debug level, whereas 9 is the highest. Revisor turns up the volume quickly. The logfile on debug level 9 may very easily become 20-30MB.		
`destination_directory`		**`--destination-directory`**	
	`/srv/revisor/`	**`[path]`**	global, model
	The destination directory for the product. Revisor creates a sub-directory with the name of the model used, and places the product in that directory.		
`getsource`		**`--source`**	
		False	global, model
	Whether to obtain the source along with the binary RPMs used. This is False by default, and therefor the source is not included by default. Note that if you are distributing your		

Configuration Options			
Configuration Directive		CLI Option	
	Possible Values	Default	Context
	Description		
	product to third parties, you need to be able to provide the sources along with the binary product.		
`include_bootiso`			
	0, 1	0	global, model
	Whether to include the relatively large boot.iso on the optical installation media created. Setting this to 0 will still include boot.iso in the installation tree created (if configured with **`media_installation_tree`**[1])		
`kickstart_default`		**`--kickstart-default`**	
	0, 1	0	global, model
	Whether to set the isolinux.cfg entry that makes the installer use the kickstart included on the media, as a default.		
`kickstart_exact`		**`--kickstart-exact`**	
	0, 1	0	global, model
	Tells Revisor to ignore @core and @base groups (like with **`%packages --nobase`**) and only add what is in the package manifest.		
`kickstart_exact_nevra`		**`--kickstart-exact-nevra`**	
	0, 1	0	global, model
	Tells Revisor to only add what is in the package manifest. In addition, if the transaction changes (because of dependency resolving, for example), Revisor will stop composing.		
`kickstart_file`		**`--kickstart`**	
	`[file]`	0	global, model
	What kickstart file to use. When in CLI mode, this is a mandatory setting.		
`kickstart_include`		**`--kickstart-include`**	
	0, 1	0	global, model
	Whether to include the kickstart on the media so that the installer may find it as **`cdrom:/ks.cfg`**.		
`kickstart_repos`		**`--kickstart-repos`**	
	0, 1	0	global, model
	Whether to use the **`repo`** directives in the kickstart provided to Revisor. Useful in cases where the **`repo`** directive in the kickstart file provided points to an arbitrary repository such as a repository mirrored by Cobbler, since there is no default URI location for those repositories -and Cobbler refers to them in the kickstart (template) using **`$yum_repo_stanza`**		
`kickstart_save`		**`--kickstart-save`**	
	0, 1	0	global, model
	Where to save the resulting kickstart. In GUI mode, when changes to the package set can be applied, saves those changes out into a new kickstart file.		
`media_installation_bluray_duallayer`		**`--install-bluray-dl`**	
	0, 1	0	global, model

Configuration Options			
Configuration Directive		**CLI Option**	
	Possible Values	**Default**	**Context**
	Description		
	Whether to create Bluray Duallayer installation media (47GiB).		
media_installation_bluray		**--install-bluray**	
	0, 1	0	global, model
	Whether to create Bluray installation media (23GiB).		
media_installation_bluray_duallayer		**--install-bluray-dl**	
	0, 1	0	global, model
	Whether to create Bluray Duallayer installation media (47GiB).		
media_installation_cd		**--install-cd**	
	0, 1	0	global, model
	Whether to create CD installation media (685MiB).		
media_installation_dvd		**--install-dvd**	
	0, 1	0	global, model
	Whether to create DVD installation media (4.3GiB).		
media_installation_dvd_duallayer		**--install-dvd-dl**	
	0, 1	0	global, model
	Whether to create DVD Duallayer installation media (8.0GiB).		
media_installation_tree		**--install-tree**	
	0, 1	0	global, model
	Whether to create a an installation tree[2] (for publication over HTTP or FTP, or through Cobbler). No size limit.		
media_installation_unified		**--install-unified**	
	0, 1	0	global, model
	Whether to create a unified ISO, installation media (no size limit).		
media_live_optical		**--live-optical**	
	0, 1	0	global, model
	Whether to create Optical Live media (size unknown).		
model		**--model**	
	[model]		global
	The model to use.		
report_sizes		**--report_sizes**	
	0, 1	0	global, model
	Report the sizes of RPM packages used. Lists the biggest packages in the transaction		
mode_respin		**--respin**	
		False	global, model
	Whether Revisor should operate in *respin* mode. See also *Section 8.4, “Respin Mode”*		
working_directory		**-d**, **--debug**	

Configuration Options			
Configuration Directive		**CLI Option**	
	Possible Values	**Default**	**Context**
	Description		
	0 - 9	0	global, model
	The level of debugging. 0 is the lowest debug level, whereas 9 is the highest. Revisor turns up the volume quickly. The logfile on debug level 9 may very easily become 20-30MB.		

[1] Note that the installation tree is always created. See *Chapter 8, Compose Process Details* for more details.

[2] Note that the installation tree is always created. See *Chapter 8, Compose Process Details* for more details.