
# Creating An Application Image with rBuilder Online

## Getting Started

The first step is to download and install rPath Linux. A standard build is available for download here: ftp://download.rpath.com/pub/linux/

Next you need to create a project at http://www.rpath.com. Instructions are provided on the site for this step. Creating a project will provide you with a web interface for collaborative development on your Linux application image and a central source/binary repository for your code.

## Building the Build Environment

The project you created has a repository that must be filled using Conary and cvc from your local system. To start this process you need a build environment to work from locally. In your home directory you need to create a directory for your build trees. Substitute <projectname> with your project name:

```
[user@localhost ~]$ mkdir conary
[user@localhost ~]$ cd conary
[user@localhost conary]$ mkdir <projectname>
[user@localhost conary]$ cd <projectname>
[user@localhost <projectname>]$ mkdir builds cache src
```

The primary configuration file that needs to be set up in your build environment is the conaryrc file. Local conaryrc files override conflicting sections of the user level .conaryrc file or the global /etc/conaryrc file, so it is important to have local conaryrc files for your build directories that point to the project repository.

```
[user@localhost <projectname>]$ cd src
[user@localhost src]$ pwd
/home/nathan/conary/<projectname>/src
[user@localhost src]$ gedit conaryrc
```

Use your editor of choice to create a file that looks more or less like the following:

```
repositoryMap       <projectname>.rpath.org https://user:password@<projectname>.rpath.org/conary/
installLabelPath    <projectname>.rpath.org@rpl:devel
buildLabel          <projectname>.rpath.org@rpl:devel
buildPath           /home/<user>/conary/<projectname>/builds
lookaside           /home/<user>/conary/<projectname>/cache
contact             <youremailaddress>
name                <user>
```

## Creating the group Recipe

Now you can create the group recipe that will define your distribution. The first thing is to create a new package with cvc. cvc is the tool that checks things in and out of a repository, so you will be using it quite a bit throughout the build process. The way cvc works is by making all changes locally and then commiting them to the repository

when you tell it to. So, you can tell it to create the new package, but it does not do anything to the online repository until you explicitly tell it to commit the changes.

```
[user@localhost src]$ pwd
/home/<user>/conary/<projectname>/src
[user@localhost src]$ cvc newpkg group-<projectname>
[user@localhost src]$ ls
conaryrc  group-<projectname>
[user@localhost src]$ cd group-<projectname>
[user@localhost group-<projectname>]$ ls
CONARY
```

For ease of use, don't forget to copy the conaryrc file from your src directory down into this directory as well, or simply set up a symlink back to that conaryrc.

```
[user@localhost group-<projectname>]$ ln -s ../conaryrc conaryrc
```

Now create the group-<projectname> recipe that will define what is going to go into your distribution. There are a bunch to look at in the repositories as examples, but here is a simple one (use spaces, not tabs in this file):

```
[user@localhost group-<projectname>]$ gedit group-<projectname>.recipe


class Dist(GroupRecipe):                                    #Says you are building a group recipe
   name = 'group-<projectname>'                             #Names the group
   version = '0.1'                                          #Versions this group recipe
   autoResolve = True                                       #Tells the build system to resolve dependencies
   def setup(r):
      r.setLabelPath('<projectname>.rpath.org@rpl:devel',   #sets the repositories to pull from
                     'conary.rpath.com@rpl:rpl1',           #these should all be lowercase
                     'contrib.rpath.org@rpl:devel')
      r.addTrove('<myspecialapplication>')                  #includes your application
      r.addTrove('group-core', 'conary.rpath.com@rpl:rpl1') #includes the minimal rPL components
```

It is important to notice the autoResolve = True statement. This is the statement that tells the build system that you want it figure out what dependencies the packages listed in the recipe need.

It is always possible to build a distribution without autoResolve if you would rather specify the components manually.

You can always include more applications to resolve against with additional r.addTrove commands. If <myspecialapplication> is not already packaged and imported into rBuilder you will need to do that before building the distribution as well. Additional information on packaging applications with Conary is available at http://www.rpath.com.

### Cooking The Group

Now that you have built your group-<projectname>.recipe file it is time to verify that it works. You do this by cooking the group.

```
[user@localhost group-<projectname>]$ pwd
/home/<user>/conary/<projectname>/src/group-<projectname>
[user@localhost group-<projectname>]$ cvc cook group-<projectname>.recipe
info: Building group-<projectname>
Created component: group-<projectname> /<projectname>.rpath.org@rpl:devel/0.1-4-1/local@local:COOK/2 is: x86(cmov
i486 i586 i686 ~!mmx ~!x86_64) use: use(!bootstrap ~!builddocs buildtests gcj ~glibc.tls ~!kernel.debug ~!kernel.debugdata
~!kernel.numa ~!kernel.smp krb ~nptl pam pcre ~!pie ~!sasl ~!selinux ~!slang ssl)
```

```
Changeset written to: group-<projectname>-0.1.ccs
```

What got created by this cook is a local changeset. The way that conary works is not by creating a static package, but rather by creating changesets that define the differences between two known states, much like a diff file.

The “conary showcs” command can be used to look at the local changeset to see what it specifies:

```
[user@localhost group-<projectname>]$ pwd
/home/<user>/conary/<projectname>/src/group-<projectname>
[user@localhost group-<projectname>]$ ls
CONARY  conaryrc  group-<projectname>-0.1.ccs  group-<projectname>.recipe
[user@localhost group-<projectname>]$ conary showcs group-<projectname>-0.1.ccs
Note: changeset has no primary troves, showing all troves
group-<projectname>                0.1-4-2 (absolute)
      New Flavor: is: x86(cmov i486 i586 i686 ~!mmx ~!x86_64)
                  use: use(!bootstrap ~!builddocs buildtests gcj ~glibc.tls ~!kernel.debug ~!kernel.debugdata ~!kernel.numa
~!kernel.smp krb ~nptl pam pcre ~!pie ~!sasl ~!selinux ~!slang ssl)
```

You can give the conary showcs command a directive to give all available information as well:

```
[user@localhost group-<projectname>]$ conary showcs --all group-<projectname>-0.1.ccs
```

### Adding and Checking the Group Into the Repository

Now that you have verified that the group cooks correctly, it is time to check it into the repository.

```
[user@localhost group-<projectname>]$ pwd
/home/<user>/conary/<projectname>/src/group-<projectname>
[user@localhost group-<projectname>]$ cvc add group-<projectname>.recipe
[user@localhost group-<projectname>]$ cvc commit --message "note explaining updates"
```

These files will now show up in the repository browser on rBuilder Online. To cook the version that you have now uploaded into the repository, you need to issue a cvc cook command with the package name and not just the specific file name (ie., without the .recipe).

```
[user@localhost src]$ pwd
/home/<user>/conary/<projectname>/src/group-<projectname>
[user@localhost group-<projectname>]$ cvc cook group-<projectname>
info: Building group-<projectname>
Created component: group-<projectname> /<projectname>.rpath.org@rpl:devel/0.1-4-2 is: x86(cmov i486 i586 i686 ~!mmx
~!x86_64) use: use(!bootstrap ~!builddocs buildtests gcj ~glibc.tls ~!kernel.debug ~!kernel.debugdata ~!kernel.numa
~!kernel.smp krb ~nptl pam pcre ~!pie ~!sasl ~!selinux ~!slang ssl)
Changeset committed to the repository.
```

Now if you go back to the repository you will see that the group-<projectname> has an entry as a cooked trove, not just as a source file. You may also see two new groups, group-os and group-core. These may be generated by group-<projectname> as part of the build process.

### Generating a Release

Back on the rpath.org website go the homepage for your project. Using the navigation bar on the left hand side of the window select “releases”. Once on the releases page select “Create a new release”. This will take you to the window where you can select which group file to base a release on. You should select the 'group-os' that was generated when you cooked your project group if available; otherwise select group-<projectname>. A full installable iso will be created for you to publish and make available to other rBuilder Online users.

## Application Image Creation Cheat Sheet

### 1. Install rPath Linux

### 2. Create the project on http://www.rpath.org

<projectname> here refers to the Project Name you selected for your project (the part that shows up in the URL), not the Project Title. <user> refers to your username on www.rpath.org

### 3. Create the build environment for the group that defines your distribution.

```
[user@localhost ~]$ mkdir conary
[user@localhost ~]$ cd conary
[user@localhost conary]$ mkdir <projectname>
[user@localhost conary]$ cd <projectname>
[user@localhost <projectname>]$ mkdir builds cache src
[user@localhost <projectname>]$ cd src
[user@localhost src]$ pwd
/home/nathan/conary/<projectname>/src
[user@localhost src]$ gedit conaryrc
```

Now set up the repository map to point to your project

### 4. Create group package

```
[user@localhost src]$ cvc newpkg group-<projectname>
[user@localhost src]$ ls
conaryrc  group-<projectname>
[user@localhost src]$ cd group-<projectname>
[user@localhost group-<projectname>]$ ln -s ../conaryrc conaryrc
[user@localhost group-<projectname>]$ gedit group-<projectname>.recipe
```

Now build your group recipe

### 5. Cook your group and check it in

```
[user@localhost group-<projectname>]$ cvc cook group-<projectname>.recipe
[user@localhost group-<projectname>]$ cvc add group-<projectname>.recipe
[user@localhost group-<projectname>]$ cvc commit
[user@localhost group-<projectname>]$ cvc cook group-<projectname>
```

### 6. Build a release

Use the rBuilder Online interface to generate an image and release it!

**About rPath**

rPath provides technology and services for creating and maintaining Linux software appliances

Contact rPath

phone: 919-851-3984
email: info@rpath.com
http://www.rpath.com

rPath, rPath Linux, Conary, and rBuilder are registered trademarks of rPath, Inc. Copyright 2005, rPath, Inc