Application to Appliance

A Hands-on Guide

Application to Appliance: A Hands-on Guide

rPath, Inc.
701 Corporate Center Drive, Suite 450
Raleigh, North Carolina 27607
USA

Table of Contents

Chapter 1. Introduction

This guide is intended for audiences who have some software development or server setup experience, but who are new to virutal appliances or rPath's technologies such as rBuilder. This guide provides instructions for creating an rBuilder project, establishing an environment for developing appliances, developing a basic appliance, and generating appliance images to deploy in virtual environments. Developers can also use these steps to create software appliances for hardware-based installs. This guide cites additional resources and links to the rPath Wiki (wiki.rpath.com [http://wiki.rpath.com]) for extended information when you are ready to dive deeper into the appliance development experience.

The most recent product versions associated with this guide are:

- rBuilder Online version 4.0.0
- Application to Appliance Development Image version 1.0
- rMake version 1.0.11
- rPath Appliance Platform version 1.0.7
- rPath Appliance Platform Agent version 2.1.4
- rBuilder Appliance version 3.1.4

Chapter 2. Building Appliances with rBuilder and rMake

rPath created the term **software appliance** to describe a software application combined with just enough operating system (JeOS) for it to run optimally on industry-standard hardware. Software appliances simplify server applications by eliminating the hassles of installation, configuration, and maintenance. A **virtual appliance** is a software appliance optimized for virtual environments such as VMware ESX Server. **rBuilder** and **rMake** form the structure needed to create, release, and maintain software appliances and virtual appliances. Other rPath products extend that structure for particular appliance production needs.

Figure 2.1, "Appliance Development with rMake and rBuilder" shows that rMake is used to build an appliance group from application software and the rPath Appliance Platform. rBuilder uses the appliance group to generate images for software and virtual appliances.

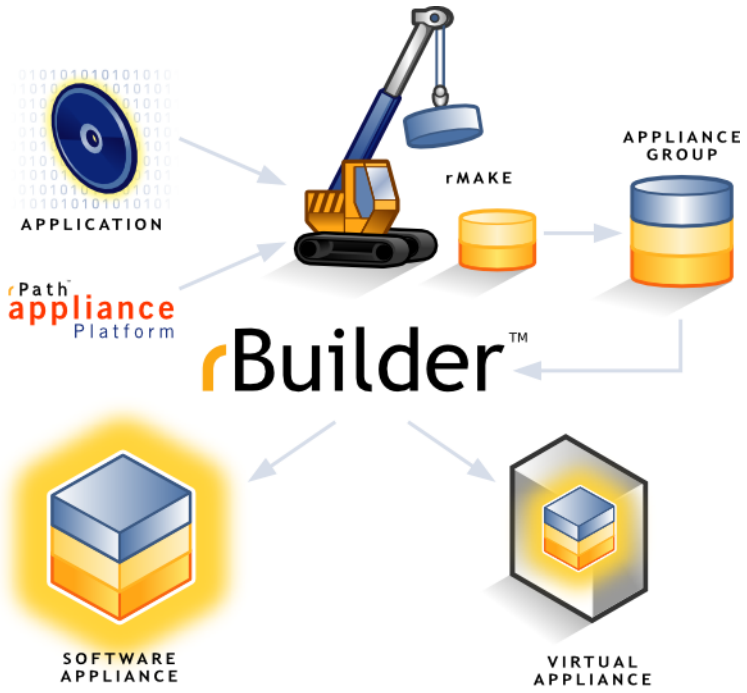

Figure 2.1. Appliance Development with rMake and rBuilder

2.1. rBuilder:The Heart of the Process

rBuilder is the first and only development tool that simplifies and automates the creation of software appliances. rBuilder combines powerful features with innovative packaging techniques to yield a repeatable appliance creation process. Appliance development makes use of rBuilder's project structure to organize appliances and their software, and it employs rBuilder to generate appliance images and to release complete appliance products.

rPath provides rBuilder Online (http://www.rpath.com/rbuilder) as a public rBuilder environment for appliance projects that can be distributed freely to anyone. For corporate appliance production, rPath offers a licensed rBuilder product that can be installed behind a firewall for internal use on a private network.

2.2. rMake: The Tool for Building

rMake is a command-line tool used to build software consistently across varied developers' environments. Appliance developers write a **recipe** to define an appliance, and rMake uses the recipe to pull the software together and build the **appliance group** that rBuilder uses to create appliance images.

rMake uses development tools provided by Conary to perform its build tasks. The **Conary** open-source package management system is a means to package, deliver, and maintain software for Linux. This guide makes use of a virtual development environment in which rMake and Conary are configured and ready to use for appliance development. The guide also cites additional resources you can use to extend your knowledge of rMake and Conary.

2.3. Recipes for Appliance Development Success

Appliance development is centered around **recipes**. A recipe is a text file that provides instructions for building an installable unit of software, how the software should be installed, and whether it should follow any custom policies. rPath provides a set of templates to aid developers in starting new recipes, plus resources for developing those recipes with Conary's application programming interface (API).

Recipes can define an entire appliance or a single piece of software. This guide provides instructions for developing recipes used by rMake to build the software for an appliance.

2.4. The Appliance Group

A **group** ensures that all software installed as part of that group is managed together instead of as separate units. Grouped software can be built, tested, and released together.

An **appliance group** represents everything to be installed on an appliance. rMake follows an **appliance group recipe** to add the application software, the rPath Appliance Platform, and the right amount of JeOS to complete the appliance group. By using an appliance group to build, test, and deploy an appliance, developers can be assued that when users update an appliance, those updates will work as they did during testing.

When the completed appliance group is saved to the rBuilder project, developers can use rBuilder to create multiple images from the appliance group which can be deployed in various physical and virtual environments.

2.5. The Appliance Development Process

This guide steps through the following procedure for developing an appliance:

1. *Set up an rBuilder Project* -- For each appliance, set up a project in rBuilder to represent the appliance.

2. *Configure the Development Context* -- The **context** is a directory location that has been associated with a context entry in a user's "conaryrc" configuration file. rMake uses the context configuration to associate the directory to a particular branch of development in an rBuilder project. This guide requires only a small modification to the example context entry in the configuration file, pointing it to your rBuilder project. You can imitate the context entry when configuring and setting up new contexts.
3. *Write the Appliance Group Recipe* -- Use a preferred text editor to write the recipe that defines what should be included in the appliance. As previously defined, rMake uses this recipe to build the appliance group from which rBuilder creates appliance images.
4. *Build the Appliance Group with rMake* -- Use rMake to build the appliance group recipe "source" into a "binary" appliance group. rBuilder uses this binary group to make one or more different images of the appliance; a single appliance group can generate images for any of several physical and virtual environments.
5. *Commit the Appliance Group to rBuilder* -- rBuilder appliance projects each have a **repository** where the appliance group is maintained for the appliance. Other software can be built and placed into the same repository. The repository is a database that enforces version control for the software. When you **commit**, or "check in," an appliance group to an rBuilder repository, you can then use rBuilder to build images from that appliance group. When you need to modify the appliance group, you can commit a new revision, and you can choose to make images from any of the revisions you have in the repository.
6. *Generate Appliance Images* -- Use rBuilder to create images of the appliance using the appliance group. rBuilder has built-in image types that allow you to generate images that are ready to deploy on any of several environments. This guide demonstrates generating virtual appliances for use in VMware.
7. *Package Application Software* -- A **package** in Conary package management is an installable application that is built especially for installation on Conary-based systems. Appliances generated on rBuilder are based on Conary, and applications to be included on those appliances should be **packaged** for Conary. Like with the appliance group, a package is assembled by writing a recipe and building with rMake. The application software can be packaged from its installable binary files or from its original source code. This guide steps through packaging an application from a binary file, and it demonstrates how rMake automatically locates and includes some supporting software on which the application depends.
8. *Test Packages with Changesets* -- A **changeset** represents the software exactly as it will be installed on an appliance. rMake allows developers to create a changeset from each package build. Developers should test installing the changeset on a basic appliance image to determine whether the package installs and behaves as desired. After testing with changesets, developers can commit an application package to an rBuilder project with confidence that the package will install and work as desired on the appliance.
9. *Build New Packages into the Appliance* -- After packaging application software for the appliance, add lines to the appliance group recipe and build the appliance group again with the updated packages. Commit the new appliance group build to the rBuilder project. This new revision has a dual role: it can be used to generate new images for new appliance installations, and it is automatically used to update previously installed appliances when they check for system updates.

Figure 2.2, "The Appliance Group Recipe and the rMake Group Build" shows that the appliance group recipe defines how rMake should build the appliance as well as how the appliance should install. Each item added to the recipe becomes part of the appliance build when it is assembled by rMake.

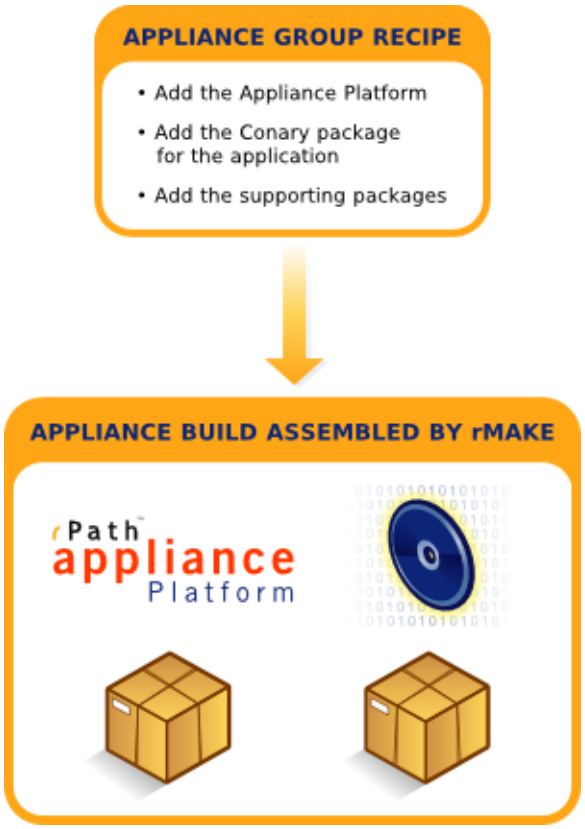

Figure 2.2. The Appliance Group Recipe and the rMake Group Build

10. *Ensure Appliances can Update* -- Use the rPath Appliance Platform Agent web interface to perform an appliance update. This convenient web interface tool is included as part of the rPath Appliance Platform, and it can be customized to the needs of the appliance. Using its System Updates page, click to update the appliance, and ensure the updates you committed to your rBuilder project are successfully deployed to the running appliance. In the appliance model, these one-click appliance updates should be the only requirement for appliance users to keep software up-to-date.

2.6. rPath Appliance Platform: The Essentials for Appliance Management

After distributing an appliance, the underlying technology ensures efficient maintenance. Because appliances are a "black box" concept, administrators should require little or no administrative access to the system, and then only for scheduled maintenance tasks.

To help meet this goal, rPath provides the **rPath Appliance Platform Agent** (rAPA), a web-based administration tool included as part of the rPath Appliance Platform. The agent uses the features of Conary and Linux behind a convenient customizable interface. rAPA is used for such tasks as configuring the appliance, performing system updates, rolling back one or more updates, and backing up and restoring the appliance. Also, the interface can be extended for appliance-specific functions and branded with a vendor-specific look-and-feel.

Continue to the next section to start using these technologies and developing your appliance.

Chapter 3. Set Up an rBuilder Project

The first step in developing your appliance is to set up a project in rBuilder. Though a single project can be used for multiple appliances, rPath recommends a dedicated project for each appliance.

rPath provides *rBuilder Online* [http://www.rpath.com/rbuilder] for free rBuilder services to those working on public appliance projects. You may use any software already available on any rBuilder Online project for your own appliance project. (Be aware of any license constraints for the application software you are using or packaging prior to making that software available on rBuilder Online.)

For convenience and consistency, this guide steps through appliance development for rBuilder Online projects. If you have an rBuilder appliance product for corporate or educational use, you may want to step through this guide first using rBuilder Online, then adapt the procedures to your local rBuilder.

Use the following steps to set up your rBuilder Online account:

1. Navigate to rBuilder Online in your web browser: *http://www.rpath.com/rbuilder*
2. Under the sign-in section, click **Set one up**.
3. Complete the form as instructed in the web interface.
4. Use the confirmation email to confirm your registration.

Log in to rBuilder Online with your new account, and use the following steps to create a new project for your appliance:

1. Click **Create a new project** in your project panel on the right.
2. Type a *Project Name* and *Project Title* in the form. *Note* that the project name will become part of the URL used to access your project and cannot be changed, though the project title can be changed at any time. *Also note* that rBuilder indicates in the interface that there is a limit to the number of characters you may use for the project name.
3. Complete other project details if you want to do so at this time.

Note

Though this guide steps you through creating an example appliance, you may want to use the same rBuilder Online project to create a custom appliance, too. If this is the case, select "Yes" to indicate that your project is a software appliance. This will add your project to a searchable list that highlights appliance projects in rBuilder Online.

4. Click **Create**.
5. Verify the new project is listed in the *Projects I Own* section of your project panel, and click the linked project name at any time to go to your project's main page.

Note the following features of your rBuilder project from the menu on the left side. Click each menu item to follow along with the descriptions:

- *Project Home* -- the main page of the project
- *Manage Builds* -- create and manage installable images used to deploy an appliance; builds are only accessible by project members
- *Manage Releases* -- create and manage releases, each with one or more images, to represent a single appliance product release; releases are accessible by anyone visiting the rBuilder interface
- *Manage Project Membership* -- manage rBuilder users with permission to develop appliances on the project or to act as project owners
- *Group Builder* -- assemble a basic appliance "group" using web-based tools on existing packaged software (ideal for creating a simple appliance quickly from existing components throughout rBuilder, but not recommended for serious appliance development)
- *Browse Repository* -- navigate the contents of your project's repository, which is the central part of the version control system for your appliance software
- *Download Statistics* -- view statistics about the number of downloads from the project over a seven-day period

See the rPath Wiki for more information about using the rBuilder interface, such as the repository browser and search features (http://wiki.rpath.com/wiki/rBuilder).

The next section describes setting up the tools and environment for developing your appliance.

Chapter 4. Set Up an Appliance Development Environment

After you set up your rBuilder project, configure an environment for developing appliances for the project. rPath provides a downloadable image that includes rMake and Conary tools plus a pre-configured development environment that you can point to your rBuilder project.

4.1. Download and Launch the Appliance Development Image

This guide recommends installing *VMware® Player* [http://www.vmware.com/products/player/] which is free and available for most operating systems. (Mac users should also see information about *VMware Fusion* [http://www.vmware.com/mac]). After installing VMware Player, download one of the VMware virtual machine images from the following location. Be sure to download either the 32-bit ("x86") or 64-bit ("x86_64") version as appropriate. If you are not sure which to use, use the "x86 VMware (R) Virtual Appliance" download:

http://www.rpath.org/rbuilder/project/app2app/releases

After downloading the archive file, uncompress it, and then use VMware Player to select and start the image. After the system boots, verify that you are presented with a "login" prompt. Enter the the following credentials to log in as the pre-configured "devuser":

```
username: devuser
password: password
```

Tip

If you are unfamiliar with Linux or using common commands at a Linux or UNIX command prompt, look for additional instructions in any of several online resources for new Linux users. Also, make use of the **man** command to view manual pages on the installed system (such as `man ls`). This guide assumes knowledge of navigating a filesystem at a command prompt, selecting and using a text editor, and using basic Linux commands (`cd`, `pwd`, `mkdir`, `ls`, `rm`, and `more` or `less`).

Warning

You cannot develop appliances as the Linux "root" user. Be sure to use a different user for these activities, but also be sure that you can run commands with root permissions when necessary (by changing to root with `su -` or by configuring the `sudo` utility for running commands with root permissions). For convenience, this guide uses the prompt `$>` to demonstrate commands that all users can execute, and the prompt `#>` for commands that must be executed with root permissions.

4.2. Configure the Context

A **context** is a directory location configured to associate with an rBuilder project. This typically requires two things: a configuration for the context in a `conaryrc` file, and running a setup command in the corresponding directory location. For convenience, the development image you downloaded and launched is mostly pre-configured for your first appliance experience.

The development image has an "example-1-devel" context. The remaining configuration needed is pointing the context to your rBuilder Online project. The configuration is in the text file `.conaryrc` ("dot-conary-r-c") in the developer user's home directory (represented by "~/"), and the corresponding directory is `~/conary/example/example-1-devel`.

Use the following steps to configure the example context for your rBuilder Online project:

1. Identify the repository hostname. rBuilder Online creates this based on the project name value you provided when creating your project. In rBuilder Online, the URL to your project should read *http://www.rpath.org/rbuilder/project/example/* where "example" is the project name value. Your project is available by an abbreviated URL using this value, such as in *example.rpath.org*. This abbreviated URL is called the **repository hostname** when it is used in your development environment. Verify that you can access your rBuilder Online project in a web browser using its repository hostname.

2. Select a text editor to edit the file `~/.conaryrc`. Your development image includes three popular Linux text editors: Vim (Vi IMproved), Emacs, and Nano (an improved version of Pico). As with other Linux commands and utilities, reference online resources when needed for information about a text editor. The following is an example command that opens the file in the Nano text editor:

   ```
   $> nano ~/.conaryrc
   ```

3. Modify `~/.conaryrc` so that it reflects your rBuilder Online user and project information. First, note that the file consists of a **global** section at the top which applies to all appliance projects, and then it includes one or more **context entries** with information about a single development stage of a single project. Figure 4.1, "Context Configuration in the ~/.conaryrc File" shows how the parts of the `~/.conaryrc` apply to multiple directory locations where contexts are set up for appliance development activities.

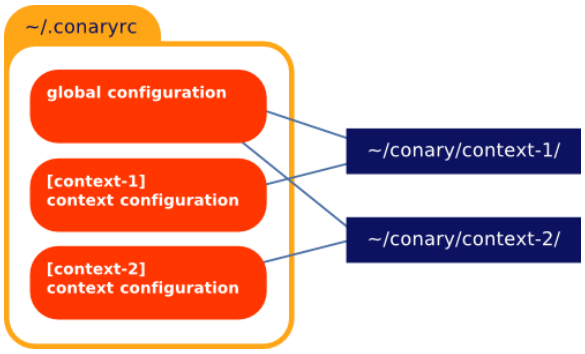

Figure 4.1. Context Configuration in the ~/.conaryrc File

Each context entry starts with its name, and your development image includes a configuration for the "example-1-devel" context. The following shows the text as it should appear in your `~/.conaryrc` file before you make your modifications:

```
                contact          user@example.com
                name             User Name
                user             *.rpath.org rbo_username rbo_password

                [example-1-devel]
                buildLabel       example.rpath.org@corp:example-1-devel

                installLabelPath   example.rpath.org@corp:example-1-devel
conary.rpath.com@rpl:1
```

Use the following steps to modify the file:

a. Change the contact value **`user@example.com`** to an email or Web address, and change the name value **`User Name`** to your name. Because these are in the "global" section of your file, this information is attached to the repository contents in each of your rBuilder projects.

b. Change the user values **`rbo_username`** and **`rbo_password`** to match your user credentials for logging in to rBuilder Online. Note that the string **`*.rpath.org`** indicates this applies to all rBuilder Online projects (or anything ending in ".rpath.org" in the repository hostname). Also note that you can choose to leave out the password, and then enter that password when prompted for each interaction with rBuilder.

c. The section starting with **`[example-1-devel]`** is a **context entry** in your Conary configuration. Leave the name "example-1-devel" as it is, but change each instance of "example.rpath.org" to the repository hostname for your project. As described in a previous step, this is the abbreviated URL obtained by replacing "example" with the project name value that rBuilder Online created for your project.

Note

The string **`example.rpath.org@corp:example-1-devel`** is called a **label**, and it indicates to Conary where repository contents reside. Labels are covered in more detail at the rPath Wiki (http://wiki.rpath.com/wiki/Conary:Labels). Labels are used when implementing rPath's recommended release management structure for appliances, also covered in extensive detail at the rPath Wiki (http://wiki.rpath.com/wiki/Appliance_Development:Release_Management).

d. When working with a local rBuilder instead of rBuilder Online, you will need to add a **`repositoryMap`** line in the context entries for the projects on that rBuilder. Determine what to put on this line by looking through the `conaryrc` values on the rBuilder, accessible through a web browser by accessing *http://rbuilderhost/conaryrc* where "rbuilderhost" is the hostname of your rBuilder. Find the **`repositoryMap`** line that matches your rBuilder project, and add that line to the context entry in `~/.conaryrc`. *This step is not required for your rBuilder Online project.*

4. Save and close the `~/.conaryrc` file after editing. (If you are using Nano, you can use Control-x to exit followed by y and Enter to answer "yes" to whether to save the file.)
5. Change to the `~/conary/example/1-devel` directory that has already been set up for the "example-1-devel" context, and verify that a `CONARY` file resides in that directory. The `CONARY` file is created when setting up a context, and Conary and rMake use that file to maintain some context-specific information:

```
$> cd ~/conary/example/1-devel
$> ls
CONARY
```

6. Confirm the association between the directory and the context by running the `cvc context` command. Verify that the output displayed matches your modified context entry from `~/.conaryrc`:

```
$> cvc context
[example-1-devel]
buildLabel       example.rpath.org@corp:example-1-devel

installLabelPath   example.rpath.org@corp:example-1-devel
conary.rpath.com@rpl:1
```

After stepping through this guide, see the Conary Configuration page at the rPath Wiki for information about adding new context entries in `~/.conaryrc` and setting up new directories to associate with those contexts (http://wiki.rpath.com/wiki/Conary:Configuration).

The next section steps through developing the appliance group.

Chapter 5. Build an Appliance Group

A Conary **group** identifies packages and components (installable portions of packages) intended to be installed and maintained together. An **appliance group** is a group that identifies all the software that should be on an appliance. By using appliance groups, developers ensure that a particular version of software is tested and released with compatible supporting software. Appliance users, then, can update an appliance with confidence that the updates will function properly.

rPath has simplified the creation of an appliance group by automating the search for the appropriate components needed to support your application software.

5.1. Create a New Group

In your "example-1-devel" context, use the `cvc newpkg` command to create a new package or group to be managed by Conary. For a group, Conary requires the group name to start with `group-`. Use the following steps to create `group-example`:

1. Create the new group to be managed by Conary:

```
$> cd ~/conary/example/1-devel/
$> cvc newpkg group-example
```

2. Change to the directory created for the development of the new group:

```
$> cd group-example
$> pwd
/home/user/conary/example/1-devel/group-example
```

5.2. Write an Appliance Group Recipe

To develop the appliance group, write a **recipe** that includes instructions for building the group. rPath provides recipe templates at the rPath Wiki to help with starting package and group recipes (http://wiki.rpath.com/wiki/Conary:Recipe_Templates). Like other Conary code, recipes are written in the Python programming language. However, you do not need to know Python to develop appliances.

rPath provides a Python superclass called `group-appliance` to be used in appliance group recipes. This superclass and the function call `r.addAppliancePlatform()` in a group recipe ensures that the appliance obtains all the operating system components needed to complete the appliance.

Using your preferred text editor, copy the following text and save it as `group-example.recipe` in your group directory. In the "r.setSearchPath" line, be sure to change "example.rpath.org" as appropriate to match your project's repository hostname:

```
# Example Appliance group recipe
loadSuperClass('group-appliance=conary.rpath.com@rpl:1')
```

```
class GroupExample(ApplianceGroupRecipe):
   name = 'group-example'
   version = '1.0'
   autoResolve = True

   def setup (r):
      # Set the labels (repository locations) which Conary
      # should search to find software for the appliance
r.setSearchPath('example.rpath.org@corp:example-1-devel',
                'raa.rpath.org@rpath:raa-2',
                'conary.rpath.com@rpl:1')

      # Add the core operating system components
      r.addAppliancePlatform()

      # Add the rPath Appliance Platform Agent
      r.add('group-raa')
```

Be sure there are four spaces for each indent; DO NOT use tabs for indenting in recipes, and note that Python uses indenting instead of enclosures like braces to determine what is contained within a section of code.

5.3. Use rMake to Build the Appliance Group

In software development, source code is written in a programming language, and it is built in some way to create binary files that can be installed on a computer system. A group recipe is called the **group source** used to build the installable group. In the case of your appliance group, you are building an entire installable appliance.

Software developers who use version-controlled repositories perform a **commit** to save updated versions of source code and binaries into the repository. Conary has built-in version control that uses your rBuilder project repository, and you commit source code (recipes) and binaries (rMake builds) to that repository as part of appliance development.

Figure 5.1, "Recipe Sources and rMake-generated Binary Builds" shows that rMake can be used on a single recipe source to create multiple binary builds. Note that later builds from the same recipe source are usually done when packages change, even if the appliance group recipe stays the same. Source control ensures that when the recipe is modified, the modified recipe is considered a new version of the source which can be used to generate new binary builds.

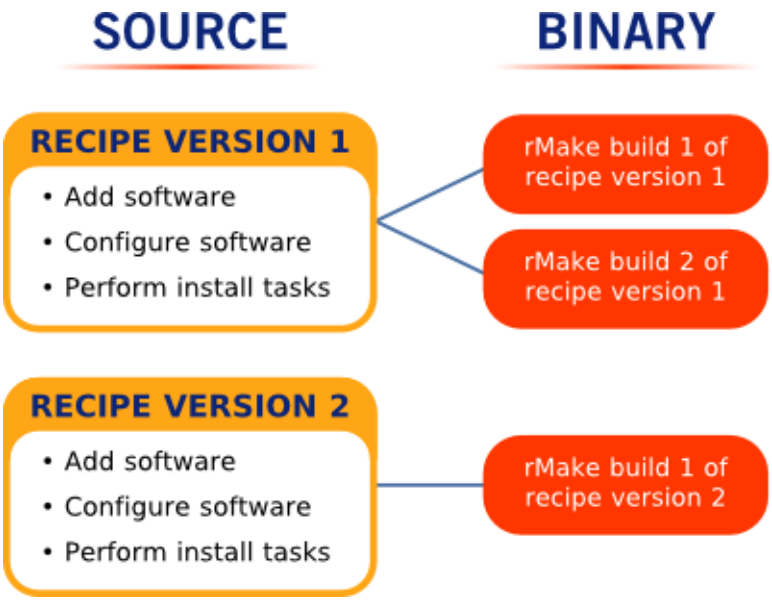

Figure 5.1. Recipe Sources and rMake-generated Binary Builds

Appliance developers can use Conary alone or with rMake to build a group from its group recipe. rPath recommends rMake for a consistent environment across dissimilar development environments. In other words, if three people are working on the project on three different Conary-based systems, rMake ensures that the differences between those systems does not matter, and that the final build is the same no matter where it was built.

Use the following steps to build and commit your group:

1. Build the group recipe locally to verify whether any errors occur, and monitor rMake's output message on your screen as the build progresses. Use the following command:

```
$> rmake build group-example.recipe[vmware]
```

Note

The "[vmware]" part of the command is called a **flavor specification**, which is a means of ensuring the final appliance applies for certain conditions. This particular flavor specification automatically optimizes the image for use in VMware products. Flavors and flavor specifications are explained further at the rPath Wiki (http://wiki.rpath.com/wiki/Conary:Flavors).

2. Identify the **job number** in rMake's output. The job number is repeated throughout as a number in square brackets, such as *[2]*. You will use this number to perform other rMake operations such as reading the log of build messages and committing the build to your project's repository.
3. Verify that rMake finishes the build with the message "build job finished successfully." If any errors occur, check the syntax of your group recipe and try the build command again.
4. After the job finishes successfully, commit the source (recipe) and build to your project repository. This can be done with a single `rmake commit` command. Use the following command, replacing "2" with your rMake job number:

```
$> rmake commit 2 --message="First appliance build."
```

5. Verify that rMake outputs a message indicating that the job was committed.

Note

Appliance developers can rebuild a group from group source (the appliance group recipe) that is already committed to the repository. This is useful when you want to incorporate software updates, but the group recipe itself does not have to change. To rebuild a group at any time using the most recent committed source, execute the command `rmake build group-example` (without the ".recipe") replacing "example" as appropriate for your appliance project.

Bookmark http://wiki.rpath.com/wiki/Conary:Group_Recipe_Classes as a continued reference for other code that you can use in your Conary group recipes.

At this point, you have finished building your first appliance. The appliance does not yet contain its application software, but it does have the rPath Appliance Platform and just enough operating system to support it. You can generate images of this appliance base and use the image to help prepare your application for the appliance. Then, you can finalize the appliance group and rebuild it to include your application.

The next section provides instructions for generating installable images for your appliance. After that, the guide covers packaging the application and completing the appliance.

Chapter 6. Generate Installable Images for an Appliance

When you are ready to create an image from your appliance group build, rBuilder provides several options to serve a variety of deployment needs. Note that rBuilder uses the word "build" to refer to image builds of binary groups, not to be confused with rMake's use of "build" to describe binary builds from source code. In other words, the progression is as shown in Figure 6.1, "Appliance Group, rMake Build, and rBuilder Image Build":

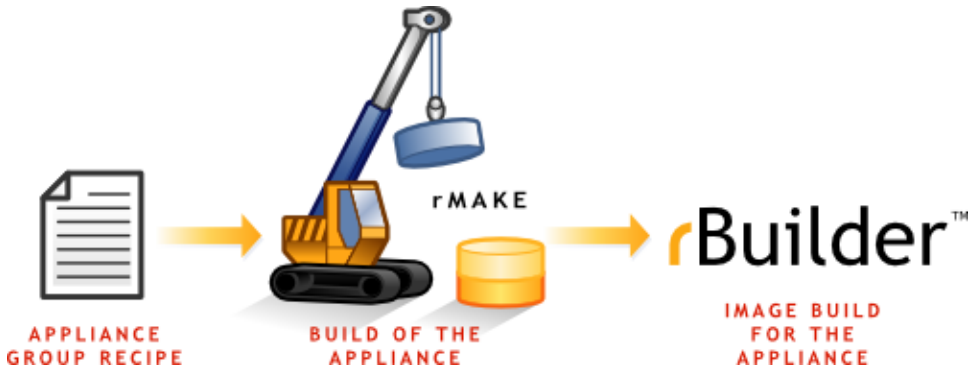

Figure 6.1. Appliance Group, rMake Build, and rBuilder Image Build

Generate images from your *Manage Builds* page in your rBuilder project. Use the following steps to create a new VMware Virtual Machine image, and note that you can create as many images as desired for a single appliance group by repeating these steps and selecting different options each time:

1. Navigate to the *Manage Builds* page for your project in rBuilder, and click **Create a new build**.
2. In the *Distribution Information* section of the form, adjust the name as desired, and type any notes to describe the contents or purpose of the image.
3. In the Build Types section, select the type of image to generate. Use the rBuilder Build Types chart at the rPath Wiki (http://wiki.rpath.com/wiki/rBuilder:Build_Types) to determine which image build types are most appropriate for your deployment scenario, and use links from that chart for additional deployment and Advanced Options information. For your first image, select **VMware Virtual Appliance** so that you can download and launch the image in VMware.
4. Skip the Advanced Options section to accept the default values for those options. If desired, click the section title to expand the section and view all the available options and their default values.
5. In the Build Contents section, select the group that defines your appliance. rBuilder requires you to select the label, revision, and flavor of the group. Currently, you should only have one label, one revision, and one flavor from which to select. The selection process for the example group is as follows:
    a. Click **group-example**.
    b. Click **example.rpath.org@corp:example-1-devel**.
    c. Click **1.0-1-1**. This value is the group's **revision**, part of its Conary version string as described at the rPath Wiki (http://wiki.rpath.com/wiki/Conary:Version_String).

d. Click the string displayed which includes such things as "~!bootstrap," "python," and "vmware." This string represents the group's **flavor**, a term introduced when you built your group with the "vmware" flavor specification. Flavors are described further at the rPath Wiki (http://wiki.rpath.com/wiki/Conary:Flavors).
e. Verify the selected group has a *Back* link followed by a confirmation of the selected group.

6. Click **Create Build**.
7. Verify the browser refreshes to a page dedicated to the new image build. Monitor the status of the image build in the *Build Status* section near the top of the page. When the build is complete, the status should read *Finished*.

Note

If the messages in the box do not change regularly during the process, force a complete refresh of the web page in your browser. In many browsers, you can hold the Shift key while clicking Reload to accomplish this.

Scroll to the bottom of the image build page to view the available downloads. Download the appliance image by clicking its corresponding *Download* link. Note the size of the image displayed below the name of the image.

After downloading, unpack the archive and open the image in VMware. Verify that the system boots in a similar way to your development image.

If you can determine the IP address or hostname assigned to your virtual appliance, you can access the rPath Appliance Platform Agent (rAPA) web interface included in the appliance. rAPA can be accessed using a web browser by browsing to the following URL, replacing "hostname" with the assigned hostname or IP address of the appliance:

```
https://hostname:8003
```

The initial user for rAPA is **`admin`** with a password of **`password`**. Later in this guide are instructions for signing in and using the rAPA interface to perform an update on the appliance.

The next section introduces you to packaging application software for your appliance project and rebuilding the appliance with the new software.

Chapter 7. Package an Application for an Appliance

To add your application to your appliance, you first package it for Conary, and then you add it to your appliance group. Conary packaging prepares software to be installed on Conary-based Linux systems, including appliances created with rMake and rBuilder. Developers can create new software packages either from installable (binary) files or from the application's source code:

- *Binaries* -- Application developers already have ways to compile source code into an installable application for general-purpose operating systems. You can package those installable files (binaries) for Conary so they can be readily added to your software appliance.
- *Source* -- Application developers, especially for open source applications, can package the application source code for Conary and instruct Conary to compile the code as part of building the package. For common languages like C, Conary provides several built-in functions to handle the compiling and compile options.

Bookmark the Conary Packaging page at the rPath Wiki (http://wiki.rpath.com/wiki/Conary:Packaging) as the top-level to Conary packaging reference documentation.

All of your packaging activites can be done in the environment you configured for appliance development. With your context already set up, you can prepare your new application package as described in the next section.

7.1. Develop a New Package

Use the following steps to package a basic application. This exercise includes information for packaging the DokuWiki PHP-based wiki software:

1. Determine whether this application has been packaged and whether its current packaging meets your needs. When software is already packaged, it can be added directly to the appliance group recipe with instructions to obtain the package from its current location. If that package needs specific modifications for use in your appliance, Conary offers two features to help you make those modifications while keeping track of upstream changes: shadows and derived packages. For more information, see Adopting and Adapting in the Appliance Development pages at the rPath Wiki (http://wiki.rpath.com/wiki/Appliance_Development:Adopting_and_Adapting).

   This exercise assumes the software is not yet packaged for Conary.
2. Locate the files for the application. For DokuWiki, the URL for download can be obtained from the application's website. The following URL should work for your example package:

   ```
   http://www.splitbrain.org/_media/projects/dokuwiki/dokuwiki-2007-
   ```

   Packagers with local application files can choose to put those files in the same directory as the package recipe, or they can choose to keep the application files in a consistent network location where rMake can access them when building the package.

3. In the context for your appliance project, create a new package for the application. For the DokuWiki application, use `dokuwiki`, which is used in parts of the download URL. Use the following commands:

```
$> cd /home/user/conary/example/1-devel
$> cvc newpkg dokuwiki
$> cd dokuwiki
$> pwd
/home/user/conary/example/1-devel/dokuwiki
```

4. Using your preferred text editor, create the recipe file appropriate for packaging your application. Recipe development is where you will focus most of your time when packaging your application for Conary. In each case, you will select a recipe template that is appropriate, and then develop the recipe until the package builds successfully.

   Binary files use the Recipe Template for Binary Files [http://wiki.rpath.com/wiki/Conary:Recipe_Template_for_Binary_ part of the Conary Recipe Templates at the rPath Wiki (http://wiki.rpath.com/wiki/Conary:Recipe_Templates). See the Packaging Binaries page for information about developing such a recipe (http://wiki.rpath.com/wiki/Conary:Packaging_Binaries).

   Source files use one of several recipe templates, each designed to address the needs of building from that particular type of source. This includes C, Java, PHP, Python, and software for which you have an RPM package. Packaging from source requires some knowledge of how to troubleshoot the errors particular to compiling that type of code. As with building any type of source code, this type of recipe development is an iterative process of reading error messages and incorporating changes until the package builds successfully. rPath's instructor-led hands-on training course covers scenarios for packaging from source, including troubleshooting builds and using build messages from Conary and rMake.

   Create the recipe for your example package using the following code, which is based on the recipe template for PHP applications:

```
# DokiWiki package recipe from the recipe
# template for PHP applications
loadSuperClass('phpapppackage=contrib.rpath.org@rpl:devel')

class Dokuwiki(PHPAppPackageRecipe):
   name = 'dokuwiki'
   version = '2007_06_26b'

   buildRequires = []

   def unpack(r):
      # Because of Conary conventions, use underscores in
      # place of the hyphens, then create a new macro to
      # convert between them
      r.macros.upstreamVersion = r.macros.version.replace('_','-')
```

```
# Download and install the PHP application
r.extractArchive('http://www.splitbrain.org/_media/projects/'

   '%(name)s/%(name)s-%(upstreamVersion)s.tgz')

# Change the ownership to the "apache" user and group
# to ensure the application has appropriate
# access permissions
r.Ownership('apache', 'apache', '%(approot)s/*.*')
```

When the package builds, rMake completes the archive value using the "name" and "upstreamVersion" macros so that it looks like the original URL you identified for downloading the software.

5. Save and close the new package recipe file and use `rmake build` to build the package from the recipe file. As previously described, this local recipe build displays messages that can be used to troubleshoot the build. Your `dokuwiki` package should not have any errors that require attention. Use the following command to build the package from the local recipe:

```
$> rmake build dokuwiki.recipe
```

6. Prior to committing the package to a repository, rMake provides the ability to create a changeset (".ccs") file from the rMake build. The **changeset** represents all the file additions and modifications necessary to install that build of the package. The changeset file should be installed on a test system that represents your appliance, such as the image you created and launched in Chapter 6, *Generate Installable Images for an Appliance*. Use the following command to create the changeset file for testing the DokiWiki install. Replace the "3" as appropriate with your rMake build:

```
$> rmake changeset 3 dokuwiki dokuwiki.ccs
```

Use `conary showcs` to view the contents of the changeset file, and add the `--ls` option to list the files that are installed when the package is installed, including the permissions and ownership set on each file:

```
$> conary showcs --lsl dokuwiki-2007_06_26b.ccs
```

rPath recommends installing this changeset on a test system, such as the one you created in previous steps, to determine whether the package installs and functions as desired. To make your first packaging experience more efficient, you can skip this otherwise important testing step.

Note

For future packaging activities, use the `conary update` and `conary rollback` commands for installing and rolling back the changeset on the test system. See the Conary QuickReference at the rPath Wiki as a reference of common Conary commands such as these (http://wiki.rpath.com/wiki/Conary:QuickReference).

7. When the changeset installations work as desired, commit the package recipe and the build of the package to the rBuilder project repository. Use the following command, replacing "3" as appropriate for your rMake build:

```
$> rmake commit 3 --message="First package build"
```

Note

View rMake's status using `rmake q` at any time during your appliance development activities. The output indicates builds that completed successfully, that failed, and that have been committed. To review the build messages for a particular build use the job number with the query and add the `--logs` option, such as in the following command:

```
$> rmake q 3 --logs
```

Bookmark http://wiki.rpath.com/wiki/Conary:Package_Recipe_Classes as a continued reference for other code that you can use in your Conary package recipes.

7.2. Add a Package to an Appliance

Add your new package to your appliance by modifying the appliance group recipe and rebuilding the appliance group. Use the following steps to complete this process:

1. Navigate back to your group directory, open the recipe file, and add the following lines to the end of the recipe. INDENT these lines so that they align with the *r.add* line for *group-raa*:

```
# Add Apache and DokuWiki
r.add('httpd')
r.add('dokuwiki')
```

Even though there are other packages named "DokuWiki" in various rBuilder Online projects, your recipe uses your new package because it is on the same **label** (such as `example.rpath.com@corp:example-1-devel`), and you have not indicated for Conary to look elsewhere. As previously stated, labels are described further in the rPath Wiki (http://wiki.rpath.com/wiki/Conary:Labels). To include packages from other rBuilder projects, use Conary labels and other identifying information in an *r.add* line.

Note

Though not explicitly stated in the group recipe, the group will automatically include PHP to support the PHP package.

2. Save the file and rebuild the group using the updated local recipe file. You must specify ".recipe" to instruct rMake to use your local updated recipe instead of the last recipe you committed to your repository. Use the following command:

```
$> rmake build group-example.recipe[vmware]
```

3. When the build is complete, commit the build and updated recipe. rMake should commit the updated source (recipe) at the same time that it commits the build. Use the following commands, changing the job number "4" as appropriate for your rMake build:

```
$> rmake commit 4 -m "Added the DokuWiki package"
```

4. Update an installed version of the appliance to the new build using the rPath Appliance Platform Agent (rAPA) installed on your appliance:
   a. Navigate to the rAPA interface, which can be accessed using a web browser and browsing to the following URL, replacing "hostname" with the assigned hostname or IP address of the appliance:

```
https://hostname:8003/
```

   b. Log in with the default credentials of **`admin`** for the username and **`password`** for the password.

   c. Upon your first access of rAPA on an appliance, unless otherwise configured, you must step through rAPA's initial configuration wizard. Enter information as prompted in required fields; for this appliance guide, it is not important that you have accurate or functional values, but be sure to remember your updated password for admin. When you have completed the wizard, verify that rAPA displays its system summary page and includes a *System Updates* link in the menu on the left.
   d. Click **System Updates**.
   e. Click **Check Now** to check for updates.
   f. After the check is complete, click **OK** to continue, and then click **Apply Now**.
   g. Click **OK** when updates are complete.

   Note that the new build you committed is used by rAPA to update an installed appliance. The same build can also be used to generate new appliances images. When generating new images, select the newer revision number ("1.0-2-1" with 2 representing the second revision of this recipe source, instead of "1.0-1-1").
5. After these steps, the latest version of the appliance includes DokuWiki when it is installed or updated from its previous version. After an update, use the **Manage Services** page in rAPA to restart the Apache (httpd) service.

Note

Post-install service restarts can be added as **group scripts** so they occur automatically when the appliance updates (http://wiki.rpath.com/wiki/Conary:Group_Scripts).)

6. Confirm DokuWiki is working by accessing its configuration screen in a web browser. Use the hostname or IP of the appliance in place of "hostname" in the following URL:

   ```
   http://hostname/dokuwiki/install.php
   ```

   The browser should display a form used to set up the DokuWiki application.

Note

You may continue configuring and using the application software, but you may run into some issues that could be resolved with some additional appliance development. Use rPath's instructor-led hands-on training and the rPath Wiki as references for such extended tasks.

Repeat these steps each time you need to add packages to the appliance group. Use build messages from rMake when necessary to troubleshoot failed builds. Troubleshooting scenarios are covered in rPath's hands-on appliance development training (www.rpath.com/training [http://www.rpath.com/training]).

The next section provides some additional resources for continuing your appliance development journey.

Chapter 8. Resources for Continuing your Appliance Journey

Use the following resources to continue your journey in appliance development with rPath products and technologies:

- *rPath Wiki* -- This guide has cited several resources at the rPath Wiki that can be used when packaging software and building appliances. It also includes other resources for developing appliances, troubleshooting throughout the development process, and customizing the rPath Appliance Platform Agent and other appliance features.

  Use the search box on the left side of each wiki page to search for information from the wiki as well as from the rPath Issue Tracking System, Conary API documentation, rPath corporate website, and other online resources at rPath.

  http://wiki.rpath.com

- *rPath Global Support and Training Program* -- Register for instructor-led hands-on training to deepen your understanding of appliance development and to extend your experience in customizing and releasing an appliance product.

  http://www.rpath.com/training

- *rPath Issue Tracking System* (rITS) -- Use this to report issues with rBuilder, rMake, Conary, and other rPath products. Use the "Quick Search" and "Find Issues" features to see whether someone else has encountered related issues. General rITS users can file issues on "projects" specific to each rPath product, and customers' designated contacts have a Support project to handle support issues.

  http://issues.rpath.com

- *rPath Corporate Home Page* -- Visit the rPath corporate home page for a description of rPath's products, a demonstration of the appliance concept in action, and more information on using rPath's technology to produce your appliance products.

  http://www.rpath.com