# SystemBuilder User Manual

March 10, 2010

**SystemBuilder User Manual**

# Contents

# Chapter 1

# Preface

## 1.1 Audience

SystemBuilder is a product intended for system administrators who need to build, test, deploy and maintain Red Hat Enterprise Linux or CentOS-based systems. Certain chapters in this manual assume you have a particular level of knowledge.

For Chapter 3, "Building a Distribution," in addition to knowing how to create and edit XML files, manage a web server and create USB and CD/DVD boot media, you should have an intermediate skill level with the following components:

- Red Hat Enterprise Linux or CentOS installation, usage, system administration and configuration;
- RPM package usage; and
- YUM usage and configuration.

For Chapter 4, "Creating a Production Environment," experience with version control systems, local mirror hosting, XPath and XInclude.

For Chapter 5, "Doing More with SystemBuilder," key skills include understanding how to automate Anaconda installations using kickstart files and command line arguments.

## 1.2 What's Not Covered

This manual complements several SystemBuilder documents. The "Installation Instructions" and "Getting Started Guide" provide basic information about obtaining and installing SystemBuilder and creating a basic CentOS system distribution. The "Distribution Definition File Reference" (DDFR) contains the syntax for distribution definition elements and attributes. Finally, SystemBuilder has a set of **man pages** explaining the **systembuilder** command, its options and the SystemBuilder configuration file, `systembuilder.conf`. This manual contains references to all four throughout the text.

These SystemBuilder documents are available at the Rendition Software SystemBuilder documentation website.

http://www.renditionsoftware.com/systembuilder/docs/

## 1.3 How to Read

The intent of the manual is to help you use SystemBuilder effectively to build and maintain distributions, and to use distributions to install and update client machines. This manual takes a top-down approach, explaining the concepts before giving steps, commands and code examples that you can use to build, deploy and enhance your own distributions.

## 1.4 Conventions Used

The document conventions are alphabetically listed below.

**Caution** The Caution tag flags issues of importance.

CAUTION

Using the 'dd' command to copy the diskboot.img file onto a USB key will destroy existing data on the device.

**Commands** **Command** names in SystemBuilder or `function` names in other programs appear throughout the document.

**Elements and Attributes** Elements and attributes in XML are referred to by their XPath equivalents. The active node is always considered to be the root, distribution; thus, the first element in the XPath expression is the top-level element, while subsequent elements/attributes are its children. Top-level elements in the DDFR and their child elements use the *parent/child/child* format, while *@attribute-names* are preceded by an @ symbol. The assumed parent element is *distribution*. For more details on XPath syntax, see XML Path Language.

**Examples** Code examples appear in a box, displayed below.

```
<!-- This is a comment in an example code section -->
```

**Filenames** `Filenames`, `foldernames`, `directorytrees` and `fileextensions` all use a monospace format.

**Important** Critical items appear with an "Important" set off tag:

IMPORTANT

The *main* top-level element is required.

**Notes** Within distribution definition examples, the traditional XML comment tags "<!--" and "-->" contain notes. See Also example .

A set-off tag highlights things that a user needs to know.

NOTE

The top-level elements can be in any order in the distribution definition.

**Screen output** Computer output to the screen can be `inline computer output`, usually in Courier monospace font, or included in a box, as in the case below.

```
/usr/bin/systembuilder
```

**User Input** Commands entered by the user appear as `inline commands` or formatted as an example. If the latter, then special characters indicate different users, as shown below.

```
# precedes lines entered as a root user

$ precedes lines entered as a normal user
```

**Variables** SystemBuilder variables are presented in monospace font preceded by a dollar sign, e.g. `$ID`. Variables outside the SystemBuilder realm appear in lowercase in monospace font preceded by a dollar sign, e.g. `$releasever`.

## 1.5 Organization

This manual contains five chapters, each with an increasing level of complexity and detail.

- Chapter I, "Preface," provides information about how to use and read this manual. It also contains information on obtaining the SystemBuilder product.
- Chapter 2, "Introduction" explains what a distribution is, how SystemBuilder helps you build it in an efficient timely manner and the underlying architecture that makes this improved performance possible. The section also includes a "Glossary of Terms" explaining terms used within the SystemBuilder realm.
- Chapter 3, "Building a Distribution," covers the basic concepts for configuring the SystemBuilder build machine, defining and deploying a distribution, running SystemBuilder and updating a distribution. As a use case model, these sections will help you create a basic Samba server distribution based on CentOS 5.
- Chapter 4, "Creating a Production Environment," will help you set up a managed development and deployment environment using SystemBuilder as its cornerstone. It explains implementing the development - staging - production environment within the context of building and deploying distributions using SystemBuilder. The Samba server distribution you created in Part I will have new features requirements to implement in Part II.

  Chapter 4 also discusses how SystemBuilder makes your life easier as a system or build administrator. Versioning distribution files and utilizing an object-oriented model through the XInclude mechanism for the distribution definitions are important usability features. Choosing the best location for storing and downloading a distribution via the web is another key concept.
- Chapter 5, "Doing More with SystemBuilder" will help you customize your distributions. It describes using kickstart files to automate client installation, discusses GPG checking and using GPG keys to sign your own packages, and provides information on creating CD/DVD installation images.

## 1.6 Getting SystemBuilder

SystemBuilder and associated documents are available at the Rendition Software SystemBuilder website at www.renditionsoftware.com/systembuilder.

# Chapter 2

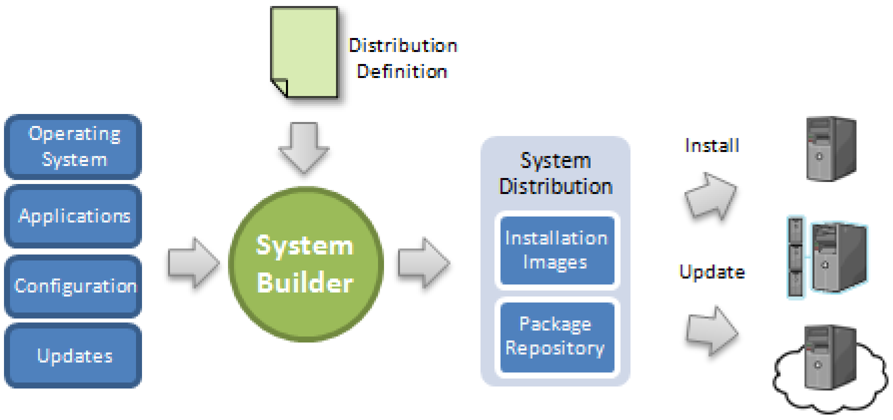
# Introduction

## 2.1 SystemBuilder Overview

Automating customization, installation and maintenance of Red Hat Enterprise Linux and CentOS systems today requires costly and complex management solutions. SystemBuilder provides a simple, standard, reliable alternative. SystemBuilder packages user-specified content into standard operating system deployment bundles, or *distributions*. System distributions can be used to deploy and maintain client machines, physical and virtual, in a wide range of IT environments.

A distribution is a set of operating system components, applications and configuration, packaged for deployment in a standard format. Using SystemBuilder you can create and maintain distributions:

- select an operating system,
- combine it with one or more applications,
- set operating system and application configuration parameters,
- build the distribution,
- deploy it to clients using standard tools and methods,
- change components and configuration,
- rebuild the distribution and
- update existing clients using standard update tools.



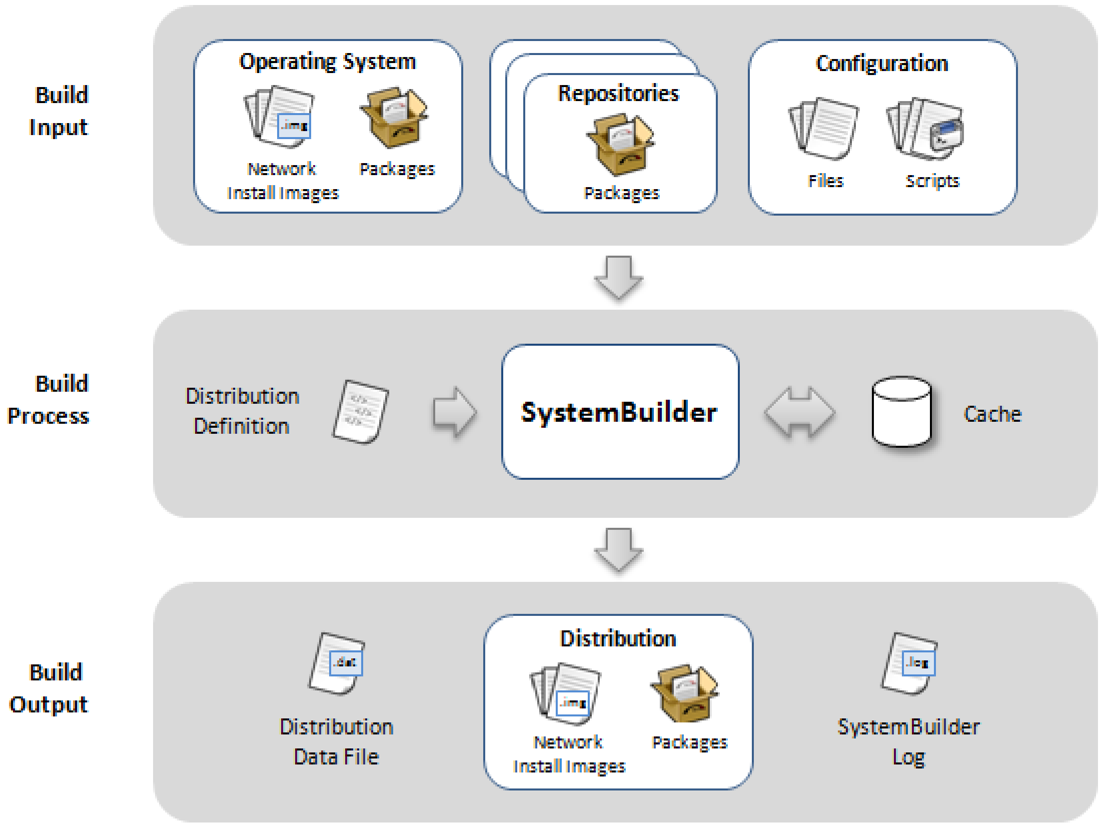
*SystemBuilder Overview*

SystemBuilder reads a distribution definition, gathers various inputs (operating system, applications, and configuration files), and creates a distribution with installation images and a package repository. Using the installation images, you can deploy the distribution to one or more client machines. You can also update existing clients from the distribution package repository.

In summary, SystemBuilder is a production system life cycle manager for use in defining, building and maintaining distributions that can be easily installed and updated on physical, virtual and cloud machines using industry standard deployment and update tools.

## 2.2 Building Distributions

Running SystemBuilder gathers distribution content, processes it, and creates a complete distribution.



*SystemBuilder Build Process*

### Build Input

SystemBuilder uses three classes of input to create distributions: a base operating system, package repositories, and configuration files and scripts. Distribution content is specified using a distribution definition.

- **Base Operating System.** SystemBuilder requires a base operating system distribution as input for creating new distributions. The base operating system is defined at the level of operating system name (Red Hat Enterprise Linux or CentOS), version (5) and architecture (i386 or x86_64). The four base operating systems SystemBuilder supports are RHEL-5-i386, RHEL-5-x86_64, CentOS-5-i386 and CentOS-5-x86_64. Base operating system distributions must include both network installation images and package repositories. Base operating system distribution are covered in more detail under Specifying Repositories.

- **Package Repositories.** SystemBuilder uses RPM package repositories to locate and download software components for the distribution. SystemBuilder requires at least one package repository, that of the base operating system distribution. A typical distribution definition will specify multiple input repositories, including the base operating system update repository, and repositories made available from other internal and external providers. See Specifying Repositories for more information.
- **Configuration Files and Scripts.** The final category of distribution content is user-provided configuration files and scripts. Files are copied, and scripts are executed, during client machine installation and updates. For more information, see Configuring Client Machines.

## Build Process

SystemBuilder uses information from the distribution definition to gather distribution content. It processes this information in three primary ways.

- **Creates a configuration package.** SystemBuilder creates a configuration package in RPM format. This package contains configuration files and scripts specified in the distribution definition. It also contains SystemBuilder-generated YUM update configuration. See Updating Client Machines for more information on SystemBuilder-generated YUM configuration.

  During this step, SystemBuilder also creates a distribution data file, if one does not exist, in the same folder as the distribution definition. The file has a `.dat` extension. SystemBuilder uses the data file to track and increment the configuration package release number.
- **Assembles a package repository.** Using the distribution definition, SystemBuilder compiles a list of required packages. It then resolves package dependencies, downloads packages, adds the configuration package (mentioned in the previous item) and creates a complete, self-contained package repository. The package repository is used during network installations to client machines. It is also used during client machine updates.
- **Customizes network install images.** SystemBuilder downloads network installation images from the base operating system and modifies them to include distribution-specific information. By default, SystemBuilder configures the installer program contained within the images to access distribution content from the distribution's download location. This allows client machines to be installed over the network without needing to provide the download location manually at installation time. SystemBuilder can also be used to completely automate client installations. See Section 5.2 Automating Installations for more information.

SystemBuilder stores distribution content, both input and generated, in the cache folder, by default `/var/cache/systembuilder/`. It reuses information from the cache on subsequent runs, where possible, to improve build performance.

## Build Output

SystemBuilder outputs a complete distribution, a distribution data file and a log file.

- **Distribution.** The primary output of the build process is a complete distribution containing distribution-specific network installation images and a package repository. Using network installation images you can deploy the distribution to one or more physical or virtual client machines. See Installing Client Machines for more information. The distribution package repository supports client machine updates using YUM. The section Updating Client Machines provides an overview of SystemBuilder's client machine update features.
- **Distribution Data File.** As mentioned above, SystemBuilder creates a distribution data file during the build process. This file is used to track and increment the release number of the configuration package.

- **SystemBuilder Log File.** SystemBuilder sends output both to the screen and to a log file, located by default at `/var/log/systembuilder.log`.

In summary, SystemBuilder is a production distribution life cycle manager for use in defining, building and maintaining distributions that can be easily installed and updated on physical and virtual machines using industry standard deployment and update tools.

## 2.3 Installing Client Machines

Because SystemBuilder creates distributions in a standard operating system format, client machines can be deployed using the same tools and methods as used for Red Hat Enterprise Linux (RHEL) and CentOS.

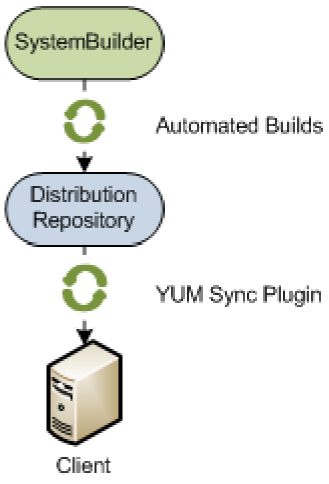
- Network installation using a network interface card (NIC) with Pre-Execution Environment (PXE) support, CD/DVD or USB boot media.
- CD/DVD installation (for use in environments with limited network connectivity).
- Virtual machine installation using the virt-manager or virt-install tools provided with RHEL and CentOS 5.4 operating systems

See the section on Installing Physical and Virtual Clients for complete information.

## 2.4 Updating Client Machines

SystemBuilder makes it easy to keep installed client machines up to date with changes to the distribution. This is done using automated builds and the YUM Sync Plugin.

**Automated Builds.** SystemBuilder runs from the command line with no need for manual intervention. Because of this, SystemBuilder can be automated to refresh distributions on a periodic basis, e.g. nightly, using job scheduling programs such as **cron**. Doing so has two benefits. First, it ensures that your distributions are current and ready to install or reinstall new clients, whenever the need occurs. Second, it allows your distributions to serve as "master repositories", providing ongoing updates to client machines via the YUM Sync Plugin tool. More information on using **cron** to automate SystemBuilder builds is available in Chapter 4, Creating a Production Environment.



*SystemBuilder Update Model*

**YUM Sync Plugin.** The SystemBuilder product provides a small extension to YUM, the *YUM Sync Plugin* that allows keeping installed clients up to date with changes in the distribution or "master" repository. The YUM SyncPlugin is installed to client machines by default via the configuration package. The plugin performs two functions:

- Establishes the distribution repository as the master repository for the client system. This disables all other default package repositories, allowing the plugin to get an accurate list of what packages *should be installed* on the system using packages available in the distribution repository.
- Provides a new command line argument *sync* for use with YUM:

  **# yum sync**

  The *sync* argument behaves similarly to the YUM *update* argument, with two additions: 1) it adds packages from the distribution repository that are not installed on the client, and 2) it removes packages from the client that do not exist in the repository.

See the section on Updating a Client Machine for details and an example of YUM Sync Plugin usage.

## 2.5 SystemBuilder Files and Folders

SystemBuilder has several key files and folders, listed in the table below.

| | |
|---|---|
| /etc/systembuilder/systembuilder.conf | SystemBuilder looks for this file to locate systembuilder configuration. See the systembuilder and systembuilder.conf man pages for more information. This file is optional and does not exist in the default SystemBuilder installation. |
| /usr/bin/systembuilder | The **systembuilder** command. |
| /usr/lib/python`$version`/site-packages/systembuilder | Python code for SystemBuilder, where `$version` is the Python version. |
| /usr/share/doc/systembuilder-`$version` | Documentation for SystemBuilder, where `$version` is the SystemBuilder version. |
| /usr/share/systembuilder/schemas | Relax NG schemas for distribution definition validation. |
| /var/cache/systembuilder/ | SystemBuilder stores cached content in this folder. See the systembuilder and systembuilder.conf man pages for cache-related options. |
| /var/log/systembuilder.log | SystemBuilder's default log file. |
| /var/www/html/distributions/ | SystemBuilder publishes completed distributions to this folder by default. An alternative location can be specified on a per distribution basis within a distribution definition. |

## 2.6 Glossary of Terms

These terms are specific to the SystemBuilder environment, not to Linux or applications that run on Linux. They are arranged in alphabetical order.

**`$ID`** The id for a distribution. The default value consists of the */name*, */version* and */arch* child elements of the *main* element in the distribution definition in the format `$NAME-$VERSION-$ARCH`.

**build machine** The server on which SystemBuilder is installed.

**client machine** The target system on which distribution contents are installed.

**configuration package** An RPM package created by SystemBuilder during the build process. This package contains the YUM sync plugin, SystemBuilder created YUM configuration, and user-specified configuration files and scripts. The name of the configuration package is in the following format, where $NAME and $VERSION are taken from the *name*, and *version* elements in the distribution definition and $RELEASE is taken from the distribution data file.

```
$NAME-config-$VERSION-$RELEASE.noarch.rpm
```

For example, given a distribution with the name "samba-server" and version "5", the first time SystemBuilder creates the distribution, the configuration package will be named as follows:

```
samba-server-config-5-1.noarch.rpm
```

The release number "1" will be written to the distribution data file. When configuration changes in the future, e.g. when files or scripts are added, deleted or modified, SystemBuilder will increment the release number by 1, create a new configuration package (in this case samba-server-config-5-2.noarch.rpm), and store the new release number ("2") in the distribution data file. The incremented version number allows installed client machines to determine that a new configuration package is available for download and installation.

For more information on what the configuration package contains and how to customize its contents, see the section on *config* in the Distribution Definition File Reference.

**DDFR** "Distribution Definition File Reference," the rules of syntax for the distribution definition.

**distribution definition** The information SystemBuilder uses to build, configure and store a distribution. By convention the text file uses the `.distribution` extension. The file uses XML tagging as defined in the DDFR.

**distribution data file** A file SystemBuilder creates to track and increment the release number of the distribution's configuration package. This file is created in the same folder as the distribution definition, using the same filename, with a `.dat` file extension. For example, given the full path to a distribution definition:

```
/~/samba-server.distribution
```

SystemBuilder will create the data file as follows:

```
/~/samba-server.distribution.dat
```

Care should be taken not to delete this file. Doing so will cause the release number of the configuration package to restart at "1". As a result, updated configuration packages will fail to be downloaded and installed during client machine updates.

**download location** The web location (URL) for downloading distribution content. SystemBuilder uses this location during the build process for two purposes, for creating YUM client configuration (included in the configuration package), and for customizing network install images. By default the download location is as follows, where $IP_ADDRESS is the address of the build machine, and $ID is the distribution id.

```
http://$IP_ADDRESS/distributions/$ID
```

For example, if the IP Address for the build machine is `192.168.1.5` and the distribution id is *samba-server-5-i386* the download location is as follows:

```
http://192.168.1.5/distributions/samba-server-5-i386
```

The download location is configurable using the *publish* element of the distribution definition. See the DDFR for information on the publish element. See also Production Definitions for a discussion on using download locations in a production environment.

**distribution repository** RPM package repository created during the distribution build process. Provides packages for use during client machine installations and updates. Can be accessed over the network from the `os` folder of the download location.

**production server** Within a SystemBuilder development - staging - production environment, it serves as the download server from which clients retrieve the distribution contents to install.

**staging server** The server within a development - staging - production environment that uses SystemBuilder to build production-ready distributions for migration to the production, or download, server.

**YUM Sync Plugin** A YUM plugin provided by SystemBuilder that allows synchronizing packages on client machines with packages in the distribution repository. The plugin is included by default in the configuration package, which serves as the mechanism for installing it on client machines. The plugin establishes the distribution repository as the *master repository*, disables all non-master repositories, and provides a new argument, *sync*, to YUM. Help for the *sync* argument is available on client machines using the **yum help** command. Running **yum sync** allows YUM to calculate package additions and removals based on packages available in the distribution repository. For more information, see the section on Updating Client Machines.

# Chapter 3

# Building a Distribution

## 3.1 Configuring a Build Machine

SystemBuilder is designed to work in a networked environment. You can install SystemBuilder on a Linux system operating with CentOS 5 or Red Hat Enterprise Linux 5.

Recent advances in virtualization, including the introduction of a Kernel-based Virtual Machine solution (KVM), have made it both easy and compelling to use virtual machines for testing client installation and updates. For this reason, we recommend installing SystemBuilder on a system installed with the x86_64 version Red Hat Enterprise Linux 5.4 or CentOS 5.4, using one of the following architectures:

- an Intel processor with the Intel VT and the Intel 64 extensions, or
- an AMD processor with the AMD-V and the AMD64 extensions.

This document will focus on using virtual machines a primary method for testing client installations and updates.

For capacity planning purposes, the machine should have a separate logical or physical drive for the `/var` directory. SystemBuilder uses the `/var/cache/systembuilder` folder for caching distribution contents and the `/var/www/html/distributions` folder for storing completed distributions. The recommended minimum drive space allocated for each of these folders is 15GB (30GB total). Depending upon the number and size of distributions created, you may need more disk space.

Disk access is another consideration. Due to the size of the files handled, fast disk access is preferable. To accommodate the speed requirements, it is best to use hard disk drives with Serial Advanced Technology Attachment (SATA) interfaces.

SystemBuilder makes it easy to test client machine installations and updates over the network directly from the build machine. To make use of this feature, you will need to install and configure a web server.

First, install and configure the Apache web server by completing the following steps as the root user.

```
# yum install httpd
# service httpd start
# chkconfig --level 3 httpd on
```

Next, modify your firewall configuration, if needed, to allow www/http traffic (port 80) using the command below as the root user.

```
# system-config-securitylevel
```

To install and configure support for testing distributions using KVM-based virtual machines (recommended), complete these additional steps as the root user. Note the OS and system requirements specified earlier in this section.

First, install the kvm package group.

```
# yum groupinstall kvm
```

Next, using a text editor, modify `/etc/libvirt/qemu/networks/default.xml` to add <forward mode='nat'/> above <bridge name='virbr0'/> as follows:

```
====
<forward mode='nat'/>
<bridge name='virbr0'/>
=====
```

Finally, restart your system.

```
# shutdown -r now
```

Now you are ready to begin defining your distribution!

## 3.2 Choosing a Use Case: A Samba Server

For the purpose of this user guide, examples, processes and illustrations will focus on defining, building, deploying and updating a Samba server based on CentOS 5. Samba is a project of the Software Freedom Conservancy. Samba is a key interoperability component in a blended operating system environment. When installed on a Linux or Unix operating system, it enables that system to act like a Windows file or print server, enabling Windows users to print to or save files using it.

The discussions and code examples in this chapter and subsequent parts focus on a Samba server based on CentOS 5 that enables print and file services. In this chapter, the examples assume the server will be installed via the network directly from the build machine. The DDFR provides the syntax for all parent and child elements and their attributes.

Creating the Samba server distribution entails several process steps. After configuring the SystemBuilder build machine, as described in the previous chapter, you need to create a distribution distribution definition (`samba-server.distribution`). Then you run **systembuilder** to validate the file, retrieve the packages and build the distribution. From there you prepare the boot media.

The next section will explain how to define the system before you build it and prepare for deployment to a client system.

## 3.3 Defining a Distribution

To define your Samba distribution, you will need to do the following:

- Define some basic system information;
- Select the repositories from which to download the operating system, the Samba RPMs and any dependency files;
- Define the application groups and packages to include with the operating system; and
- Provide configuration information.

### Creating a distribution definition

The first step to creating a distribution is creating the distribution definition. The distribution definition is a text file in XML format. It can be created using any text editor - vi, emacs, nano, or gedit - or a dedicated XML editor, if you prefer. The tags <distribution> and </distribution> encapsulate all additional elements and attributes.

```
<?xml version="1.0" encoding="utf-8"?>
<distribution xmlns:xi="http://www.w3.org/2001/XInclude">

<!-- Additional elements and attributes -->

</distribution>
```

The first line, beginning in "<?xml", identifies the file as an XML file. This line is not required, but is a good practice for compatibility with future XML format changes.

Likewise, the "xmlns:xi..." attribute is not required on the distribution element. It allows the use of xinclude statements within the document. XIncludes, a convenient mechanism for reusing content across multiple definitions, are discussed in Chapter 4, "Creating a Production Environment".

### Naming a distribution

The first required element of the definition, *main*, includes the name of the distribution, the system architecture and the version. These three items make up the default distribution id (`$ID`), which SystemBuilder uses to create folders for storing the completed distribution (under `/var/www/html/distributions`, by default) and for caching distribution content (under `/var/cache/systembuilder`, by default).

NOTE

The elements can be in any order within a distribution definition file.

In the example below, the distribution version will be 5, the system architecture will be i386, and the `$ID` will be `samba-server-5-i386`.

```
<main>
  <name>samba-server</name>
  <version>5</version>
  <arch>i386</arch>
</main>
```

You can override the default distribution id using an *id* element. See the DDFR for more details about the *main* element.

Save the file as `samba-server.distribution` before preceding.

NOTE

Each time you change a distribution's id, either implicitly by changing the *name*, *version* or *arch* elements, or explicitly by changing the *id* element, SystemBuilder creates new folders for it in the publish and cache folders (*/var/www/html/distributions* and */var/cache/system-builder*, by default, respectively). To preserve hard drive space, you may wish to review the publish and cache folders on occasion. Folders for obsolete distributions can be deleted using standard file system methods (e.g. **rm -rf *obsolete_folder***).

## Specifying repositories

SystemBuilder obtains software from RPM package repositories. Repositories are collections of RPMs with associated metadata files. Metadata files allow applications, like SystemBuilder, to learn about packages in the repository, without having to download and inspect each package individually. Access to package repositories may be either public or restricted. Common public repositories include CentOS 5 Base, CentOS 5 Updates, and Extra Packages for Enterprise Linux (EPEL). Access to Red Hat Enterprise Linux 5 repositories is restricted. Administrators can create custom package repositories using the createrepo utility.

In this section, you will define the repositories to be used for your distribution.

- Operating System Base Repository. For CentOS, this will be a public or local mirror. For Red Hat Enterprise Linux, an install tree created using an ISO image file. This repository is unique in that it also contains `isolinux` and `images` folders. Your definition must include an operating system base repository.
- Operating System Update Repositories. For CentOS, again this will be a public or local mirror. For Red Hat Enterprise Network, a channel within Red Hat Network. Typically, your definition will include both base and updates repositories.
- Additional Repositories. Your distribution may include any number of additional public or private repositories.

IMPORTANT

You must include a repository definition for the base operating system. This repository must contain `isolinux` and `images` folders.

### CentOS repositories

This example directs SystemBuilder to use public mirrors to access the CentOS 5 base and updates repositories for the i386 architecture.

```
<repos>

  <repo id="base">
    <name>CentOS-5-Base</name>
    <baseurl>http://mirror.centos.org/centos/5/os/i386/</baseurl>
  </repo>
  <repo id="updates">
    <name>CentOS-5-Updates</name>
    <baseurl>http://mirror.centos.org/centos/5/updates/i386/</baseurl>
  </repo>

</repos>
```

**Red Hat Enterprise Linux repositories**

Prior to using SystemBuilder to access Red Hat Enterprise Linux repositories, you will need to complete the following steps:

- Create a Red Hat Enterprise Linux 5 install tree. This can be accomplished by downloading an ISO image (a single DVD image is recommended for easier mounting) from Red Hat Network and using the **mount -o loop** command to mount the `.iso` to a folder location. See the *Red Hat Enterprise Linux 5 Installation Guide*, under the topic *Preparing for a Network Installation*, for more information.
- Create a systemid for accessing updates from RHN. This step requires an available subscription to Red Hat Enterprise Linux. See the Red Hat web site for purchase information. Once you have an available subscription, install Red Hat Enterprise Linux 5 in the desired architecture version (i386 or x86_64) onto a client system. Register the system to Red Hat Network during installation, or first boot, or later using the **rhn_register** command. Once the system has successfully registered with RHN, a systemid file will be created in the `/etc/sysconfig/` folder of the machine. This is the file you will use to configure SystemBuilder to access software from RHN. You may copy this file to the SystemBuilder build machine if necessary. Note, to ensure the systemid has access to the software desired for your distribution, you can use the Red Hat Network web user interface to locate the system, and view and alter it's channel subscriptions.

After the above steps are completed, you are ready to specify base and updates repositories for Red Hat Enterprise Linux.

The base repository will point to the Red Hat Enterprise Linux install tree, created in the first step above. In the example below, this is located at `http://www.examplecompany.com/install_server/redhat/enterprise/5/en/os/i386/`. This location must contain `isolinux` and `images` folders.

The update repository will point to the subscribed channel in RHN using a custom `rhn://` protocol in the following format:

**rhn://[$rhnhost]/$rhnchannel/**

where $rhnhost is the hostname of the server to contact and $rhnchannel is the name of the RHN channel from which to obtain packages. If not specified, $rhnhost defaults to a public Red Hat Network server at xmlrpc.rhn.redhat.com. Valid $rhnchannel values include:

- rhel-i386-server-5
- rhel-x86_64-server-5

In the following example, SystemBuilder will contact the RHN server located at `xmlrpc.rhn.examplecompany.com` and request packages from the `rhel-i386-server-5` software channel.

```
<repos>

  <repo id="base">
    <name>Red Hat Enterprise Linux - Base</name>
    <baseurl>http://www.examplecompany.com/install_server/redhat/enterprise/
            5/en/os/i386</baseurl>
  </repo>
  <repo id="updates">
    <name>Red Hat Enterprise Linux - Updates</name>
    <baseurl>rhn://xmlrpc.rhn.examplecompany.com/rhel-i386-server-5</baseurl>
    <systemid>systemid-5Server-i386</systemid>
  </repo>

</repos>
```

Now that you've identified package repositories, you are ready to start including packages.

## Specifying packages

Once you have defined your repositories, you specify the software packages and groups to include. Use the *packages* top-level element to accomplish this.

The example below includes package groups for the Samba application and related tools plus an X Window System.

```
<packages>
  <group>core</group>
  <group>base</group>
  <group>base-x</group>
  <group>gnome-desktop</group>
  <group>admin-tools</group>
  <group>smb-server</group>
</packages>
```

The distribution will include the GNOME graphical user interface (GUI) and common system administration tools.

## Configuring client machines

The final step for your samba-server definition is to specify configuration for client machines. SystemBuilder allows you to deploy and update client machine configuration in the same manner as you deploy and update individual software components, using an RPM package. This simplifies your tasks as a system administrator by allowing you to manage software and configuration using uniform tools and processes. Creating RPM packages manually can be cumbersome and time consuming. SystemBuilder makes it easy by creating the configuration package automatically during the build process, based on information you provide in the distribution definition.

Using the *config* element, you can complete the following:

- Configure YUM.
- Install files and folders, either as new files or to replace files installed by other packages.
- Run scripts that perform actions when RPM packages are installed or uninstalled.
- Require packages to be installed prior to installing the configuration package. This prevents your files from being overwritten, and ensures that your scripts operate as expected.

### Configuring updates

SystemBuilder creates a configuration package by default, even if no *config* element is specified in the definition. In this case, the configuration package performs just two actions:

- Installs a plugin module for YUM that allows synchronizing packages on the client machine with packages in the distribution repository. See Updating Client Machines for more information on the YUM Sync Plugin.
- Installs a repository definition for the distribution repository. Use the **man yum.conf** command for more information on YUM repository definitions. For the samba-server distribution, the file will be installed at `/etc/yum.repos.d/samba-server.repo`.

The sync plugin can be disabled on individual client machines using the `/etc/yum/yumplugin.d/sync.conf` file. The sync plugin can be excluded from the configuration package altogether using the *config/updates* element. See the DDFR for more information on YUM update configuration using the */config/updates* element.

For the samba-server example, you will use the default update settings.

### Installing files

The *config* element allows you to specify files to be installed at certain locations on the client system. For the samba-server distribution, you will install two files, a `smb.conf` file for configuring the samba server, and a `yum-sync.sh` cron script file for executing the **yum sync** command, provided by the YUM Sync Plugin, on a daily basis.

First, create a basic smb.conf file containing the following content. For this example we are configuring the samba server to provide anonymous, read-only access to content in the client machine's `/public` folder.

```
[global]
  workgroup = WORKGROUP
  netbios name = SAMBA
  security = share

[public]
  comment = Public
  path = /public
  read only = no
  guest only = yes
```

Save the smb.conf file in the same folder as the `samba-server.distribution.`

Next, add the following section to your samba-server.distribution file:

```
<config>

  <!-- samba configuration -->
  <files destdir="/etc/samba" mode="644">smb.conf</files>

  <!-- yum configuration -->
  <files destname="yum-sync.cron" destdir="/etc/cron.daily" mode="644"
  content="text">
#!/bin/sh
/usr/bin/yum sync
  </files>

</config>
```

The example above includes two *files* elements. The first element directs SystemBuilder to locate a file named smb.conf in the same folder as the distribution definition, download it, include it in the configuration package for installation to the `/etc/samba` folder with the file mode "644". The second element directs SystemBuilder to create a new file named `yum-sync.cron`, containing the text supplied between the <files> and </files> tags, and include it configuration package for installation to the `/etc/cron.daily` folder with the mode "644".

NOTE

SystemBuilder resolves all paths in the definition relative to the definition's folder location.

NOTE

The attribute value *content="text"* provides a convenient way to include short files and scripts without needing to create and manage separate files on disk. Larger files can also be created this way, however, this makes the definition more difficult to read.

### Executing scripts

The *config* element also allows you to run scripts when packages are installed and uninstalled.

For the samba-server example, let's include a script to restart the samba daemon each time the configuration package is installed or updated.

```
<config>

  ...

  <script type="post" content="text">service smb restart</script>

</config>
```

Scripts are discussed in more detail under Using Scripts and Triggers.

### Requiring Packages

Using the *requires* element you can specify packages that must be installed prior to installing the configuration package, for example, so that your scripts will execute as expected.

For the samba-server example, the *samba* package is required to enable the command **service smb restart** used in the *script* element above.

```
<config>
  ...

  <requires>samba</requires>

</config>
```

For more information on child elements of *config*, see the *config* element in the DDFR.

### Validating a definition

Your complete samba-server distribution should now look as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<distribution xmlns:xi="http://www.w3.org/2001/XInclude">

<main>
  <name>samba-server</name>
  <version>5</version>
  <arch>i386</arch>
</main>

<repos>
  <repo id="base">
    <name>CentOS-5-Base</name>
    <baseurl>http://mirror.centos.org/centos/5/os/i386/</baseurl>
  </repo>
  <repo id="updates">
    <name>CentOS-5-Updates</name>
    <baseurl>http://mirror.centos.org/centos/5/updates/i386/</baseurl>
  </repo>
</repos>

<packages>
  <group>core</group>
  <group>base</group>
  <group>base-x</group>
  <group>gnome-desktop</group>
  <group>admin-tools</group>
  <group>smb-server</group>
</packages>

<config>
  <!-- samba configuration -->
  <files destdir="/etc/samba" mode="644">smb.conf</files>
  <script type="post" content="text">service smb restart</script>
  <requires>samba</requires>

  <!-- yum configuration -->
  <files destname="yum-sync.cron" destdir="/etc/cron.daily" mode="644"
  content="text">
#!/bin/sh
/usr/bin/yum sync
  </files>

</config>

</distribution>
```

Once your definition is complete, you can use SystemBuilder to validate it's syntax.

```
#systembuilder --validate-only samba-server.distribution
```

If your definition is valid, SystemBuilder will display the complete definition and exit. Otherwise, it will display error information, allowing you to locate and correct issues.

## 3.4 Building a Distribution

Now that you have defined your distribution, you use the **systembuilder** command to build it. The basic command validates the distribution definition syntax, creates the distribution, and records process results. The default level of screen output includes errors, build status and file download and creation status. SystemBuilder also sends this output to the log file, by default `/var/log/systembuilder.log`. If a critical error occurs, SystemBuilder includes detailed error information in the log file.

The initial build of a distribution can be time consuming, on the order of several minutes to an hour or more. Initial build time depends on a number of variables including the location of package repositories (local or remote), network connection speed, the number of packages included in the distribution, and the amount of content that can be reused from the SystemBuilder shared cache. Subsequent builds complete in much less time, often 1-2 minutes, depending on the level and type of change since the previous build.

To run your build, enter the command below:

```
#systembuilder samba-server.distribution
```

The output for an initial run details all the steps for downloading packages, checking package dependencies and creating images. If there are any errors, SystemBuilder notes those and, in many cases, halts the process. See Appendix A for an example of a successful first-build output.

NOTE

If your distribution uses RHN channels as repositories, SystemBuilder may occasionally report errors such as the following while downloading or verifying repository metadata:

```
[repos] long int too large to convert to int
```

This can happen when SystemBuilder receives invalid data from RHN. (Typically this occurs when data is being replicated across RHN servers). The error may be corrected by running SystemBuilder again, either immediately or after waiting several minutes.

## 3.5 Installing Physical and Virtual Clients

Deployment to a client machine is the next step. Because SystemBuilder creates distributions in a standard operating system format, client machines can be deployed using the same tools and methods as used for Red Hat Enterprise Linux and CentOS.

For the samba-server example, we will discuss two installation methods. In the first method, you will install to a physical machine using a bootable CD or DVD. In the second, you will install to a virtual machine.

The methods discussed below are only two of several possible methods for installing your distribution to physical or virtual client machines. See the *Red Hat Enterprise Linux 5 Installation Guide* and *Red Hat Enterprise Linux 5 Virtualization Guide* for additional methods and complete information.

### Installing to a Physical Machine

In this method, you will use a network install image (`boot.iso`) to create an installation CD or DVD. To start the installation, you will boot the client machine using the CD/DVD. The installer program contained on the CD/DVD will then use the network to download files from the distribution URL to complete the installation. To complete this example, you must have configured the build machine as a web server, as described in Configuring a Build Machine

### Creating boot media

After SystemBuilder completes the build, you will find your distribution in a folder beneath the `/var/www/html/distributions` default directory on the download server. The folder name will be the distribution `$ID`, e.g. `samba-server-5-i386`.

To install to a client machine, you burn a boot image to a separate media. For this example, burn the `boot.iso` image (found in the `/os/images` folder below the `/distributions/samba-server-5-i386` folder, above) to a CD or DVD.

To burn the image, one option is to run the **cdrecord** command from the `/os/images` folder as illustrated below. Note the scanbus number sequence, then record the ISO image onto a CD or DVD and label the media, including the number sequence. The number sequence, e.g. 1,0,0, may differ from machine to machine.

```
# cdrecord -scanbus -dev=ATA
# cdrecord -v -dao dev=ATA:1,0,0 boot.iso
```

### Booting a client machine

Now you are ready to create your client. Place the boot media into the CD/DVD drive of the client machine. Now restart the client, booting from the CD or DVD. The boot process will load an installation program, or *installer* from the CD or DVD into system memory. The installer will complete the installation, retrieving additional content over the network from the download location. When the installation successfully completes, the client system will reboot and display a login prompt. You can check the `sys.log` file for installation and boot process log information.

## Installing to a Virtual Machine

Machine virtualization is a relatively new technology with many possibilities for simplifying system administration. In the samba-server example, we will use machine virtualization to test client installation, quickly and easily, without requiring a separate physical client machine, or the creation of physical boot media media (e.g. a CD/DVD disc as in the prior method). Distribution testing using virtual machines is convenient, even when your target clients are physical machines, as it allows you to work more efficiently during the highly iterative development and testing process. To use the method described below, you must have previously configured the build machine to meet exacting hardware, operating system, and configuration requirements. See the section on Configuring a Build Machine for more information.

Once you have configured, and rebooted, your machine as described under *Configuring a Build Machine*, you are ready to install a new virtual machine using either the **virt-manager** or **virt-install** commands. For best results, complete the process one time using virt-manager. Virt-manager provides a graphical user interface that will aid your understanding of the process and help you to ensure that your machine is configured correctly. Once you are familiar with the process, you may find it more efficient to use the command-line utility, virt-install, to complete additional virtual machine installations.

### Using virt-manager

Virt-manager provides a graphical user interface for creating and managing virtual machines. For the samba-server example, create a new virtual machine using virt-manager as follows:

1. As the root user, start virt-manager by typing "`virt-manager &`" at the command prompt.
2. Click on the row beginning with the name "localhost". This will enable the "New" button in the lower right corner of the dialog. If the new button is not available, try exiting virt-manager, switching to the root user, and restarting.
3. Click the "New" button. This brings up a dialog titled "Create a new virtual machine". Click the "Forward" button.
4. In the "Virtual Machine Name" panel, type `samba-server` as the machine name. Click the "Forward" button.
5. In the "Virtualization Method" panel, ensure that the "Fully Virtualized" option is active and selected. If it is greyed out, your machine may not support kernel virtualization extensions, or it may not be configured correctly. Return to the section on Configuring a Build Machine for hardware and software requirements, and machine configuration instructions. Be sure to reboot the machine after following all instructions. If the "Fully Virtualized" option is active and selected, click the "Forward" button.
6. In the "Installation Method" panel, click the "Network Install Tree" option. Click "Forward".
7. In the "Installation Source" panel, type "`http://$IP/distributions/samba-server-5-i386/os`", where $IP is the IP address of the build machine, locatable by typing `ifconfig eth0` at a command prompt on the build machine. Alternatively, if your build machine is locatable via DNS, you can enter the Fully Qualified Domain Name (FQDN) of the build machine in place of the IP address. Note, however, that you cannot enter a hostname (e.g. "myhost") that is not a FQDN (e.g. "myhost.mycompany.com"), as domain searching is not currently supported when using virtual networks. Doing so will result in an error later in the installation process ("Unable to retrieve http://myhost/distributions/samba-server/os/images/stage2.img"). Click "Forward".
8. In the "Storage" panel, ensure the "File (disk image)" option is selected and the value is set to `/var/lib/libvirt/images/samba-server.img`. In subsequent installations (after deleting an existing virtual machine - see Deleting a virtual machine) virt-manager will attempt to automatically increment the image name, e.g. to `/var/lib/libvirt/images/samba-server-1.img`. To preserve disk space on the build machine, you should either reset the image name to `samba-server.img`, or make it a practice to clean out the /var/lib/libvirt/images folder on a regular basis. Click "Forward".
9. In the remaining panels, accept the default values and continue to click "Forward".
10. Click "Finish" to start the installation. If you get an error "Unable to retrieve http://$IP/distributions/samba-server-5-i386/os/stage2.img" during installation, check that 1) you have typed the url correctly, 2) you have provided an IP address or FQDN in the host portion of the URL and 3) the httpd service is running on your build machine and you have allowed port 80 traffic through the machine's firewall (see Configuring a Build Machine).
11. Once installation completes, choose "Reboot".
12. In the resulting dialog, click "Run" to start the machine and test that it functions as expected.

#### Using virt-install

The **virt-install** utility provides a command-line user interface for installing virtual machines. For the samba-server example, use virt-install to create a new virtual machine as follows. You must be the root user.

```
# virt-install --accelerate --hvm --connect qemu:///system \
  --network network:default \
  --name samba-server --ram=512\
  --file=/var/lib/libvirt/images/samba-server.img \
  --file-size=4 --vnc --location=http://$IP/distributions/samba-server-5-i386/os
```

The variable $IP above indicates the IP address of the build machine, locatable by typing `ipconfig eth0` at the command line.

See the section on virt-manager above for help with related common problems and error messages

**Deleting a virtual machine**

While developing a new distribution, you may wish to reuse the same virtual-machine name for testing multiple iterations of the same distribution. You can do this using **virt-manager** as follows:

- From the virt-manager graphical user interface, click on the line beginning "samba-server".
- If the machine is currently running, choose Shutdown and/or Force Off.
- Then click the "Delete" button in the lower right corner of the dialog.

### Verifying the Installation

One quick test to make sure the install completed correctly, on either a physical or virtual client, is to verify the contents of the `smb.conf` installed on the client machine. It should match the one provided in the distribution definition.

Congratulations! You now have a Samba server installed on a CentOS 5 client machine.

## 3.6 Updating a Distribution

SystemBuilder makes it easy to keep your distribution up to date over time by simply rerunning the **systembuilder** command. When SystemBuilder runs, it looks for changes in build input, or in the distribution definition, and makes adjustments to the published distribution as necessary.

### Running systembuilder to check for repository updates

To include updated packages from the repositories, execute **systembuilder** with the `samba-server.distribution` file. Rerunning **systembuilder** with the same `.distribution` file will dramatically reduce the build time, assuming there are no changes to the file's contents. SystemBuilder uses a cache of the packages and verifies the cache contents against the metadata for the repository and the groups. The screen output below shows the results of a build with no changes to the `samba-server.distribution` file and no updates to the repositories.

```
Reading '/home/systembuilder/samba-server.distribution'
Starting build of 'samba-server-5-i386' at 2009-10-25 18:29:55
Build complete at 2009-10-25 18:30:26
```

Since there were no updates, **systembuilder** completed successfully in less than two minutes. (Results will vary based on factors including network connection speed between the build machine and input repositories, number of packages in the distribution, and build machine hardware configuration.) Only if something had changed would **systembuilder** have rerun the repository updates and provided more detail about this build.

Unless you change the log level or the contents of the distribution, **systembuilder** will append the minimal output above to the log file.

NOTE

SystemBuilder's default behavior is to append all build output to the `systembuilder.log` file in the `/var/log` directory. To redirect the output for a particular build, use the **systembuilder** command

```
# systembuilder --log-file /var/log/$MYLOGFILE
samba-server.distribution
```

where `$MYLOGFILE` is the new name of the log file.

If SystemBuilder finds updated packages, it will refresh the distribution repository to contain just the current set of packages. Specifically, when SystemBuilder determines that packages have changed, it calculates the list of packages needed to satisfy distribution requirements, and then adds and removes packages as necessary to bring the distribution repository into comformance with the list. This behavior allows the YUM Sync Plugin to rely on packages in the distribution repository as a master list of packages to be installed on client machines.

IMPORTANT

If the published download location is the same, **systembuilder** updates the distribution, adding new files and replacing old ones.

### Modifying the distribution definition

If you change the distribution definition, or other build input such as configuration files and scripts, then you need to rerun **systembuilder**. You may redefine your distribution in a plethora of ways. Some of the more common options could include selecting different repositories; including or excluding groups, packages or custom files (configuration files, scripts or triggers); or defining a new destination for the published distribution.

If you change the *main/name*, *main/version* or *main/arch* child elements, then you are creating a new distribution with a new `$ID` and download location matching the new parameters. To avoid confusion between the different distributions, save the new distribution definition with a new name.

To try out distribution updates for the samba-server example, you can make two modifications: remove a group, and modify a configuration file.

First, let's say you have determined you don't need packages from the admin-tools group on this system. You can remove them by removing the corresponding */packages/group* element. Modify the *packages* element to look as follows:

```
<packages>
  <group>core</group>
  <group>base</group>
  <group>base-x</group>
  <group>gnome-desktop</group>
  <group>smb-server</group>
</packages>
```

Save the `samba-server.distribution` file.

Next, presuming you would like to change the name of the workgroup for your samba server, modify the content of the smb.conf file as follows:

```
[global]
  workgroup = MYCOMPANY      # was WORKGROUP
  netbios name = SAMBA
  security = share

[public]
  comment = Public
  path = /public
  read only = no
  guest only = yes
```

Save the smb.conf file

Now run the command **systembuilder samba-server.distribution**. You should see that a new configuration package is generated, and that the distribution is updated.

In the next section you will see how these changes are applied to previously installed client systems.

## 3.7 Updating a Client Machine

If there are changes to the distribution, you will need to update installed client machines using the YUM Sync Plugin. For this example build, log on to the client machine as the root user and execute the following commands:

```
# yum clean metadata
# yum sync
```

The first command, `yum clean metadata`, ensures that YUM is using current repository information. For improved performance, YUM caches repository metadata as a matter of course, with a default cache expiration period of 1.5 hours. While you are building and testing a new distribution, you will want to override this behavior by forcing YUM to clean the metadata cache prior to each execution of the **yum sync** command. See the *metadata_expire* option in **man yum.conf**, as well as **man yum** for more information on YUM's caching and cleaning options.

The second command, `yum sync`, will update all packages on the client machine from the distribution repository, including the system-specific configuration package created during the build process. The *sync* command is provided by the YUM Sync Plugin, which is included in the configuration package by default.

To see what packages changed on the client machine, check the `yum.log` file in the `/var/log/` directory.

For the samba-server example, you should see that a new version of the samba-server-config package is installed, the prior version of the samba-server-config package is removed, and a number packages from the admin-tools group, including system-config-network and system-config-users, are removed. You can verify that the `smb.conf` has been updated by typing **cat /etc/samba/smb.conf** at the command line and ensuring the workgroup is now set to "MYCOMPANY" rather than "WORKGROUP".

## 3.8 Tying it All Together

In summary, there are several steps to creating a successful distribution.

First, you configure the SystemBuilder build machine to accommodate the size and number of distributions you will build and store concurrently. The system should be accessible through your local network. After configuring the hardware, you can install one of the Linux operating system versions on which SystemBuilder runs.

Then you define your distribution in the `samba-server.distribution` file using XML tags defined in the DDFR. Your definition includes several key pieces of information:

- the name of the distribution, its version and architecture;
- the repositories for the operating system and application packages, including base install and update mirror locations;
- the package groups and packages to include and any packages to exclude; and
- system configuration files and scripts.

Once the distribution definition is complete, you log on as the root user and execute the command `systembuilder samba-server.distribution` to build the distribution. Output goes to the screen and the default log file. The first build may take some time, since SystemBuilder needs to do everything from scratch:

- file:///usr/share/doc/HTML/index.html validate the `samba-server.distribution` file,
- locate input repositories,
- create the configuration package,
- resolve package dependencies,
- download packages from input repositories,
- create the distribution repository (and it's metadata),
- create network installation images, and
- copy, or *publish*, distribution content to a web-accessible location.

Deploying client machines requires two steps. First, you create the CD or DVD boot media. And then you install the client machine by inserting the disk into the client machine and later verifying that the contents are correct. Alternatively, you can use virt-manager or virt-install to deploy new virtual machines based on the distribution.

Updating your distribution can involve simply checking for mirror repository updates. Or it can involve changing the distribution definition or configuration files and scripts. Depending upon the changes it encounters, SystemBuilder will update the distribution as necessary. After updating the distribution, you will need to run `yum sync` on client machines to keep them up to date with the current distribution.

In the Samba CentOS 5 Server example presented here, what you have in the end is a distribution for a Samba server on CentOS 5 for the i386 architecture. Your `smb.conf` file provides custom Samba configuration instructions. The packages come from the repositories at http://mirror.centos.org .

Now that you have your definition, you can prepare it for use in a production environment. The next chapter explains this process.

# Chapter 4

# Production Distributions

You can harness SystemBuilder's architecture and features to create a production system development environment with reusable components for the distribution definition, version tracking for that and other configuration files, expedited testing procedures and alternate development and production locations. Using SystemBuilder as part of a long-term deployment and shop management strategy will enable you to be more efficient and provide an distribution development and deployment history that is repeatable and predictable.

In this chapter, you will set up a production process for creating, testing, deploying and updating your Samba server. You will divide your `samba-server.distribution` file into reusable components. You'll also choose appropriate download and storage locations. Then you'll log those files into the versioning system of your choice. The testing process for your distribution definition has several options, which are covered in the last section.

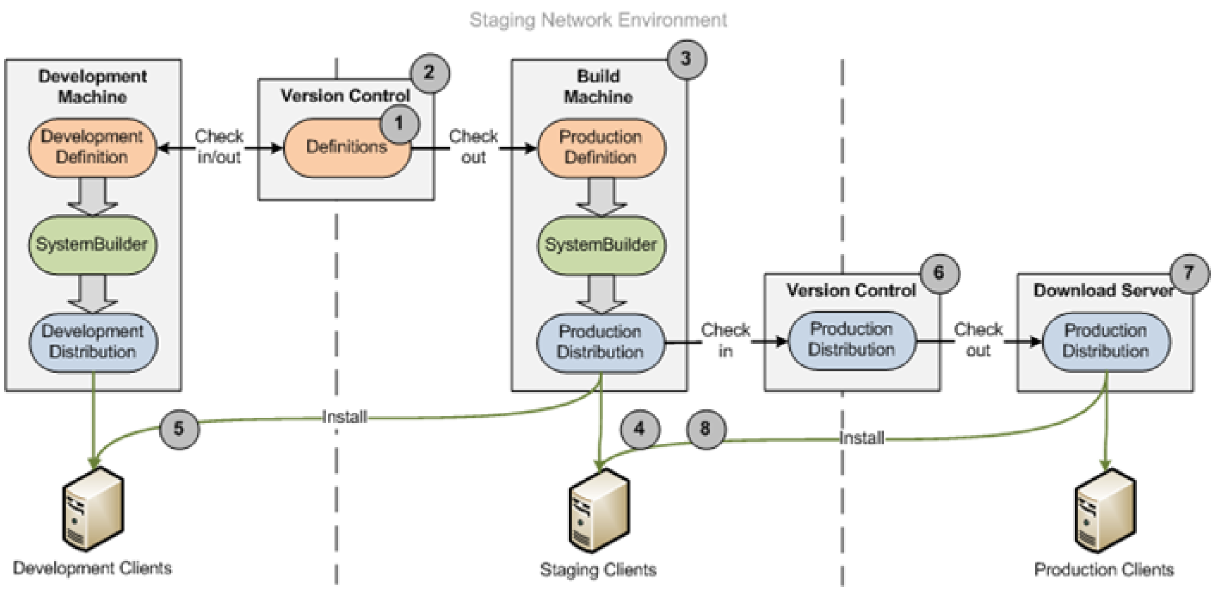
## 4.1 Overview

The traditional system development life cycle (SDLC) includes several stages, such as requirements analysis, system architecture, design and implementation, testing, deployment and maintenance. The stages with which SystemBuilder is an integral part are implementation, testing, deployment and maintenance. Within the SystemBuilder realm, implementation and testing include development, with a development build machine and client machines, which can be virtual machines, for testing installation and updates. A second phase of testing includes staging, which allows you to test the pre-release distribution before releasing it for production. This simulated environment includes completing the build and deployment processes plus using the distribution as intended for a period of time. The final component, production, enables deployment of the distribution to a client system. Maintenance is an on-going process that could involve all of these stages.

The advantage of using a repetitive isolated process as described above is that you can develop, test and fine-tune your distribution before gradually testing its performance in a simulated environment and, finally, releasing it for real-world use. Within the Linux release environment, applications and operating system are updated and upgraded at a variety of rates. This process gives you optimum control over what patches, updates, enhancements and upgrades you choose to maintain and release in your production environment.

### The Production Process Step-by-Step

SystemBuilder's logical network diagram, below, and subsequent description of steps covers the process in more detail.



*SystemBuilder Logical Network*

To set up production environment with SystemBuilder, you will need a Development system (D); a Version Control system (V); a Staging system (S); and a Production system (P). Development and Staging will contain identical installations of a SystemBuilder version. The Production system, which will serve as your download server, needs to be web-accessible for client machines to be able to access it to retrieve distribution content. The Version system needs to have a version control system that supports Linux clients.

In early stages of the process, Steps 1-7, you will work with two versions of your distribution, one for development and one for production. The two versions will be identical with one exception, the download location. This will allow you to test client installation, basic functionality, and updates directly from a development machine prior to moving the distribution to a staging server. In later stages, Steps 5-10, you will work with a single distribution, the production distribution, across separate network environments, staging and production. The staging process allows you to test client interactions with simulated network servers and services prior to deployment. The staging environment is also where you will test ongoing distribution updates from external package repositories.

There are several variables worth noting in the diagram. One is that the hostname for Staging and Production systems is identical; this compatibility will aid in Step 9, migrating the production distribution to a production download server. The hostnames for Development and Version Control are different from the others. Another point about the diagram is that, as mentioned above, the Production system is in a separate network from the Staging system.

**Step 1: Create Development and Production Definitions** During this phase you will work with the Development system (D). You will create two definitions, in our example, `samba-server-development.distribution` and `samba-server-production.distribution`. The definitions will be the same with the exception of the download location, specified using the *publish* element. See the section in this chapter on Production Definitions for more information on creating these files. In addition to creating development and production definitions, you will also create any configuration files necessary for the system to run properly, such as `smb.conf`.

**Step 2: Build a Development Distribution** This step entails building the *development* distribution, specifically by running the command `systembuilder samba-server-development.distribution` . (You will build the *production* distribution in Step 6.)

**Step 3: Test on Development Clients** After building the development distribution, you will install it on development clients and test client functionality. During this step you will also test updates to existing client machines. See Testing Production Updates for more information. Steps 1 through 3 are iterative during the development phase, with you incorporating changes to the definition and configuration files and testing until you are satisfied with the end result.

**Step 4: Check-in Definition Files to Version Control** Once you are satisfied with the results in Step 3, you will set up your Version Control repository (V) and check in definition and configuration files prior to creating the production distribution. From this point forward, if you need to adjust your distribution or configuration files in any way, you should return to Step 1, checking out and altering these files using the Development system. See [?] for information on using a version control system.

**Step 5: Check-out Definition Files to Staging** Now that development is complete, you are ready to build, deploy and test the production distribution on a staging server. Check out the repository with the definition and configuration files from Version Control. If you made several revisions to the files, check out the version you want to deploy.

**Step 6: Build a Production Distribution** Using SystemBuilder on the Staging system (S), build the production distribution, e.g. `systembuilder samba-server-production.distribution .` In addition you should add a **cron** instruction on the Staging system to run SystemBuilder on a regular basis. This will serve to update the distribution over time as new packages are made available in input repositories. See Creating a Staging Machine for more information.

**Step 7: Test on Staging Clients** Next you will deploy and test the distribution on a Staging Client. If the results are unsatisfactory, go back to Development (Step 1), check out the appropriate files from Version Control, make changes as needed, and continue with steps 2 through 6. Use the iterative process in Steps 1-7 until you are satisfied with the results.

**Step 8: Check-in Your Distribution to Version Control** Once you have successfully built, deployed and tested your distribution on the Staging Client, you can check in the distribution *itself* to Version Control. Versioning the distribution allows you to maintain a record of who made changes to the distribution and when. It also audits what changed in each version.

**Step 9: Migrate Your Distribution to Production for Release** Now you are ready to migrate the distribution that is a result of the successful build in Step 6 and that was checked into the Version Control system in Step 8. You will not rerun SystemBuilder on the Production system (P). Rather, you will migrate the distribution to Production and release it for download access. This practice also ensures a controlled release into Production, which improves the lock-down and control process for releasing distributions such as DNS or SMTP servers to the production network environment.

**Step 10: Deploy and Update Your Distribution on Production Clients** As in Chapter 2, the "Deploying Clients" section, you will create the boot media from the Production system and use it to install client machines. In turn, the clients will download the system RPMs from the Production system. For your production environment, you will most likely find it convenient to further automate client installation by setting up a PXE boot environment. See Deploying Production Clients for more information.

Once you have deployed the distribution to production clients, you may discover changes that need to be made. Operating system or applications can have updates, incorporating bug fixes or other planned changes. They can have upgrades or full version releases with new features. Or you may want to add or remove applications or change configuration settings. Depending upon the change needed, the process will differ. Part III will discuss the differences and methods of implementation.

## 4.2 Production Definitions

When working with your distribution through the development - staging - production cycle, you will manage two versions of it, a development version and a production version. The development version, which specifies the Development server as the download location, is used for modifying and testing changes to the definition itself and to configuration files such as smb.conf. The production version is used in both the Staging and Production environments. It specifies a download location using a hostname that is common across Staging and Production network environments.

To create definitions for both versions, you will first create the development definition. Then you will create the production defintion. To keep the two documents synchronized, with the exception of the download-location (specified using the *publish* element), you will use XML Includes (XInclude) to reuse content from the former within the latter.

XIncludes are a convenient feature, in general, for reusing content across multiple definitions. Later in this section we will discuss an additional scenario for using XInclude. We will also cover important limitations in SystemBuilder's XInclude support.

### Create Development and Production Definitions

To create a *development* definition for the samba-server example, begin with the `samba-server.distribution` file you created in Defining a Distribution. First, rename the definition:

```
mv samba-server.distribution samba-server-developer.distribution
```

Then modify the */main/name* element as follows:

```
<main>
  <name>samba-server-development</name>
  ...
</main>
```

The next step is to create a *production* definition. You have three objectives when creating this file. The first is to reuse as much content from the development definition as possible. You will do this using XInclude statements. The second is to change the name of the distribution, which will allow both the development and production distributions to be built on the same physical machine, if desired, without overwriting each other. This objective is accomplished by changing the */main/name* element. The third is to change the download location for the distribution, specifying a fixed URL that is common across Staging and Production environments. You will do this by adding a */publish/remote-url* element. In our example the URL will be `http://download.company.com/distributions`. The completed `samba-server-production.distribution` will look as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<distribution xmlns:xi="http://www.w3.org/2001/XInclude">

<main>
  <name>samba-server</name>
  <!-- reuse all elements from /distribution/main except "name" -->
  <xi:include href="samba-server-development.distribution"
   xpointer="xpointer(/distribution/main/*[name()!=&quot;name&quot;])"/>
</main>

<!-- reuse all elements from /distribution except "main" -->
<xi:include href="samba-server-development.distribution"
 xpointer="xpointer(/distribution/*[name()!=&quot;main&quot;])"/>

<publish>
  <remote-url>http://download.company.com/distributions/</remote-url>
</publish>

</distribution>
```

You can see how the file appears after the xincludes have been processed using the **xmllint** command as follows:

```
xmllint --xinclude --format samba-server-production.distribution
```

Once you have created the files, you can ensure they are valid for use with SystemBuilder using the --validate-only command as follows, where $filename is the name of the file to validate:

```
# systembuilder --validate-only $filename
```

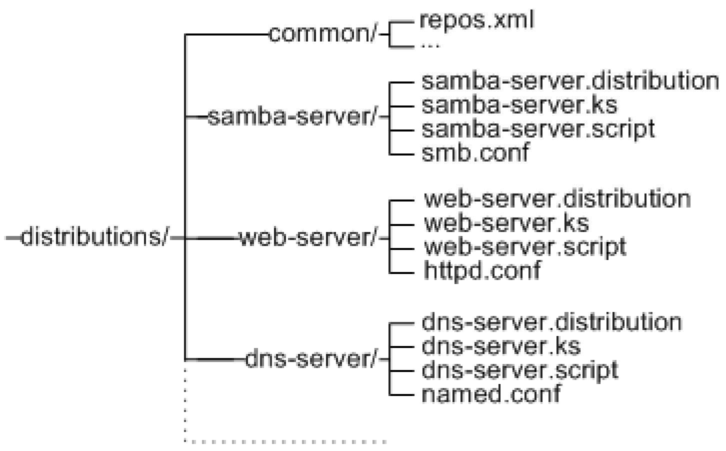
## Strategies for Content Reuse with XInclude

To ease the organization and reuse of information, the software development community commonly uses substitution methods. In the case of SystemBuilder, you can create and store content in one location and use XML Inclusions (XInclude) and XPaths to reuse content across distribution definitions and condense large definition files into more manageable and readable sections using an object-oriented tree. Within that context, some components of a distribution definition are prime candidates for substitution and reuse.

Using XInclude instructions within your definitions has a number of advantages. XInclude processing is recursive. You can have multiple layers of XIncluded documents within one collective distribution definition. By using XPointer you can target one portion of a document. From a time management standpoint, you spend time creating static sections once, then reuse them. If your network topology changes, for example, then you have one file to change, not ten, and the inherent risk of inaccuracies or inconsistencies among the definitions and within your network.

Using XIncludes with SystemBuilder does have one significant limitation, and this has to do with relative path resolution. Specifically, while paths within *http* attributes of *include* elements are resolved relative to the *including document's location*, SystemBuilder resolves all other paths, as noted in Defining a Distribution, relative to the *location of the definition*. Because of this limitation, you will need to exercise care when planning the file system structure for your definitions and shared content.

Within the context of SystemBuilder, a two-part structure on the SystemBuilder build machine will suffice for managing the common versus unique definition components. As in the diagram below, under the subdirectory `distributions/`, the `common/` branch contains reusable components while the other branches each contain files unique to a single distribution.



*Distribution Definition File Reuse*

To properly use the XInclude and XPointer structures, the directory tree must represent the tiered structure above. In other words, the common files to be included must exist at the same tier as any distribution definitions that references them.

In the `distributions/common/` directory above, there are several reusable files. `Repos.xml` contains several repository mirror lists and base URLs from which to download the Linux operating system based upon brand, architecture and whether they are "base" or "updates." The `groups-packages.xml` file contains common application groups and packages that would be included in distributions using any operating system, such as "admin" or "editors."

For the Samba Server, the directory `distributions/samba-server/` in the drawing above contains the `samba-server.distribution` file, the Samba server configuration file, a kickstart file and a validation script. The distribution definition would XInclude components from the repositories, common groups and packages. The same logic holds true for the web and DNS servers.

To see the differences between one large versus several small files, let's take the `samba-server.distribution` file you created in Part I and modify that to use XInclude and XPointer statements. To do that you will create a separate repository file, `repos.xml`.

First, create your `distributions/` subdirectory with the two subdirectories `common/` and `samba-server/` as above. Place your `samba-server.distribution` and `smb.conf` files in the latter.

Next, create the `distributions/common/repos.xml` using the example below. The separate repository file includes the URLs for several operating system brands, architectures and statuses, e.g. "base" and "updates."

```
<?xml version="1.0" encoding="utf-8"?>
<distribution xmlns:xi="http://www.w3.org/2001/XInclude">

<!-- reposistories for centos -->
<repos-centos>
<!-- See www.centos.org for a list of public mirrors -->
    <repo id="base">
      <name>CentOS-$releasever - Base</name>
      <baseurl>http://mirror.centos.org/centos/$releasever/os/
           $basearch/</baseurl>
      <mirrorlist>http://mirrorlist.centos.org/?release=$releasever&amp;
           arch=$basearch&amp;repo=os&amp;country=us</mirrorlist>
    </repo>
    <repo id="updates">
      <name>CentOS-$releasever - Updates</name>
      <baseurl>http://mirror.centos.org/centos/$releasever/updates/
           $basearch/</baseurl>
      <mirrorlist>http://mirrorlist.centos.org/?release=$releasever&amp;
           arch=$basearch&amp;repo=updates&amp;country=us</mirrorlist>
    </repo>

</repos-centos>

</distribution>
```

The final step is to modify the `samba-server.distribution` file, removing the *repos* details and using XInclude and XPointer to reuse common content. You may want to rename the distribution definition or include it in a version control system.

```
<repos>

    <!-- use global repos for CentOS -->
    <xi:include href="../common/repos.xml"
                xpointer="xpointer(/distributions/repos-centos/*)" />

  </repos>
```

To build the distribution definition, you will also have to change the path to the `smb.conf` file in *config/file*.

The result is an object-oriented distribution definition that will reduce your maintenance overhead.

## 4.3 Versioning Definitions

To track and audit any changes to the SystemBuilder files that are critical to the definition, configuration and scripts of a system, use a revision control system, such as Subversion, Git or Mercurial. The Red Hat-based Linux operating system' that SystemBuilder works with by default (RHEL and CentOS) have the revision control system Subversion (**svn**) available by default.

While everything could be revision controlled, there are some key non-image files that should be. Within the SystemBuilder infrastructure and distribution building process, candidates for revision control include configuration files, distribution definitions and scripts, i.e. files that contain critical textual information that may change over time or are commonly used to define aspects of multiple distributions, such as common repositories or network configuration.

- distribution definitions,
- `systembuilder.conf`,
- system configuration files,
- system configuration scripts, and
- distribution definition XInclude files.

If you decide to not use a revision control system, you could rename the distribution definition or other key files every time you make a change to help you "version" your distribution. You can also enter comments in the files. Keep in mind that changing the name of distribution definition does not mean SystemBuilder will build a new distribution. As long as the `$ID` remains the same, the distribution remains the same.

## 4.4 Creating a Staging Machine

## 4.5 Deploying Production Clients

## 4.6 Testing Production Updates

## 4.7 Versioning Distributions

## 4.8 Creating a Download Server

# Chapter 5

# Doing More with SystemBuilder

Now that you have created your basic `samba-server.distribution`, you can customize your system through additional elements in the distribution definition. When selecting packages, you may want certain events to occur depending upon what package version is installed or deinstalled. Scripts that run automatically or are triggered upon certain installation or deinstallation events give you more power over the sequence of events and combination of packages. You can also automate system installation by including a kickstart file. SystemBuilder also accomodates your preferences for media and kernel boot arguments. If your company's legal department or custom software development department requires certain content, you can include those components, as well.

Updating your distribution is easy with SystemBuilder, but you need to understand whether the client machine or just the distribution repository or both will include any updates. The next chapter addresses that critical issue.

## 5.1 Enhancing Your Samba Server

In this part, you will learn how to update your distribution with some common server administration components using two different methods. These updates require changes to the *config* and *packages* elements already in place in your `samba-server.distribution` file.

Following the illustrations in this section, you can conditionally configure applications in your distribution, automate installation, create various install images, add business support required by the legal and software development departments and secure your system. All of these options utilize top-level elements that you include in your system distribution definition, `samba-server.distribution`.

## 5.2 Automating Installations

SystemBuilder can save you time and labor by helping you automate the client boot-up and kickstart process.

### Kickstarting Client Systems

Kickstart installations automate the install process of a RedHat operating system and make the installations reproducible. You may want to use this method if you have large-scale installations, complex storage requirements, multiple network interfaces, secure systems or site-specific strategies. Within the

SystemBuilder process, the kickstart file automates the installation and manages several functions during the install: selection of language, network configuration, distribution source, keyboard, filesystem, mouse, timezone, root password and package handling.

Instead of managing multiple kickstart files, SystemBuilder lets you make it part of the distribution definition. By revision controlling the SystemBuilder files (see Part 2), the kickstart process can be controlled, too.

Use the *kickstart* element in the `samba-server.distribution` file to identify one kickstart file, as in the example below. Substitute your own kickstart filename for `ks.cfg`.

```
<kickstart>ks.cfg</kickstart>
```

See the DDFR for the *kickstart* element syntax.

### Modifying boot.iso and diskboot.img network installation images

To add custom kernel boot arguments to the `/isolinux/isolinux.cfg` file of the `boot.iso` image, use the *bootiso* element in your distribution definition. Substitute your appropriate IP addresses.

```
<bootiso>
  <boot-args use-defaults="false">
  keymap=us lang=en_US ip=192.168.0.222
  netmask=255.255.255.0 gateway=192.168.0.2 dns=192.168.0.7
  </boot-args>
  </bootiso>
```

To create a USB diskboot `IMG` file and possibly add custom boot arguments to the `/syslinux.cfg` file, use the *diskboot-image* element in your `samba-server.distribution` file. The example below disables the default boot arguments "%{ks}," "%{method}," and "nousbstorage."

```
<diskboot-image enabled="true">
  <!-- no args will be appended to syslinux.cfg -->
  <boot-args use-defaults="false" />
  </diskboot-image>
```

See the DDFR for the complete syntaxes of the *bootiso* and *diskboot* elements.

## 5.3 Using Scripts and Triggers

Scripts and triggers enable you to control your client's environment to a great degree. To include these in your distribution, use the *config* element in your `samba-server.distribution` file.

The program that installs RPMs on the client machine actually runs scripts depending upon one of several possible situations. Include these scripts using *config/scripts* in your distribution definition. Commonly referred to by their installation script markers in an RPM spec file, the scripts execute prior to or following installation or uninstallation. See Section 9.4.4 "Defining installation scripts," in the Fedora Project's documentation, RPM Guide by Eric Foster-Johnson. See also Fedora Packaging ScriptletSnippets wiki page and the Maximum RPMbook.

You can also use trigger events as in the RPM spec file. The installation program will run these scripts based upon various combinations of installing, upgrading or removing the target package or your package. See Section 10.2 "Setting Triggers," in the Fedora Project's documentation, RPM Guide by Eric Foster-Johnson for a discussion of the trigger options and their implications. To activate these trigger options using SystemBuilder, include them with the *config/trigger* element of your distribution definition.

In the example below, the */script* element directs the installation program to execute the `smb.script` file after installing the RPM.

```
<!-- start samba daemon at init levels 3 and 5  -->
<script type="post" content="filename">
scripts/smb.script
</script>
```

For the installation to run properly, you will need to create the `smb.script` file under a `scripts` directory and include the contents below.

```
/sbin/service smb start
/sbin/chkconfig --level 35 smb on
```

In the */trigger* element example, below, includes an `smb.sh` script file to run when the program installs the samba package.

```
<!-- run shell script to check if samba-related services are present -->
<trigger package="samba" type="triggerin" content="filename">
scripts/smb.sh
</trigger>
```

You will need to create a sample `smb.sh` file and store it in the relative location. The contents are below.

```
#!/bin/sh
if test -s /etc/samba/smb.conf; then
 cp smb.conf /etc/samba
# could also run an /etc/init.d script
else
 mkdir /etc/samba
 cp smb.conf /etc/samba
fi
```

Include both of these examples in the *config* element of your `samba-server.distribution`.

## 5.4 Verifying Package Integrity

Verifying package integrity using GPG is a secure way of ensuring that you have downloaded the correct set of data; if the key matches the information in the data set, then the set is correct.

### Checking the packages

For a repository that you use in the *repos/repo* element, you can indicate whether GPG checking is enabled by setting */repo/gpgcheck* to TRUE. You must also include at least one GPG key file using */repo/gpgkey*.

In your `samba-server.distribution` file under your *repos/repo* listing for "base" repository, you can add the following code to use the public key provided for CentOS 5.

```
<gpgcheck>yes</gpgcheck>
<gpgkey>http://mirror.centos.org/centos/RPM-GPG-KEY-CentOS-5<gpgkey>
```

### Signing your packages

To authenticate your distribution RPMS, you can also sign them and provide a password in your `samba-server.distribution` file. Use the *gpgsign/secret-key* for exported, armored GPG secret key that signs your distribution and *gpgsign/public-key* to enter the exported, armored GPG public key file that matches the */secret-key*. You may enter a password in plain text form using the */passphrase* element, which is necessary if you want to run SystemBuilder without user interaction. If there is no passphrase, then SystemBuilder will prompt you for the passphrase during the build process.

The example below uses the keys without the passphrase, thus requiring you to be present during the process to enter the passphrase.

```
<gpgsign>
     <public-key>
     GPG/GPG-public.key
     </public-key>
     <secret-key>
     GPG/GPG-secret.key
     </secret-key>
</gpgsign>
```

Since all RedHat repositories have a GPG key for the distributions, you may want to sign and key any new distribution repository as standard practice.

## 5.5 Creating CD/DVD Images

By default SystemBuilder creates `boot.iso` and `diskboot.img` files for CD media. All images come with default sets of kernel boot arguments. But SystemBuilder allows you to do more. You can create multiple sets of release images for CD or DVD or create a USB key-ready image. You can also specify custom kernel boot arguments for each file type. Modify your `samba-server.distribution` file to include what image customization options you want.

To specify custom `ISO` file sizes and media, use the *iso* element and specify which sets you want. You can also add custom kernel boot arguments to the `ISO` files, as well. The default values are "%{ks}" and "method=cdrom." The example below creates several image types.

```
<iso>
<set>DVD</set> <!-- 4700000000 bytes/disc -->
<set>400000000</set> <!-- 400000000 bytes/disc -->
<set>640 mb</set> <!-- 640000000 bytes/disc -->
<set>640 mib</set> <!-- 671088640 bytes/disc -->
</iso>
```

See the DDFR for the complete syntaxes of *iso*, *bootiso* and *diskboot* elements.

## 5.6 Renaming a Distribution

Renaming a distribution brings up several considerations including folder cleanup and existing client migration.

# Chapter 6

# Appendix A

## 6.1 Example SystemBuilder Output

Following is an example of SystemBuilder output for a successful first build.

```
Reading '/~/samba-server.distribution'
Starting build of 'samba-server-5-i386' at 2010-02-09 09:10:54
autoclean
repos
 * downloading repodata - 'base'
   - comps.xml              [=========================]     942 kB (00:06)
   - filelists.sqlite.bz2   [=========================]     3.4 MB (00:10)
   - other.sqlite.bz2       [=========================]     8.3 MB (00:14)
   - primary.sqlite.bz2     [=========================]     1.7 MB (00:03)
   - repomd.xml             [=========================]     2.1 kB (00:00)
 * downloading repodata - 'updates'
   - filelists.sqlite.bz2   [=========================]     2.4 MB (00:19)
   - other.sqlite.bz2       [=========================]     8.1 MB (00:26)
   - primary.sqlite.bz2     [=========================]     474 kB (00:02)
   - repomd.xml             [=========================]     2.0 kB (00:00)
 * verifying repodata file checksums
 * reading available packages
   - base
   - updates
base-info
 * reading buildstamp file from base repository
config
 * downloading files
   - smb.conf               [=========================]     0.0  B (00:00)
 * building samba-server-config-5-1.noarch.rpm
rpmbuild-repo
 * copying packages
 * running createrepo
packages
depsolve
 * verifying repodata file checksums
 * reading available packages
   - base
   - updates
base-info
 * reading buildstamp file from base repository
config
 * downloading files
```

```
   - smb.conf                [==========================]     0.0  B (00:00)
 * building samba-server-config-5-1.noarch.rpm
rpmbuild-repo
 * copying packages
 * running createrepo
packages
depsolve
 * reading package metadata                                            (00:06)
 * checking for package changes                                        (00:01)
 * resolving package dependencies
   - loop 1 (346 packages)   [==========================]    346/346 (00:08)
   - loop 2 (371 packages)   [==========================]    371/371 (00:05)
   - loop 3 (83 packages)    [==========================]      83/83 (00:01)
   - loop 4 (16 packages)    [==========================]      16/16 (00:00)
   - loop 5 (5 packages)     [==========================]        5/5 (00:00)
   - loop 6 (1 package)      [==========================]        1/1 (00:05)
   - loop 7 (2 packages)     [==========================]        2/2 (00:00)
 * pkglist closure achieved in 819 packages
 * writing pkglist
download
 * downloading packages - 'base'
   - Deployment_Guide-en-US- [==========================]     3.6 MB (00:43)
   - GConf2-2.14.0-9.el5.i38 [==========================]     1.5 MB (00:18)
   - MAKEDEV-3.23-1.2.i386.r [==========================]     138 kB (00:02)
   ... [output abbreviated for readability]
   - zenity-2.16.0-2.el5.i38 [==========================]     1.2 MB (00:00)
   - zip-2.31-2.el5.i386.rpm [==========================]     130 kB (00:00)
   - zlib-1.2.3-3.i386.rpm   [==========================]      51 kB (00:00)
 * downloading packages - 'updates'
   - PyXML-0.8.4-4.el5_4.2.i [==========================]     1.1 MB (00:00)
   - acpid-1.0.4-9.el5_4.2.i [==========================]      24 kB (00:00)
   - autofs-5.0.1-0.rc2.131. [==========================]     904 kB (00:00)
   ... [output abbreviated for readability]
   - xterm-215-8.el5_4.1.i38 [==========================]     414 kB (00:00)
   - xulrunner-1.9.0.16-2.el [==========================]      10 MB (00:00)
   - yum-fastestmirror-1.1.1 [==========================]      19 kB (00:00)
 * downloading packages - 'rpmbuild-repo'
   - samba-server-config-5-1 [==========================]     2.8 kB (00:00)
gpgcheck
createrepo
 * downloading files
 * running createrepo                                                  (01:04)
isolinux
 * downloading files
   - boot.msg                [==========================]     292  B (00:00)
   - general.msg             [==========================]     919  B (00:00)
   - initrd.img              [==========================]     6.9 MB (00:00)
   - isolinux.bin            [==========================]      11 kB (00:00)
   - isolinux.cfg            [==========================]     366  B (00:00)
   - memtest                 [==========================]      95 kB (00:00)
   - options.msg             [==========================]     817  B (00:00)
   - param.msg               [==========================]     517  B (00:00)
   - rescue.msg              [==========================]     490  B (00:00)
   - splash.lss              [==========================]      64 kB (00:00)
   - vmlinuz                 [==========================]     1.9 MB (00:00)
discinfo
buildstamp
stage2-images
 * downloading files
   - minstg2.img             [==========================]      41 MB (00:01)
   - stage2.img              [==========================]     112 MB (00:03)
initrd-image
 * downloading files
   - initrd.img              [==========================]     6.9 MB (00:00)
 * modifying initrd.img
xen-images
 * downloading files
   - initrd.img              [==========================]     6.9 MB (00:00)
   - vmlinuz                 [==========================]     2.2 MB (00:00)
 * downloading files
   - initrd.img              [==========================]     6.9 MB (00:00)
 * modifying initrd.img
product-image
 * modifying product.img
bootiso
   - isolinux.cfg            [==========================]     366  B (00:00)
diskboot-image
 * modifying diskboot.img
pxeboot-images
 * downloading files
   - initrd.img              [==========================]     6.9 MB (00:00)
   - vmlinuz                 [==========================]     1.9 MB (00:00)
treeinfo
compose
 * linking files
 * creating manifest file
publish
 * publishing to '/var/www/html/distributions/samba-server-5-i386'
Build complete at 2010-02-09 09:18:58
```