openSUSE-KIWI Image System

Cookbook

Marcus Schäfer

openSUSE-KIWI Image System: Cookbook

by Marcus Schäfer
Thomas Schraitle <toms@suse.com>
Frank Sundermeyer <fs@suse.com>
Robert Schweikert <rjschwei@suse.com>

KIWI Version 7.03

Table of Contents

Part I. Concepts and Basics

Table of Contents

1 Introduction

Table of Contents

1.1. What is KIWI?

KIWI is a command line tool, written in Perl, for building images for Linux. It supports a variety of image formats. KIWI is used as a back-end for the appliance builder SUSE Studio [http://susestudio.com/]. It is also used to build images in the openSUSE Build Service [http://build.opensuse.org/], among them images for all SUSE products.

Images for Linux are available in many different formats. A Linux `*.iso` file, that can be burned to an optical medium to install Linux, is an image. A file used by virtualization systems such as KVM, Xen, or VMware is an image. The installation of a Linux system on your hard drive can be turned into an image using the **dd** command.

Basically, an image is a Linux system in a file. Depending on the type of the image, there are different use cases for it. It can be used to burn an iso image to an optical medium with which the computer can be booted. An image can also be used to run a Virtual Machine from the `*.iso` file (image) stored on your hard drive. KIWI supports the following image types and formats:

- ISO
- Live CD/DVD
- PXEBoot
- Hard Disk
- USB
- Amazon EC2 (`.ami`)
- Docker
- Google Cloud Format (`..gce`)
- KVM/Qemu (`.qcow2`)

- Open Virtualization Format (`.ovf`, `.ova`)
- Vagrant (`.vagrant`
- VirtualBox (`.vdi`)
- Virtual Hard Disk (`.vhd`)
- VMware (`.vmdk`)
- XEN

1.2. What does KIWI do?

KIWI allows you to configure, build, and deploy your own operating system images in a variety of formats. The KIWI workflow is divided into two distinct stages. For a detailed description of this process refer to Chapter 3, *Basic Workflow*.

1. **Preparation.** Create a root directory holding the contents of the new file system. Install the required packages from a software package source such as the installation media for SUSE Linux Enterprise Server, or an online repository. Create an image description file, (`config.xml`) and optionally apply customizations. This operation results in the “unpacked root tree”.
2. **Creation.** The image itself is created using the unpacked root tree created in the previous step. The image creation process does not require user interaction, but can be fine-tuned by modifying the `images.sh` script that is called during the creation process.

1.3. How to use KIWI?

KIWI is a command line tool that is invoked with the **kiwi** command in your shell. KIWI needs to be executed as the `root` user, as administrative privileges are required for many operations that need to take place to create an image. Therefore, when using KIWI you need to be aware of what you are doing and a certain amount of caution is in order. Running KIWI on your system is not inherently dangerous to your system, just keep in mind that you are running as the `root` user.

The two phases of the image creation process outlined in Section 1.2, “What does KIWI do?” can be started with the commands **kiwi --prepare** for the first step and **kiwi --create** for the second step. For convenience KIWI also has the `--build` that combines the *prepare* and *create* steps.

2 Installation

Table of Contents

2.1. Installing KIWI Packages

KIWI is shipped with all SUSE distributions, but not installed by default. With SUSE Linux Enterprise 11, KIWI is included in the software development kit (SDK), not the main installation media. A minimum KIWI installation requires installing the kiwi package and at least one package containing the boot descriptions for the various image types:

kiwi-desc-isoboot: Live ISO boot templates
kiwi-desc-netboot: PXE network boot templates
kiwi-desc-oemboot: Expandable Virtual Machine boot templates
kiwi-desc-vmxboot: Virtual Machine boot templates

It is also recommended to install the package kiwi-doc containing the documentation. For a complete list of KIWI packages run the command **zypper se kiwi**.

2.1.1. Installing the Latest Version Available

KIWI is an active project and new releases are published regularly. Packages with the latest KIWI version for all SUSE distributions that are actively maintained can be obtained from the Virtualization:Appliances repository at http://download.opensuse.org/repositories/Virtualization:/Appliances/.

2.2. Running KIWI from a Source Checkout

KIWI is developed and maintained in a repository on GitHub. You can clone the source code using the following command.

```
git clone https://github.com/openSUSE/kiwi.git
```

Before running KIWI make sure all its dependencies are fullfilled. Get a list of required packages by running the following command in the checkout directory (`kiwi/`):

```
awk '/BuildRequires:/ { print $2 | "sort" }' rpm/kiwi.spec
```

Once all dependent packages are installed run the test suite from the checkout directory as follows

```
make test
```

If all tests pass, all dependencies are fullfilled and KIWI can be run with from the checkout directory with the following command:

```
./kiwi
```

To update to the latest version available, run **git pull** from the KIWI checkout directory.

3 Basic Workflow

Table of Contents

Installation of a Linux system generally occurs by booting the target system from an installation source such as an installation CD/DVD, a live CD/DVD, or a network boot environment (PXE). The installation process is often driven by an installer that interacts with the user to collect information about the installation. This information generally includes the *software to be installed*, the *timezone*, system *user* data, and other information. Once all the information is collected, the installer installs the software onto the target system using packages from the software sources (repositories) available. After the installation is complete the system usually reboots and enters a configuration procedure upon start-up. The configuration may be fully automatic or it may include user interaction.

A system image (usually called “image”), is a *complete installation* of a Linux system within a file. The image represents an operational system and—optionally contains the “final” configuration.

The behavior of the image upon deployment varies depending on the image type and the image configuration since KIWI allows you to completely customize the initial start-up behavior of the image. Among others, this includes images that

- can be deployed inside an existing virtual environment without requiring configuration at start-up.
- automatically configure themselves in a known target environment.
- prompt the user for an interactive system configuration.

The image creation process with KIWI is automated and does not require any user interaction. The information required for the image creation process is provided by the primary configuration file named `config.xml`. In addition, the image can optionally be customized using the `config.sh` and `images.sh` scripts and by using an *overlay tree (directory)* called “root”.

Previous Knowledge

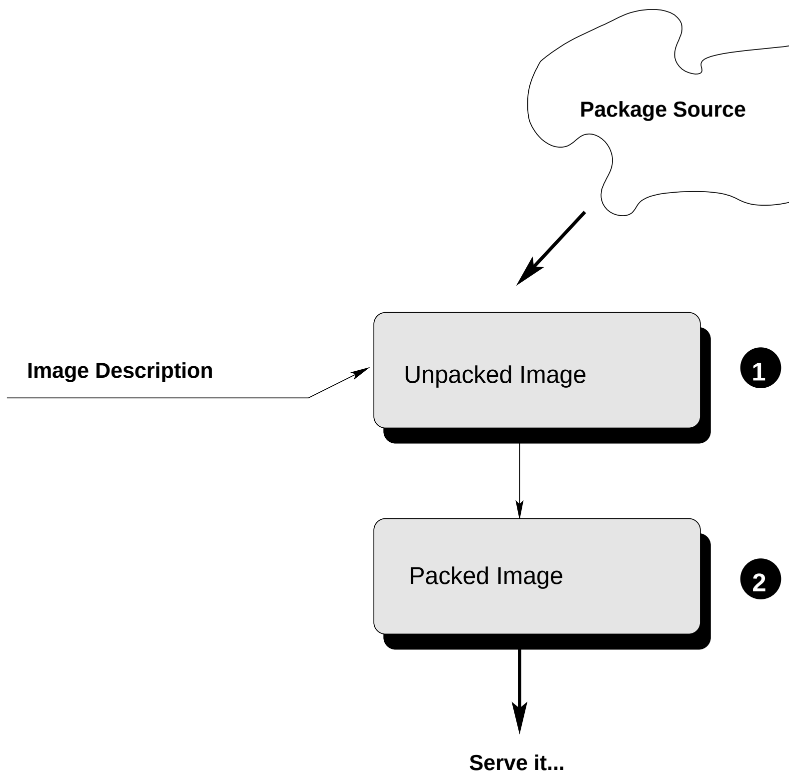
This manual assumes that you are familiar with the general concepts of Linux, including the boot process, and distribution concepts such as package management.

3.1. Building Images

KIWI creates images in a two step process. The first step, the `prepare` operation, generates a so-called *unpacked image* tree (directory) using the information provided in the `config.xml` configuration file. The `config.xml` file is part of the *configuration directory (tree)* that describes the image to be created by KIWI.

The second step, the `create` operation, creates the *packed image* or *image* in the specified format based on the unpacked image and the information provided in the `config.xml` configuration file.

Figure 3.1. Image Creation Architecture



(1) Unpacked Image
Encapsulated system reachable via chroot

(2) Packed Image
Encapsulated system reachable via kernel file system/extension drivers such as loopback mounts, etc.

3.1.1. The Prepare Step

The creation of an image with KIWI is a two step process. The first step is called the `prepare` step and it must complete successfully before the second step, the `create` step can be executed.

During the prepare step, KIWI creates an *unpacked image*, also called “root tree”. The new root tree is created in a directory specified on the command line with the `--root` argument or the value of the `defaultroot` element in the `config.xml` file. This directory will be the installation target for software packages to be installed during the image creation process.

For package installation, KIWI relies on the package manager specified with the `packagemanager` element in the `config.xml` file. KIWI supports the following package managers: `smart`, `zypper` (default), `yum` and `apt`.

The prepare step consists of the following substeps::

1. **Create Target Root Directory.**
   KIWI will exit with an error if the target root tree already exists to prevent accidental deletion of an existing unpacked image. Using the `--force-new-root` command line argument will force kiwi to delete the existing target directory and create a new unpacked image in a new directory with the same name.

2. **Install Packages.**
   Initially KIWI configures the package manager to use the repositories specified in the configuration file and/or the command line. Following the repository setup the packages specified in the `bootstrap` section of the configuration file are installed in a temporary workspace external to the target root tree. This establishes the initial environment, to support the completion of the process in chroot setting. The essential packages to specify as part of the bootstrap environment are the filesystem and glibc-locale packages. The dependency chain of these two packages is sufficient to populate the bootstrap environment with all required software to support the installation of packages into the new root tree.

   The installation of software packages through the selected package manager may install unwanted packages. Removing such packages can be accomplished by marking them for deletion in the configuration file. To do so specify a configuration entry like:

   ```
   <package type="delete">package_to_be_deleted</package>
   ```

3. **Apply The Overlay Tree.**
   After the package installation is complete, KIWI will apply all files and directories present in the overlay directory named *root* to the target root tree. Files already present in the target root directory will be overwritten, others will be added. This allows you to overwrite any file that was installed by one of the packages during the installation phase.

4. **Apply Archives.**
   Any archive specified with the `archive` element in the `config.xml` file is applied in the specified order (top to bottom) after the overlay tree copy operation is complete. Files and directories will be extracted relative to the top level of the new root tree. As with the overlay tree, it is possible to overwrite files already existing in the target root tree.

5. **Execute the User-defined Script `config.sh` .**
   At the end of the preparation stage the script named `config.sh` is executed if present. It is executed on the top level of the target root tree. The script's primary function is to complete the system configuration, for example by activating services. For a detailed description of pre-defined configuration functions consult the kiwi::config.sh(1) man page.

6. **Manage The New Root Tree.**
   The unpacked image directory is a directory, as far as the build system is concerned and you can manipulate the content of this directory according to your needs. Since it represents a system installation you can “chroot” into this directory for testing purposes. The file system contains an additional directory named `/image` that is not present in a regular system. It contains information KIWI requires during the create step, including a copy of the `config.xml` file.

   Do not make any changes to the system, since they will get lost when re-running the `prepare` step again. Whats more, you may introduce errors that will occur during the `create` step, that are difficult to track. The recommended way to apply changes to the unpacked image directory is to change the configuration and re-run the `prepare` step.

3.1.2. The Create Step

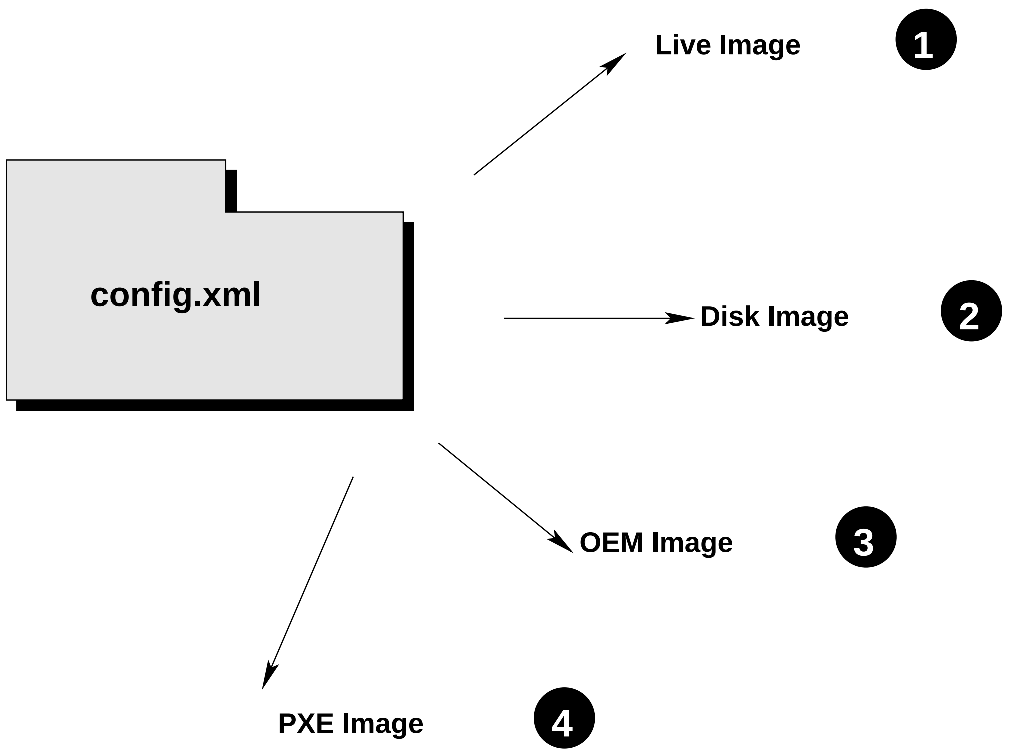
The successful completion of the `prepare` step is a prerequisite for the `create` step. It ensures the unpacked root tree is complete and consistent. Creating the packed, or final, image is done in the `create` step. Multiple images can be created using the same unpacked root tree. It is, for example, possible to create a self installing OEM image and a virtual machine image from a single unpacked root tree. The only prerequisite is that both image types are specified in the `config.xml` before the prepare step is executed.

During the `create` step the following major operations are performed by kiwi:

1. **Execute the User-defined Script `images.sh` .**
   At the beginning of the image creation process the script named `images.sh` is executed if present. It is executed on the top level of the target root tree. The script is usually used to remove files that are no needed in the final image. For example, if an appliance is being built for a specific hardware, unnecessary kernel drivers can be removed using this script. Consult the kiwi::images.sh(1) man page for a detailed description of pre-defined functions available in the `images.sh` script.

2. **Create Requested Image Type.**
   The image types that can be created from a prepared image tree depend on the types specified in the image description `config.xml` file. The configuration file must contain at least one `type` element. The figure below shows the currently supported image types:

Figure 3.2. Image Types



(1) Live Image
For CDs, DVDs or flash disks.

(2) Disk Image
Virtual system disk that can be used in virtual environments such as VMware, Xen, Amazon Cloud, KVM, and others. Depending on the format a guest configuration file is created.

(3) OEM Image
Preload system for install media CD/DVD or flash disk.

(4) PXE Image
Network boot image. KIWI also provides the bootp environment via the package kiwi-pxeboot.

Detailed information, including step-by-step instructions on building specific images can be found in Part II, “Usecases”. That part of the manual explains how to build a “Just enough Operation System” (JeOS) image from a KIWI template for all supported image types.

3.2. Customizing the Boot Process

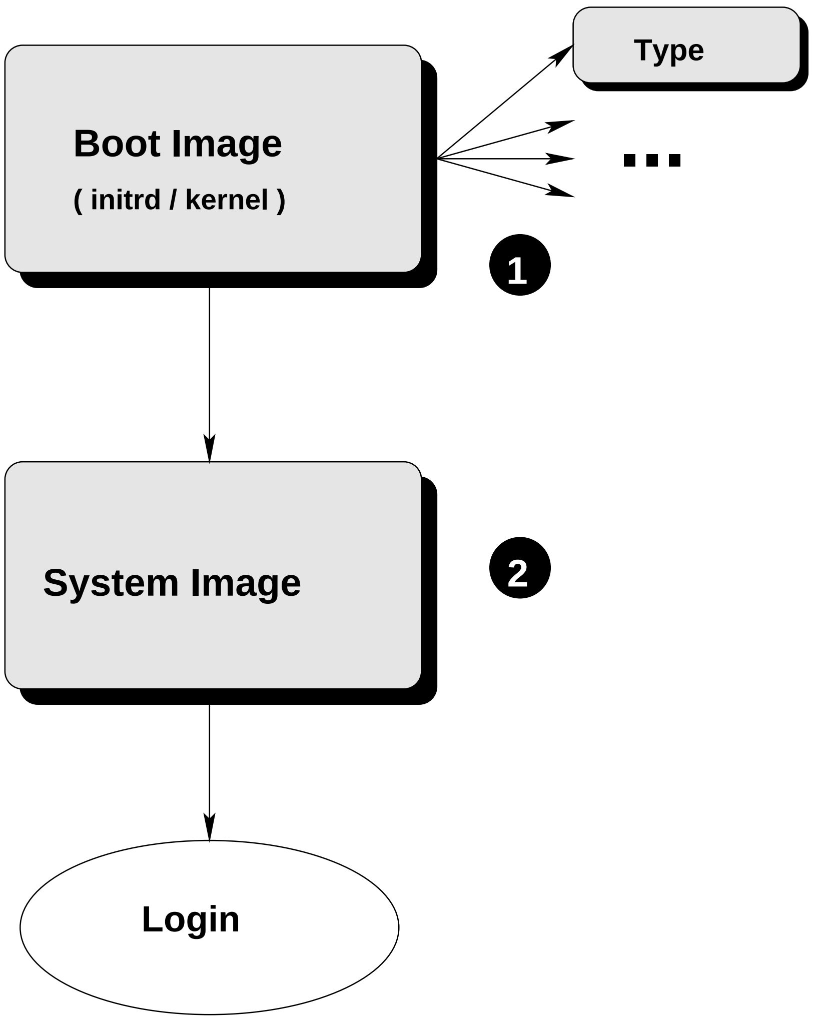
Most Linux systems use a special boot image to control the system boot process after the system firmware, BIOS or UEFI, hands control of the hardware to the operating system. This boot image is called the *initrd*. The Linux kernel loads the initrd, a compressed cpio initial RAM disk, into the RAM and executes *init* or, if present, *linuxrc*.

Depending on the image type, KIWI creates the boot image automatically during the *create* step. Each image type has its own description for the boot image. Common functionality is shared between the boot images through a set of functions. The boot image descriptions follow the same principles as the system image descriptions, KIWI ships with pre-defined boot image descriptions.

Boot Image Descriptions provided by KIWI

The boot image descriptions provided by KIWI cover almost all use cases. Creating custom boot descriptions should not be necessary, unless you have special requirements.

Figure 3.3. Image Descriptions



(1) Boot Image
Boot image descriptions are provided by KIWI, use is recommended but not required.

(2) System Image
The system image description is created by the KIWI user, or a KIWI provided template may be used.

The boot image descriptions are stored in the `/usr/share/kiwi/image/*boot` directories. KIWI selects the boot image based on the value of the `boot` attribute of the `type` element. The attribute value is expected in the general form of *`boottype/distribution`*. For example to select the OEM boot image for SLES version 12 the element would look like the following:

```
<type boot="oemboot/suse-SLES12">
```

Difference Between Boot Image and System Image Descriptions

The *boot image description* only represents the initrd used to boot the system and as such serves a limited purpose. The boot image descriptions is used to build the boot image independently from the system image. Usually a pre-defined boot image descriptions shipped with KIWI is used.

The system image description is used to build the image running on the target system. It is manually created and usually tailor-made for a specific use case.

De-activating Hooks at Boot Time

The execution of hooks can be globally deactivated by passing the following variable to the kernel command line:

```
KIWI_FORBID_HOOKS=1
```

3.2.1. Boot Image Hook-Scripts

All KIWI created boot images contain kiwi boot code that gets executed when the image is booted for the first time. This boot code differs from image type to image type. It provides hooks to execute user defined shell scripts.

These scripts may extend the firstboot process and are expected to exist inside the boot image in a specific location with specific names. The following instructions explain the concept of hook scripts, which is common to all image types, and how to include the scripts in the initrd.

3.2.1.1. Script Types

Hook scripts are executed using a predetermined name that is hard coded into the kiwi boot code. This name is extended using the `.sh` extension and differs by boot image type. Therefore, the boot script naming in the archive must be exact. Boot scripts are sourced in the kiwi boot code. This provides the hook script access to all variables set in the boot environment. This also implies that no separate shell process is started and the boot scripts do not need to have the executable bit set. Encoding the interpreter location with the `#!` comment is superfluous.

The following list provides information about the hook names, timing of the execution, and the applicable boot image.

`handleSplash`
: This hook is called prior to any dialog/exception message or progress dialog. The hook can be used to customize the behavior of the splash screen. KIWI automatically hides a plymouth or kernel based splash screen if there is only one active console.

`init`
: This hook is called before udev is started. It exists only for the *PXE* image type.

`preconfig|postconfig`
: The hooks are called before and after the client configuration files (CONF contents) are setup, respectively. The hooks only exist for the *PXE* image type.

`predownload|postdownload`
: The hooks are called before and after the client image receives the root file system, respectively. The hooks only exist for the *PXE* image type.

`preImageDump|postImageDump`
: The hooks are called before and after the install image is dumped on the target disk, respectively. The hooks only exist for the *OEM* image type.

`preLoadConfiguration|postLoadConfiguration`
: The hooks are called before and after the client configuration file `config.MAC` is loaded, respectively. The hooks only exist for the *PXE* image type.

`premount|postmount`
: The hooks are called before and after the client root file system is mounted, respectively. The hooks only exist for the *PXE* image type.

`prenetwork|postnetwork`
: The hooks are called before and after the client network is setup, respectively. The hooks only exist for the *PXE* image type.

`prepartition|postpartition`
: The hooks are called before and after the client creates the partition table on the target disk, respectively. The hooks only exist for the *PXE* image type.

`preprobe|postprobe`
: The hooks are called before and after the loading of modules not handled by udev, respectively. The hooks only exist for the *PXE* image type.

`preswap|postswap`
: The hooks are called before and after the creation of the swap space, respectively. The hooks only exist for the *PXE* image type.

`preactivate`
: This hook is called before the root file system is moved to `/`. The hook only exists for the *pxe* image type.

`preCallInit`
: This hook is called before the initialization process, init or systemd, is started. At call time the root file system has already been moved to `/`.. The hook only exists for the *OEM* and *VMX* image types.

`preRecovery|postRecovery`
: This hook is called before and after the recovery code is processed. At call time of preRecovery the recovery partition is not yet mounted. At call time of postRecovery the recovery partition is still mounted on `/reco-save`. The hook only exists for the *OEM* image type.

`preRecoverySetup|postRecoverySetup`
: This hook is called before and after the recovery setup is processed. At call time of preRecoverySetup the recovery partition is not yet mounted. At call time of postRecoverySetup the recovery partition is still mounted on */reco-save*. The hook only exists for the *OEM* image type.

`preException`
: This hook is called before a system error is handled. The error message is passed as parameter. This hook can be used for all image types.

`preHWdetect|postHWdetect`
: The hooks are called before and after the install image boot code detects the possible target storage device(s). The hooks only exist for the *OEM* image type.

`preNetworkRelease`
: This hook is called before the network connection is released. The hook only exists for the *PXE* image type.

3.2.1.2. Including Hook Scripts into the Boot Image

All hook scripts must be located in the `kiwi-hooks` directory at the top level of the initrd. The best approach to including the hook scripts in the initrd is to create an archive of a `kiwi-hooks` directory that contains the custom boot scripts.

```
mkdir kiwi-hooks
place all scripts inside kiwi-hooks
tar -cf kiwi-hooks.tgz kiwi-hooks/
```

The TAR archive must be located at the top level of the image description directory, this is the same level that contains the `config.xml` file.

Hook scripts are only executed from within kiwi's boot code and must therefore be part of the KIWI created boot image. Including the content of a TAR archive in the initrd is accomplished by setting the value of the `bootinclude` attribute of the `archive` element to `true` in the `config.xml` file as shown below:

```
<packages type="image">
  <archive name="kiwi-hooks.tgz" bootinclude="true"/>
</packages>
```

The concept of including an archive in the boot image follows the same concepts described for the system image previously. To use an archive in a pre-built boot image the archive must be part of the boot image description in which case it is not necessary to set the `bootinclude` attribute.

3.2.1.3. Post Commands

In addition to the hook script itself it is also possible to run a post command after the hook script was called. This allows to run commands tied to a hook script without changing the initrd and thus provides a certain flexibility when writing the hook. The post command execution is based on variables that can be passed to the kernel command line. The following rules for the processing post commands apply:

1. Command post processing needs to be activated within the corresponding hook script. this is achieved by setting the variable `KIWI_ALLOW_HOOK_CMD_`*`HOOKNAME`* to 1. For example:

   ```
   KIWI_ALLOW_HOOK_CMD_preHWdetect=1
   ```

   This will activate the post command execution for the `preHWdetect` hook. If this variable is not set, the post command will not be executed.

2. The corresponding variable `KIWI_HOOK_CMD_`*`HOOKNAME`* needs to passed to the Kernel command line. Its value contains the command that is to be executed, for example:

   ```
   KIWI_HOOK_CMD_preHWdetect="ls -l"
   ```

   This will cause the `preHWdetect` hook to call **ls -l** at the end of the hook script code.

3.

   #### Disable Post Command Execution at Boot Time

   To disable all post commands for the current boot process pass the following variable to the Kernel command line:

   ```
   KIWI_FORBID_HOOK_CMDS=1
   ```

3.2.2. FAQ: Boot Image Customization

The KIWI provided boot image descriptions should satisfy the requirements for a majority of image builds and the environments in which these images are deployed. In case a customized boot image is needed, KIWI provides appropriate configuration options in `config.xml`.

Using these options allows users to base the boot image on the KIWI provided descriptions rather than having to define a configuration from scratch (however, this is possible if wanted).

The following question and answer section provides solutions to the most common scenarios that require a customized boot image.

Q: Why is the boot image so big? Can I reduce its size ?

A: KIWI includes all required tools and libraries to boot the image under all circumstances in all target environments supported by the image type. In case the target environment is well defined it is possible to remove libraries, drivers and tools not needed in the target environment.

This will decrease the size of the initrd and will also decrease boot time. Removing files in the boot image is accomplished by adding a `strip` section to the system image in the `config.xml` file, with the `type` attribute set to `delete`, as shown below:

```
<strip type="delete">
    <file name="..."/>
</strip>
```

Removing files that are needed may result in an image that cannot be booted.

Q: Can drivers be added to the boot image?

A: KIWI uses a subset of the Kernel. Therefore drivers shipped with the Kernel that have not been included by the KIWI build process, can be added. Do so by adding a `drivers` section to the system image configuration file `config.xml`, as follows:

```
<drivers>
  <file name="drivers/..."/>
</drivers>
```

If the driver is provided by a package, the package itself needs to be specified as part of the `image` package section. Additionally, it must be marked for boot image inclusion by setting the value of the `bootinclude` attribute of the `package` element to `true`, as follows:

```
<packages type="image"/>
  <package name="PACKAGE" bootinclude="true"/>
</packages>
```

Q: How to add missing tools or libraries?

A: Additional software can be added to the boot image with the use of the `bootinclude` attribute of the `package` or the `archive` element. At the end of the boot image creation process kiwi attempts to reduce the size of the boot image by removing files that are not part of a known list of required files or their dependencies.

The list of required files is hard coded in the `/usr/share/kiwi/modules/KIWIConfig.txt` file. If you added files to the boot image that are needed for your specific use case, you need to instruct kiwi to not strip them from the image. This is accomplished by adding a `strip` section to the system image `config.xml` file, with the `type` attribute set to `tools`, as follows:

```
<strip type="tools"/>
  <file name="FILENAME"/>
</strip>
```

The removal/preservation of files is name-based only, so you do not need to specify a complete path, but rather the file name.

Q: Is it possible to add boot code?

A: Yes, as described in the Section 3.2.1, “Boot Image Hook-Scripts” section above, KIWI supports the execution of boot code at various times for various image types using *hook* scripts.

Q: Is it possible to include completely customized boot code?

A: No. In cases where the provided hooks are insufficient and the KIWI provided boot code needs to be replaced completely, it is necessary to create a custom boot image description. In this case, all parts of the boot image description must be created by the user. It is best to use one of the KIWI provided boot descriptions as a template.

Q: My customized boot image refuses to boot. How to debug?

A: An initrd created by KIWI that is based on one of the KIWI- provided boot image descriptions recognizes kernel parameters that are useful for debugging purposes, in case the image does not boot. These parameters may not work if the image contains a custom boot image where the kiwi boot code has been completely.

3.2.3. Boot Parameters

A KIWI created initrd based on one of the KIWI provided boot image descriptions recognizes kernel parameters that are useful for debugging purposes, should the image not boot. These parameters may not work if the image contains a custom boot image where the kiwi boot code has been replaced, and the parameters are not recognized after the initial KIWI created initrd has been replaced by the "regular" distribution created initrd after the initial boot of the image.

If the boot process encounters a fatal error, the default behavior is to reboot the system after 120 seconds. Prevent this behavior by specifying

```
kiwidebug=1
```

on the Kernel command line. With that parameter set to `1`, the system will enter a limited shell environment in case of a fatal during boot. The shell contains a basic set of commands. The first place to look for debugging information should be the boot log file `/var/log/kiwi.boot`.

In addition to the shell, KIWI also starts the `dropbear` SSH server if the environment is suitable. Support for `dropbear` can be added to the netboot and oemboot (in PXE boot mode) boot images. For isoboot and vmxboot boot images there is no remote login support because they do not set up a network. It is required that the repository setup provides dropbear.

To have dropbear installed as part of the boot image the following needs to be added to the system image configuration:

```
<packages type="image"/>
  <package name="dropbear" bootinclude="true"/>
</packages>
```

It might be useful to also include a tool for copying remote files, such as **scp** or **rsync** into the boot image. Note that the required packages need to be provided by the repositories configured. To include **rsync**, for example, add the line `<package name="rsync" bootinclude="true"/>` to the listing above.

To access the boot image via SSH it is required to provide a public key on the PXE server in the directory: *`SERVER-ROOT`*`/KIWI/debug_ssh.pub`. KIWI exclusively searches for that file name, so it is required to name it `debug_ssh.pub`. *`SERVER-ROOT`* depends on what server type was configured to download the image. By default this is done via TFTP. In that case *`SERVER-ROOT`* translates to `/srv/tftpboot` ion the PXE server. Adjust the path accordingly if having used HTTP or FTP.

Adding more than one public key to file is possible, the file uses the same format as the common SSH file “authorized_keys”. If a public key was found you can login as follows:

```
ssh root@IP-ADDRESS
```

In case **rsync** is available, you can copy the KIWI boot log to your local machine as follows:

```
RSYNC_RSH='ssh -l root'
rsync -avz <ip>:/var/log/boot.kiwi
```

3.3. Distribution-Specific Code

KIWI is designed to be distribution-independent. However, Linux distributions differ from each other, primarily in the package management area and in the area of creation and composition of the boot image. Within the KIWI code base major areas of Linux distribution differences are isolated into specific regions of the code. The remainder of the code is common and distribution- independent.

KIWI-provided functions that are distribution-specific contain the distribution name as a prefix, such as `suseStripKernel`. Scripts that are part of the boot code and are distribution-specific are identified by a prefix of the distribution name followed by a “-”, for example `suse-linuxrc`. When KIWI creates a boot image for a SUSE distribution the **suse-linuxrc** file from the boot description is used as the **linuxrc** file that the Linux kernel calls.

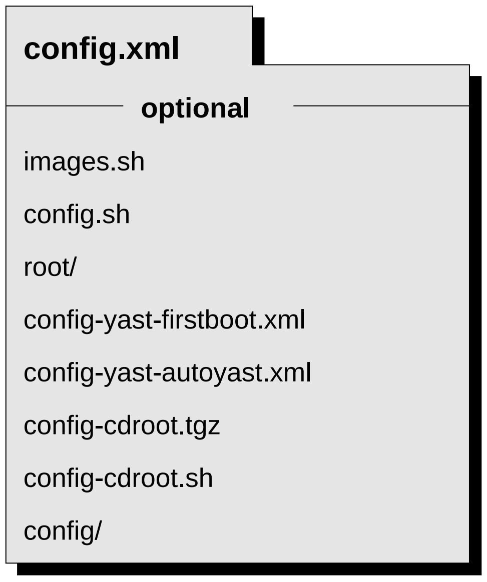
With this design it is possible to maintain distribution-specific code in the project while also providing explicit hints to the user when distribution specific code is being used. The implementation of SUSE-specific code can be used as a guideline to support other distributions.

4 KIWI Image Description

Table of Contents

To be able to create an image with KIWI, a so called image description must be created. The image description is represented by a directory which needs to contain at least one file named `config.xml` or `*.kiwi`. A good start for such a description can be found in the examples provided in `/usr/share/doc/packages/kiwi/examples`.
2015-07-24 - fs: TODO
1. Explain templates (packages, overview of templates, where to find them, how to use them, how to customize)
2. How to manually validate, editor support (loading schema or DTD)
3. Shorten, restructure for a better readability
4. Point to schema description, explain how to use/read it

Figure 4.1. Image Description Directory



The following additional information is optional for the process of building an image, but most often mandatory for the functionality of the created operating system:

`images.sh`
: Optional configuration script while creating the packed image. This script is called at the beginning of the image creation process. It is designed to clean-up the image system. Affected are all the programs and files only needed while the unpacked image exists.

`config.sh`
: Optional configuration script while creating the unpacked image. This script is called at the end of the installation, but *before* the package scripts have run. It is designed to configure the image system, such as the activation or deactivation of certain services (`insserv`). The call is not made until after the switch to the image has been made with chroot.

`root`
: Subdirectory that contains special files, directories, and scripts for adapting the image environment *after* the installation of all the image packages. The entire directory is copied into the root of the image tree using **cp** `-a`.

`config-yast-autoyast.xml`
: Configuration file which has been created by AutoYaST. To be able to create such an AutoYaST profile, run:

```
yast2 autoyast
```

Once you have saved the information from the AutoYaST UI as `config-yast-autoyast.xml` file in your image description directory KIWI will process on the file and setup your image as follows:

1. While booting the image YaST is started in AutoYaST mode automatically
2. The AutoYaST description is parsed and the instructions are handled by YaST. In other words the *system configuration* is performed
3. If the process finished successfully the environment is cleaned and AutoYaST won't be called at next reboot.

`config-cdroot.tgz`
: Archive which is used for ISO images only. The data in the archive is uncompressed and stored in the CD/DVD root directory. This archive can be used, for example, to integrate a license file or information directly readable from the CD or DVD.

`config-cdroot.sh`
: Along with the `config-cdroot.tgz` one can provide a script which allows to manipulate the extracted data.

`config/`
: Optional subdirectory that contains Bash scripts that are called after the installation of all the image packages, primarily to remove the parts of a package that are not needed for the operating system. The name of the Bash script must resemble the package name listed in the config.xml.

4.1. The config.xml File

The mandatory image definition file is divided into different sections which describes information like the image name and type as well as the packages and patterns the image should consist of.

The following information explains the basic structure of the XML document. When KIWI is executed, the XML structure is validated by the KIWI RELAX NG based schema. For details on attributes and values please refer to the schema documentation file at `/usr/share/doc/packages/kiwi/kiwi.rng.html`.

4.1.1. image Element

```
<image schemaversion="6.2" name="iname"
  displayname="text"
  kiwirevision="number"
  id="10 digit number">
  <!-- ... -->
</image>
```

The image definition starts with an image tag and requires the schema format at version 2.0. The attribute name specifies the name of the image which is also used for the filenames created by KIWI. Because we don’t want spaces in filenames the name attribute must not have any spaces in its name.

The following optional attributes can be inserted in the image tag:

displayname
: Allows setup of the boot menu title for the selected boot loader. So you can have *suse-SLED-foo* as the image name but a different name as the boot display name. Spaces are not allowed in the display name because it causes problems for some boot loaders and kiwi did not take the effort to separate the ones which can display them correctly from the ones which can't

kiwirevision
: specifies a KIWI git revision number which is known to build a working image from this description. If the KIWI git revision doesn't match the specified value, the process will exit. The currently used git revision can be queried by calling **kiwi** --version.

id
: sets an identification number which appears as file /etc/ImageID within the image.

Inside the image section the following mandatory and optional subelements exists. The simplest image description must define the elements description, preferences, repository and packages (at least one of type="bootstrap").

4.1.2. description Element

```
<description type="system">
  <author>an author</author>
  <contact>mail</contact>
  <specification>short info</specification>
</description>
```

The mandatory description section contains information about the creator of this image description. The attribute type could be either of the value system which indicates this is a system image description or at value boot for boot image descriptions.

4.1.3. profiles Element

```
<profiles>
   <profile name="name" description="text"/>
   <!-- ... -->
</profiles>
```

The optional profiles section lets you maintain one image description while allowing for variation of the sections packages and drivers that are included. A separate profile element must be specified for each variation. The profile child element, which has name and description attributes, specifies an alias name used to mark sections as belonging to a profile, and a short description explaining what this profile does.

To mark a set of packages/drivers as belonging to a profile, simply annotate them with the profiles attribute. It is also possible to mark sections as belonging to multiple profiles by separating the names in the profiles attribute with a comma. If a packages or drivers tag does not have a profiles attribute, it is assumed to be present for all profiles.

4.1.4. preferences Element

```
<preferences profiles="name">
  <version>1.1.2</version>
  <packagemanager>zypper</packagemanager>
  <type image="name" ...>
    <machine|oemconfig|pxedeploy|size|split|systemdisk|vagrantconfig>
  </type>
</preferences>
```

The mandatory preferences section contains information about the supported image type(s), the used package manager, the version of this image, and optional attributes. The image version must be a three-part version number of the format: **Major**.**Minor**.**Release**. In case of changes to the image description the following rules should apply:

- For smaller image modifications that do not add or remove any new packages, only the release number is incremented. The config.xml file remains unchanged.
- For image changes that involve the addition or removal of packages the minor number is incremented and the release number is reset.
- For image changes that change the size of the image file the major number is incremented.

By default, KIWI uses the **zypper** package manager but it is also possible to use the non SUSE native package manager called **smart**.

In general the specification of one preferences section is sufficient. However, it's possible to specify multiple preferences sections and distinguish between the sections via the profiles attribute. Data may also be shared between different profiles. Using profiles it is possible to, for example, configure specific preferences for OEM image generation. Activation of a given preferences during image generation is triggered by the use of the --add-profile command line argument.

For each preferences block at least one type element must be defined. It is possible to specify multiple type elements in any preferences block. To set a given type description as the default image use the boolean attribute primary and set its value to true. The image type to be created is determined by the value of the image attribute. The following list describes the supported types and possible values of the image attribute:

image="lxc|docker"
: Use the lxc or docker image type to build a linux container image. For additional information refer to the Chapter 9, *Docker images* chapter.

image="[filesystem]"
: Use one of the following image types to build a plain filesystem image. This will create a file containing the data in the specified filesystem and you can loop mount the image to view the contents e.g image="ext3":

- ext2
- ext3
- ext4
- btrfs
- squashfs
- xfs

`image="tbz"`
: Use the tbz image type to just pack the unpacked image tree into a tarball.

`image="cpio"`
: Use the cpio image type to specify the generation of a boot image (initrd). When generating a boot image, it is possible to specify a specific boot profile and boot kernel using the optional `bootprofile="default"` and `bootkernel="std"` attributes.

  A boot image should group the various supported kernels into profiles. If the user chooses not to use the profiles supplied by KIWI, it is required that one profile named std be created. This profile will be used if no other bootkernel is specified. Further it is required to create a profile named default. This profile is used when no bootprofile is specified.

  It is recommended that special configurations that omit drivers, use special drivers and/or special packages be specified as profiles.

  The bootprofile and bootkernel attribute are respected within the definition of a system image. Us the attribute and value `type="system"` of the `description` element to specify the creation of a system image. The values of the bootprofile and bootkernel attributes are used by KIWI when generating the boot image.

`image="iso"`
: Specify the key-value pair `image="iso"` to generate a live system suitable for deployment on optical media (CD or DVD). Use the `boot="isoboot/suse-*"` attribute when generating this image type to select the appropriate boot image for optical media. In addition the optional `flags` attribute may be set to the following values with the effects described below:

  `seed`
  : Creates a btrfs based compressed read-only filesystem which allows write operations into a btrfs seed device.

  `overlay`
  : Creates a squashfs based compressed read-only filesystem which is combined with a write space via the overlayfs filesystem. overlayfs is part of the kernel since version 3.7

  `compressed`
  : Creates a split ext3 plus squashfs filesystem and combines them via a symlink system to a complete system it is recommended to specify a `split` section as a child of this type element.

  If the flags attribute is not used the filesystem will be squashfs compressed for /bin /boot /lib /lib64 /opt /sbin and /usr. The rest of the filesystem is packed into a tmpfs and linked via symbolic links

`image="oem"`
: Use this type to create a virtual disk system suitable in a preload setting. In addition specify the attributes `filesystem`, and `boot="oemboot/suse-*"` to control the filesystem used for the virtual and to specify the proper boot image. Using the optional `format` attribute and setting, the value to iso or usb will create self installing images suitable for optical media or a USB stick, respectively. Booting from the media will deploy the OEM preload image onto the selected storage device of the system. It is also possible to configure the system to use logical volumes. Use the optional `lvm` attribute and specify the logical volume configuration with the `systemdisk` child element. The default volume group name is kiwiVG. Further configuration of the image is performed using the appropriate `*config` child block.

`image="pxe"`
: Creating a network boot image is supported by KIWI with the image="pxe" type. When specifying the creation of a network boot image use the `filesystem` and boot="netboot/suse-*" attributes to specify the filesystem of the image and the proper boot image. To compress the image file set the `compressed` boolean attribute to true. This setting will compress the image file and has no influence on the filesystem used within the image. The compression is often use to support better transfer times when the pxe image is pushed to the boot server over a network connection. The pxe image layout is controlled by using the `pxedeploy` child element.

`image="split"`
: The split image support allows the creation of an image as split files. Using this technique one can assign different file systems and different read-write properties to the different sections of the image. The `oem`, `pxe`, `usb`, and `vmx` types can be created as a split system image. Use the `boot="oem|netboot|usb|vmx/suse-*"` attribute to select the underlying type of the split image. The attributes `fsreadwrite`, `fsreadonly` are used to control the read-write properties of the filesystem specified as the attributes value. Use the appropriate `*config` child block to specify the properties of the underlying image. For example when building a OEM based split image use the `oemconfig` child section.

`image="vmx"`
: Creation of a virtual disk system is enabled with the vmx value of the image attribute. Set the filesystem of the virtual disk with the filesystem attribute and select the appropriate boot image by setting `boot="vmxboot/suse-*"` The optional format attribute is used to specify one of the virtualization formats supported by QEMU, such as vmdk (also the VMware format) or qcow2. For the virtual disk image the optional vga attribute may be used to configure the kernel framebuffer device. Acceptable values can be found in the Linux kernel documentation for the framebuffer device (see `Documentation/fb/vesafb.txt`). KIWI also supports the selection of the boot loader for the virtual disk according to the rules indicated for the USB system. Last but not least the virtual disk system may also be created with a LVM based layout by using the `lvm` attribute. The previously indicated rules apply. Use the `machine` child element to specify appropriate configuration of the virtual disk system.

Within the type section, there could be other optional attributes which are either universally valid or can be used for different image types in the same way. The following list explains these attributes:

`kernelcmdline`
: Specifies additional kernel parameters. The following example disables kernel messages: `kernelcmdline="quiet"`

`mdraid`
: For disk based image types, aka oem and vmx, mdraid activates the creation of a software raid image. The raid inside the image is created in degraded mode because at creation time we only know about one disk. It's in the hand of the user to add devices to the raid after the image runs on the target machine. The value for mdraid can be either *mirroring* or *striping*, which means the raid level is set to RAID1 (mirroring) or RAID0 (striping).

Within the preferences section, there are the following optional elements:

`showlicense`
: Specifies the base name of a license file which is displayed in oem images before the installation happens. It's possible to add more showlicense sections to display more licenses one after the other. If no such element is specified the default 'license' and 'EULA' files are searched. The search algorithm will append the .txt or .locale.txt suffix to the license name to form the license file name. You should make sure that you license files contains this suffix.

`rpm-check-signatures`
: Specifies whether RPM should check the package signature or not

`rpm-excludedocs`
: Specifies whether RPM should skip installing package documentation

`rpm-force`
: Specifies whether RPM should be called with `--force`

`keytable`
: Specifies the name of the console keymap to use. The value corresponds to a map file in `/usr/share/kbd/keymaps`. The `KEYTABLE` variable in `/etc/sysconfig/keyboard` file is set according to the keyboard mapping.

`timezone`
: Specifies the time zone. Available time zones are located in the `/usr/share/zoneinfo` directory. Specify the attribute value relative to `/usr/share/zoneinfo`. For example, specify Europe/Berlin for `/usr/share/zoneinfo/Europe/Berlin`. KIWI uses this value to configure the timezone in `/etc/localtime` for the image.

`locale`
: Specifies the name of the UTF-8 locale to use, which defines the contents of the `RC_LANG` system environment variable in `/etc/sysconfig/language`. Please note only UTF-8 locales are supported here which also means that the encoding must *not* be part of the locale information. The KIWI schema validates the locale string according to the following pattern:`[a-z]{2}_[A-Z]{2}(,[a-z]{2}_[A-Z]{2})*`. This means you need to specify the locale like the following example: en_US or en_US,de_DE

`bootsplash-theme`
: Specifies the name of the bootsplash theme to use

`bootloader-theme`
: Specifies the name of the gfxboot theme to use

`defaultdestination`
: Used if the `--destdir` option is not specified when calling KIWI

defaultroot
: Used if the option --root is not specified when calling KIWI

The type element may contain child elements to provide specific configuration information for the given type. The following lists the supported child elements:

systemdisk
: Using the optional systemdisk section it is possible to create a LVM (Logical Volume Management) based storage layout or a btrfs based layout using sub volumes. See chapter 17 for details.

  By default, the volume group is named *kiwiVG*. It is possible to change the name of the group by setting the name attribute to the desired name. Individual volumes within the volume group are specified using the volume element.

  The following example shows the creation of a volume named *usr* and a volume named *var* inside the volume group systemVG.

```
 <systemdisk name="systemVG">
  <volume name="usr" freespace="100M"/>
  <volume name="var" size="200M"/>
</systemdisk>
```

  The optional attribute freespace controls the amount of unused space available after software has been installed in the given volume. By default the available space of a created volume is between 10% and 20%. Using the optional size attribute the absolute size of the given volume is specified. The size attribute takes precedence over the freespace attribute. If the specified size is insufficient, based on the estimated software install size for the given volume, the specified value will be ignored and a volume with default settings will be created. This implies that the volume will be 80% to 90% full.

oemconfig
: *2015-11-30 - fs: This whole section has also been copied to the OEM chapter* By default, the oemboot process will create or modify a swap, and / partition. It is possible to influence the behavior by the oem-* elements explained below.

```
<oemconfig>
   <oem-systemsize>2000</oem-systemsize>
   <oem-... >
</oemconfig>
```

  <oem-boot-title>text</oem-boot-title>
  : By default, the string OEM will be used as the boot manager menu entry when KIWI creates the GRUB configuration during deployment. The oem-boot-title element allows you to set a custom name for the grub menu entry. This value is represented by the kiwi_oemtitle variable in the initrd

  <oem-bootwait>true|false</oem-bootwait>
  : Specify if the system should wait for user interaction prior to continuing the boot process after the oem image has been dumped to the designated storage device (default value is false). This value is represented by the kiwi_oembootwait variable in the initrd

  <oem-inplace-recovery>true|false</oem-inplace-recovery>
  : Specify if the recovery archive is stored as part of the image or if the archive is to be created at the time the image is deployed to the target storage device. kiwi_oemrecoveryInPlace variable in the initrd

`<oem-kiwi-initrd>`true|false`</oem-kiwi-initrd>`
: If this element is set to true (default value is false) the oemboot boot image (initrd) will *not* be replaced by the system (mkinitrd) created initrd. This option is useful when the system is installed on removable storage such as a USB stick or a portable external drive. For movable devices it is potentially necessary to detect the storage location during every boot. This detection process is part of the oemboot boot image. This value is represented by the `kiwi_oemkboot` variable in the initrd

`<oem-partition-install>`true|false`</oem-partition-install>`
: Specify if the image is to be installed into a free partition on the target storage device. By default the value is false and Kiwi installs images to a target device which causes data loss on the device. With `oem-partition-install` set to true any other settings that have influence on the partition table, such as `oem-swap` are ignored. This value is represented by the `kiwi_oempartition_install` variable in the initrd

`<oem-reboot>`true|false`</oem-reboot>`
: Specify if the system is to be rebooted after the oem image has been deployed to the designated storage device (default value is false). This value is represented by the `kiwi_oemreboot` variable in the initrd

`<oem-reboot-interactive>`true|false`</oem-reboot-interactive>`
: Specify if the system is to be rebooted after the oem image has been deployed to the designated storage device (default value is false). Prior to reboot a message is posted and must be acknowledged by the user in order for the system to reboot. This value is represented by the `kiwi_oemrebootinteractive` variable in the initrd

`<oem-recovery>`true|false`</oem-recovery>`
: If this element is set to true (default value is false), KIWI will create a recovery archive from the prepared root tree. The archive will appear as `/recovery.tar.bz2` in the image file. During first boot of the image a single recovery partition will be created and the recovery archive will be moved to the recovery partition. An additional boot menu entry is created that when selected restores the original root tree on the system. The user information on the `/home` partition or in the `/home` directory is not affected by the recovery process. This value is represented by the kiwi_oemrecovery variable in the initrd

`<oem-recoveryID>`partition-id`</oem-recoveryID>`
: Specify the partition type for the recovery partition. The default is to create a Linux partition (id = 83). This value is represented by the `kiwi_oemrecoveryID` variable in the initrd

`<oem-silent-boot>`true|false`</oem-silent-boot>`
: Specify if the system should boot in silent mode after the oem image has been deployed to the designated storage device (default value is false). This value is represented by the `kiwi_oemsilentboot` variable in the initrd

`<oem-shutdown>`true|false`</oem-shutdown>`
: Specify if the system is to be powered down after the oem image has been deployed to the designated storage device (default value is false). This value is represented by the `kiwi_oemshutdown` variable in the initrd

`<oem-shutdown-interactive>`true|false`</oem-shutdown-interactive>`
: Specify if the system is to be powered down after the oem image has been deployed to the designated storage device (default value is false). Prior to shutdown a message

is posted and must be acknowledged by the user in order for the system to power off. This value is represented by the `kiwi_oemshutdowninteractive` variable in the initrd

`<oem-swap>`true|false`</oem-swap>`
: Specify if a swap partition should be created. The creation of a swap partition is the default behavior. This value is represented by the `kiwi_oemswap` variable in the initrd

`<oem-swapsize>`number in MB`</oem-swapsize>`
: Set the size of the swap partition. If a swap partition is to be created and the size of the swap partition is not specified with this optional element, KIWI will calculate the size of the swap partition and create a swap partition equal to two times the RAM installed on the system at initial boot time. This value is represented by the `kiwi_oemswapMB` variable in the initrd

`<oem-systemsize>`number in MB`</oem-systemsize>`
: Set the size the operating system is allowed to consume on the target disk. The size limit does not include any consideration for swap space or a recovery partition. In a setup *without* a `systemdisk` element this value specifies the size of the root partition. In a setup *including* a `systemdisk` element this value specifies the size of the LVM partition which contains all specified volumes. Thus, the sum of all specified volume sizes plus the sum of the specified freespace for each volume must be smaller or equal to the size specified with the `oem-systemsize`. This value is represented by the variable `kiwi_oemrootMB` in the initrd

`<oem-unattended>`true|false`</oem-unattended>`
: The installation of the image to the target system occurs automatically without requiring user interaction. If multiple possible target devices are discovered the image is deployed to the first device. `kiwi_oemunattended` in the initrd

`pxedeploy`
: Information contained in the optional `pxedeploy` section is only considered if the `image` attribute of the `type` element is set to `pxe`. To use a PXE image it is necessary to create a network boot infrastructure. Creation of the network boot infrastructure is simplified by the KIWI provided package kiwi-pxeboot . This package configures the basic PXE boot environment as expected by KIWI pxe images. The kiwi-pxeboot package creates a directory structure in `/srv/tftpboot`. Files created by the KIWI create step need to be copied to the `/srv/tftpboot` directory structure. For additional details about the PXE image please refer to the PXE Image chapter later in this document.

In addition to the image files it is necessary that information be provided about the client setup. This information, such as the image to be used or the partitioning, is contained in a file with the name `config.`*`MAC`* in the directory `/srv/tftpboot/KIWI`. The content of this file is created automatically by KIWI if the pxedeploy section is provided in the image description. A pxedeploy section is outlined below:

```
<pxedeploy server="IP" blocksize="4096">
   <timeout>seconds</timeout>
   <kernel>kernel-file</kernel>
   <initrd>initrd-file</initrd>
   <partitions device="/dev/sda">
     <partition type="swap" number="1" size="MB"/>
     <partition type="L" number="2" size="MB"
              mountpoint="/" target="true"/>
     <partition type="fd"  number="3"/>
   </partitions>
   <union ro="dev" rw="dev" type="clicfs"/>
   <configuration source="/KIWI/../file" dest="/../file" arch="..."/>
   <configuration .../>
</pxedeploy>
```

- The server attribute is used to specify the IP address of the PXE server. The blocksize attributes specifies the blocksize for the image download. Other protocols are supported by KIWI but require the *kiwiserver* and *kiwiservertype* kernel parameters to be set when the client boots.

- The value of the optional timeout element specifies the grub timeout in seconds to be used when the KIWI initrd configures and installs the grub boot loader on the client machine after the first deployment to allow standalone boot.

- Passing kernel parameters is possible with the use of the optional kernelcmdline attribute in the type section. The value of this attribute is a string specifying the settings to be passed to the kernel by the GRUB bootloader. The KIWI initrd includes these kernel options when installing grub for standalone boot

- The optional kernel and initrd elements are used to specify the file names for the kernel and initrd on the boot server respectively. When using a special boot method not supported by the distribution's standard mkinitrd, it is imperative that the KIWI initrd remains on the PXE server and also be used for local boot. If the configured image uses the split type or the pxedeploy section includes any union information the kernel and initrd elements must be used.

- The partitions section is required if the system image is to be installed on a disk or other permanent storage device. Each partition is specified with one partition child element. The mandatory type attribute specifies the partition type id.

  The required number attribute provides the number of the partition to be created. The size of the partition may be specified with the optional size attribute. The optional mountpoint attribute provides the value for the mount point of the partition. The optional boolean target attribute identifies the partition as the system image target partition. KIWI always generates the swap partition as the first partition of the netboot boot image. By default, the second partition is used for the system image. Use the boolean target attribute to change this behavior. Providing the value image for the size attribute triggers KIWI into calculating the required size for this partition. The calculated size is sufficient for the created image.

- If the system image is based on a read-only filesystem such as squashfs and should be mounted in read-write mode use the optional union element. The type attribute is used to specify one of the supported overlay filesystem clicfs Use the ro attribute to point to the read only device and the rw attribute to point to the read-write device.

- The optional configuration element is used to integrate a network client's configuration files that are stored on the server. The source attribute specifies the path on the server for the file to be downloaded. The dest attribute specifies destination of the downloaded file on the network client starting at the root (/) of the filesystem. Multiple configuration elements may be specified such that multiple files can be transferred to the network client. In addition configuration files can be bound to a specific client architecture by setting the optional arch attribute. To specify multiple architectures use a comma separated string.

size
: Use the size element to specify the image size in Megabytes or Gigabytes. The unit attribute specifies whether the given value will be interpreted as Megabytes (unit="M") or

Gigabytes (unit="G"). The optional boolean attribute additive specifies whether or not the given size should be added to the size of the generated image or not.

In the event of a size specification that is too small for the generated image, KIWI will expand the size automatically unless the image size exceeds the specified size by 100 MB or more. In this case KIWI will generate an error and exit.

Should the given size exceed the necessary size for the image KIWI will not alter the image size as the free space might be required for proper execution of components within the image.

If the size element is not used, KIWI will create an image containing approximately 30 % free space.

```
<size unit="M">1000</size>
```

`split`
: For images of type split or iso the information provided in the optional `split` section is considered if the compressed attribute is set to true. With the configuration in this block it is possible to determine which files are writable and whether these files should be persistently writable or temporarily. Note that for ISO images only temporary write access is possible.

When processing the provided configuration KIWI distinguishes between directories and files. For example, providing `/etc` as the value of the name attribute indicates that the `/etc` directory should be writable. However, this does not include any of the files or sub-directories within `/etc`. The content of /etc is populated as symbolic links to the read-only files. The advantage of setting only a directory to read-write access is that any newly created files will be stored on the disk instead of in `tmpfs`. Creating read-write access to a directory and it’s files requires two specifications as shown below.

```
<split>
  <temporary>
    <!-- read/write access to -->
    <file name="/var"/>
    <file name="/var/*"/>
    <!-- but not on this file: -->
    <except name="/etc/shadow"/>
  </temporary>
  <persistent>
    <!-- persistent read/write access to: -->
    <file name="/etc"/>
    <file name="/etc/*"/>
    <!-- but not on this file: -->
    <except name="/etc/passwd"/>
  </persistent>
</split>
```

Use the except element to specify exceptions to previously configured rules.

`machine`
: The optional machine section serves to specify information about a VM guest machine. Using the data provided in this section, KIWI will create a guest configuration file required to run the image on the target machine.

If the target is a VMware virtual machine indicated by the format attribute set to vmdk, KIWI creates a VMware configuration file. If the target is a Xen virtual machine indicated by the domain attribute in the machine section KIWI will create a Xen guest config file.

The sample block below shows the general outline of the information that can be specified to generate the configuration file

```
<machine arch="arch" memory="MB"
  HWversion="number" guestOS="suse|sles"
  domain="dom0|domU"/>
   <vmconfig-entry>Entry_for_VM_config_file<\vmconfig-entry>
   <vmconfig-entry .../>
   <vmnic driver="name" interface="number" mode="mode"/>
   <vmnic ...>
   <vmdisk controller="ide|scsi" id="number"/>
   <vmdvd  controller="ide|scsi" id="number"/>
</machine>
```

`arch`
: The virtualized architecture. Supported values are `ix86` or `x86_64`. The default value is `ix86`.

`memory`
: The mandatory `memory` attribute specifies how much memory in MB should be allocated for the virtual machine

`HWversion`
: The VMware hardware version number, the default value is `3`.

`guestOS`
: The guest OS identifier. For the ix86 architecture the default value is suse and for the x86_64 architecture suse-64 is the default. At this point only the SUSE and SLES guestOS types are supported.

`domain`
: The Xen domain setup. This could be either a dom0 which is the host machine hosting the guests and therefore doesn’t require a configuration file, or it could be set to domU which indicates this is a guest and also requires a guest configuration which is created by KIWI.

Use the `vmconfig-entry` element to create entries in the virtual machine's configuration file; .vmx for VMware images and .xenconfig for Xen images. You may specify as many configuration options as desired. The value of the `vmconfig-entry` element is expected to be specified in the syntax required by the VM configuration file to be written. The value is free format text and is not validated by Kiwi in any way. The entry is written to the VM configuration file verbatim.

Use the `vmdisk` element to setup the virtual main storage device.

`controller`
: Supported values for the mandatory `controller` attribute are `ide` and `scsi`.

`id`
: The mandatory `id` attribute specifies the disk id. If only one disk is set the id value should be set to 0.

`device`
: The device attribute specifies the disk that should appear in the para virtual instance. Therefore only relevant for Xen

Use the `vmdvd` element to setup a virtual optical drive (CD/DVD) connection

controller
: Supported values for the mandatory controller attribute are ide and scsi.

id
: The mandatory id attribute specifies the disk id. If only one disk is set the id value should be set to 0.

Use the vmnic element to setup the virtual network interface. Multiple vmnic child elements may be specified to setup multiple virtual network interfaces.

driver
: The mandatory driver attribute specifies the driver to be used for the virtual network card. The supported values are e100, vlance, and vmxnet. If the vmxnet driver is specified the vmware tools must be installed in the image.

interface
: The mandatory interface attribute specifies the interface number. If only one interface is set the value should be set to 0.

mode
: The network mode used to communicate outside the VM. In many cases the bridged mode is used.

4.1.5. users Element

```
<users group="group_name" id="number">
  <user home="dir" id="number" name="user" password="..."
        pwdformat="encrypted|plain" realname="string" shell="path"/>
  <!-- ... -->
</users>
```

The optional users element lists the users belonging to the group specified with the group attribute. At least one user child element must be specified as part of the users element. Multiple users elements may be specified.

The attributes home, id, name, pwd, realname, and shell specify the created users home directory, the user name, the user's password, the user's real name, and the user's login shell, respectively. By default, the value of the password attribute is expected to be an encrypted string. An encrypted password can be created using **kiwi** `--createpassword`. It is also possible to specify the password as a non encrypted string by using the pwdformat attribute and setting it's value to "plain". KIWI will then encrypt the password prior to the user being added to the system.

All specified users and groups will be created if they do not already exist. By default, the defined users will be part of the group specified with the group attribute of the users element and the default group called "users". If it is desired to have the specified users to only be part of the given group it is necessary to specify the id attribute. It is recommended to use a group id greater than 100.

4.1.6. drivers Element

```
<drivers profiles="name">
  <file name="filename"/>
  <!-- ... -->
</drivers>
```

The optional drivers element is only useful for boot images (initrd). As a boot image doesn't need to contain the complete kernel one can save a lot of space if only the required drivers are part of the image. Therefore the drivers section exists. If present only the drivers which matches the file names or glob patterns will be included into the boot image. Each file is specified relative to the /lib/modules/*Version*/kernel directory.

According to the driver element the specified files are searched in the corresponding directory. The information about the driver names is provided as environment variable named like the value of the type attribute and is processed by the function suseStripKernel. According to this along with a boot image description a script called **images.sh** must exist which calls this function in order to allow the driver information to have any effect.

4.1.7. repository Element

```
<repository type="type" alias="name" imageinclude="true|false"
             password="password" priority="number" status="replaceable"
             username="user-name"> <source path="URL"/>
</repository>
```

The mandatory repository element specifies the location and type of a repository to be used by the package manager as a package installation source. The mandatory type attribute specifies the repository type. A specified repository can only be accessed by the chosen package manager if the given type is supported by the specified package manager. KIWI supports smart or zypper as package managers, specified with the packagemanager element. The default package manager is zypper. The following table shows the possible supported repository types for each package manager:

Table 4.1. Supported Package Manager Repository Types

Type	smart	zypper	apt	yum
apt-deb	yes	no	yes	no
rpm-dir	yes	yes	no	no
rpm-md	yes	yes	no	yes
yast2	yes	yes	no	no

The repository element has the following optional attributes:

alias="name"
: Specifies an alternative name for the configured repository. If the attribute is not specified KIWI will generate an alias name by replacing any "/" in the given repository location with an "_". It is helpful to set an alias name if the repository path is insufficient in expressing the purpose of the contained packages.

imageinclude="true|false"
: Specifies whether the given repository should be configured as a repository in the image or not. The default behavior is that repositories used to build an image are not configured as a repository inside the image. This feature allows you to change the behavior by setting the value to true. The repository is configured in the image according to the source path as specified with the path attribute of the source element. Therefore, if the path is not a fully qualified URL, you may need to adjust the repository file in the image to accommodate the expected location. It is recommended that you use the alias attribute in combination with the imageinclude attribute to avoid having unpredictable random names assigned to the repository you wish to include in the image. This also facilitates modification of the

"baseurl" entry in the .repo file from the config.sh script if you need to make adjustments to the path.

`password="string"`
: Specifies a password for the given repository. The `password` attribute must be used in combination with the `username` attribute. Dependent on the repository location this information may not be used.

`prefer-license="true|false"`
: The repository providing this attribute will be used primarily to install the license tarball if found on that repository. If no repository with a preferred license attribute exists, the search happens over all repositories. It's not guaranteed in that case that the search order follows the repository order like they are written into the XML description.

`priority="number"`
: Specifies the repository priority for this given repository. Priority values are treated differently by different package managers. Repository priorities allow the package management system to disambiguate packages that may be contained in more than one of the configured repositories. The smart package manager treats packages from repositories with the *highest* priority number as preferable to packages from a repository with a lower priority number. The value 0 means “no priority is set”. The zypper package manager prefers packages from a repository with a *lower* priority over packages from a repository with higher priority values. The value 99 means “no priority is set”.

`status="replaceable"`
: This attribute should only be applied in the context of a boot image description. Setting the `status` to `replaceable` indicates that the specified repository my be replaced by the repositories specified in the image description. This is important as the KIWI generated boot image, if required, should be created based on packages from the same repositories used to build the system image.

`username="name"`
: Specifies a user name for the given repository. The `username` attribute must be used in combination with the `password` attribute. Dependent on the repository location this information may not be used.

When specifying an https location for a repository it is generally necessary to include the “openssl-certs” and “cracklib-dict-full” packages in the `bootstrap` section of the image configuration.

The location of a repository is specified by the `path` attribute of the mandatory `source` child element. The location specification may include the `%arch` macro which will expand to the architecture of the image building host. The value for the `path` attribute may begin with any of the following location indicators:

`dir:///local/path`
: An absolute path to a directory accessible through the local file system. The “dir://” prefix may be omitted.

`ftp://`*`URL`*
: A ftp protocol based network location.

`http://`*`URL`*
: A http protocol based network location.

`https://`*`URL`*
: A https protocol based network location. See the comment above about the handling of certificates and additional package requirements in the `bootstrap` section of the image configuration.

`iso://`*`path/to/isofile`*
: An absolute path to an .iso file accessible via the local file system. KIWI will loop mount the the .iso file to a KIWI created directory with a generated name. The generated path is provided to the specified package manager as a repository location.

  Using multiple .iso files from the same SLE product, requires that all .iso files are located in the same directory. Only the first .iso file is to be specified as a repository in the `config.xml`. The first .iso file contains all information necessary for the package manager to locate packages that are contained in other .iso files of the same product. Attempting to use multiple .iso files in a series as standalone repositories will result in an error.

`obs://$dir1/$dir2`
: A special network location used with the http protocol. The values of `$dir1` and `$dir2` represent the project location in the openSUSE build service. The location is evaluated as `this://repos/$dir1/$dir2` .

  The “obs://” prefix is also valid as part of the value for the `boot` attribute of the `type`. If used with the `boot` attribute it is evaluated as `this://images/$dir1/$dir2` .

`opensuse://`*`PROJECTNAME`*
: A special network location used with the http protocol. The given *`PROJECTNAME`* specifies a project in the openSUSE Build Service. The repository is a repository of type `rpm-md`. For example: `path= "opensuse://openSUSE:10.3/standard"` .

`plain://`*`URI`*
: A plain resource string. Everything following 'plain://' will be forwarded to the package manager without further modification. This type of location specification is useful when KIWI does not support a specific URI but the specified package manager does.

`smb://`*`Samba share pathname`*
: A path to a samba share using the cifs protocol. KIWI creates a mount point and mounts the share including username and password, if specified. Access to the smb share from within the new root tree is provided via a cifs mount. Therefore, the package providing the cifs tools must be included in the package list for the `bootstrap` section of the image configuration. At the time of this writing the package providing the cifs tools is called *cifs-utils*. If any packages provided by the Samba share are used as part of the boot image the cifs tools must also be included in the boot image. This is accomplished with the `bootinclude` attribute of the `package` element. This is shown in the example below:

```
<packages type="bootstrap">
  <package name="cifs-utils" bootinclude="true"/>
  </packages>
```

`this://`*`PATH`*
: *`PATH`* is the relative location to the image description directory for the current image.

4.1.8. `packages` Element

```
<packages type="type" profiles="name" patternType="type"
   <package name="name" arch="arch"/>
   <package name="name" replaces="name"/>
   <package name="name" bootinclude="true" bootdelete="true"/>
   <archive name="name" bootinclude="true"/>
   <package .../>
   <namedCollection name="name"/>
   <namedCollection .../>
   <opensuseProduct name="name"/>
   <opensuseProduct .../>
   <ignore name="name"/>
   <ignore .../>
</packages>
```

The mandatory `packages` element specifies the list of packages (element `package`) and patterns (element `namedCollection`) to be used with the image. The value of the `type` attribute specifies how the packages and patterns listed are handled, supported values are as follows:

`bootstrap`
: Bootstrap packages, list of packages for the new operating system root tree. The packages list the required components to support a chroot environment in the new system root tree, such as glibc.

`delete`
: Delete packages, list of packages to be deleted from the image being created.

  When using the delete type only `package` elements are considered, all other specifications such as `namedCollection` are ignored. The given package names are stored in the `$delete` environment variable of the `/.profile` file created by KIWI. The list of package names is returned by the `baseGetPackagesForDeletion` function. This list can then be used to delete the packages ignoring requirements or dependencies. This can be accomplished in the **config.sh** or **images.sh** script by calling the following helper function:

  ```
  suseRemovePackagesMarkedForDeletion
  ```

  Note, that the delete value is indiscriminate of the image type being built.

`image`
: Image packages, list of packages to be installed in the image.

`iso`
: Image packages, a list of additional packages to be installed when building an ISO image.

`oem`
: Image packages, a list of additional packages to be installed when building an OEM image.

`pxe`
: Image packages, a list of additional packages to be installed when building an PXE image.

`vmx`
: Image packages, a list of additional packages to be installed when building a vmx virtual image of any format.

4.1.8.1. Using Patterns

Using a pattern name allows you to considerably shorten the list of specified packages in the `config.xml` file. A named pattern, specified with the `namedCollection` element is a representation of a predefined list of packages. Specifying a pattern will install all packages listed in the named pattern to be installed in the image. Support for patterns is distribution specific and available with SLES, openSUSE, CentOS and RHEL. The optional patternType attribute on the packages element allows you to control the installation of dependent packages in the image. You may assign one of the following values to the patternType attribute:

onlyRequired
: Incorporates only patterns and packages that the specified patterns and packages require. This is a "hard dependency" only resolution.

plusRecommended
: Incorporates patterns and packages that are required and recommended by the specified patterns and packages in config.xml.

By default, only required patterns and packages are installed. KIWI depends on the package manager to resolve the specified list of patterns and packages against the specified repositories and complete the installation. Note that not all supported package managers support the use of named patterns, thus the value of the packageManager element determines whether you are able to use named patterns or not. Should the list of specified packages result in a conflict the image creation process will stop and the information provided by the package manager will be captured in the build log and will be displayed in the terminal window where KIWI was started. The ignore element may be of use in resolving such conflicts. However, the ignore element is limited to effect packages named explicitly. Packages installed in the image through a named pattern are not effected by the ignore element setting. Therefore, package conflicts created by packages within named patterns cannot be resolved using the ignore mechanism. Further, if a package is specified to be ignored, but is required by another package, then the required package is installed in the image via the automatic dependency resolution by the package manager in use.

4.1.8.2. Architecture Restrictions

To restrict a package to a specific architecture, use the arch attribute to specify a comma separated list of allowed architectures. Such a package is only installed if the build systems architecture (**uname** -m) matches one of the specified values of the arch attribute.

4.1.8.3. Packages to Become Included Into the Boot Image

The optional attributes bootinclude and bootdelete can be used to mark a package inside the system image description to become part of the corresponding boot image (initrd). This feature is most often used to specify bootsplash and/or graphics boot related packages inside the system image description but they are required to be part of the boot image as the data is used at boot time of the image.

Packages included into the boot image with the bootinclude are still included into the system image as well. If packages should only be included into the boot image, but not the system image, they need to be added to the packages section of type=delete.

If the bootdelete attribute is specified along with the bootinclude attribute this means that the selected package will be marked as a “to become deleted” package and is removed by the contents of the **images.sh** script of the corresponding boot image description.

4.1.8.4. Data not Available as Packages to Become Included

With the optional archive element it’s possible to include any kind of data into the image. The archive elements expects the name of a tarball which must exist as part of the system image description. KIWI then picks up the tarball and installs it into the image. If the bootinclude attribute is set along with the archive element the data will also become installed into the boot image.

5 Advanced Configuration

Table of Contents

In this chapter you will learn how to speed up image rebuilds by using images caches. It also deals with setting up images supporting complex storage scenarios such as RAID, LVM and encrypted partitions.

5.1. Image Caches

The process of creating an appliance could take quite some time and often the same software is installed over and over again. To speed up that process KIWI can create and re-use so called image caches. An image cache in KIWI is a partial root tree created from a cache image description.

Figure 5.1. Image Caching Architecture

A cache needs to be created before it can be used. This can be done using any standard kiwi image description, including boot image descriptions. That means you can simply use one of the template or *boot descriptions and create a cache from it. However, it is more efficient to create image descriptions for the sole purpose of caching. Such descriptions could represent a set of patterns for example. The less specific a cache is the more often it can be re-used

Once there are caches in the system KIWI selects the best match and mounts the cache in a way that all write actions (copy-on-write cache) are redirected to the new root system. That way the cache itself is never changed and can be re-used simultaneously for other build processes. As a result the build process does not start with an empty tree but with a tree filled with the contents of the cache. Only the missing parts need to be added, which speeds up the build considerably.

Example 5.1. Building Multiple VMX Images

Assume we want to build some images of type 'vmx' based on the SLES 12 JeOS image description. Create image caches for the system and the boot image like follows:

1. Build the boot image (initrd) cache:

```
kiwi --init-cache /usr/share/kiwi/image/vmxboot/suse-SLES12
```

2. Build the JeOS image cache:

   ```
   kiwi --init-cache /usr/share/kiwi/image/suse-SLE12-JeOS/
   ```

By default those caches will be created in `/var/cache/kiwi-images`. To run a build which uses caches, run the following command;

```
kiwi --build suse-SLE12-JeOS -d /tmp/myimage --type vmx \
     --cache /var/cache/kiwi-images
```

This call speeds up the build a lot compared to building without caches. It is important to understand that a cache based build will create a root tree which contains only the differences compared to the used cache. Thus at any time you want to create an image out of it you need to make sure that the cache exists and is accessible on the system.

5.2. KIWI RAID Support

KIWI supports three types of RAID systems:

Real RAID Controllers with Their Own Firmware
: KIWI only needs to make sure the drivers are part of the initrd e.g cciss for the smart array controllers built into some server boards.

BIOS RAID Controllers
: Cheap onboard controller devices with the RAID software inside the BIOS (so called fake RAID). Linux supports some of them with the 'dmraid' utility and the support is a mix of BIOS calls and some device mapper calls.

  The check for these devices can be switched on and off with <oem-ataraid-scan>true|false</oem-ataraid-scan>

Linux Software RAID
: There is no hardware involved. The Linux kernel can control any storage device by adding RAID capabilities. All the work done by a real hardware controller is done in software.

  All this is done using the 'mdadm' utility. The metadata for the devices are stored in RAID blocks on the storage device which requires them to be of the correct partition type.

  The software RAID is supported in a so called degraded mode. This means the RAID is created but not all devices to build it are attached. That is because an image initially only consists of a single disk. The user needs to add devices or change the RAID mode manually after deployment. This is an easy task if the system comes up prepared accordingly. To use Linux software raid in KIWI images you only need to set:

  <type ... mdraid="mirroring">

  Currently kiwi supports a degraded mirroring (raid:1) or stripping (raid:0) configuratiom but you can change the mode to any supported raid level after deployment.

5.3. KIWI Custom Partitions

KIWI supports custom partitioning via LVM, the logical volume manager for the Linux kernel, or on file systems with volume support like Btrfs or ZFS.

5.3.1. Custom Partitioning via LVM

To define an LVM volume, a `systemdisk` element within the `type` element in the `config.xml` file must be defined. The `systemdisk` element has an optional attribute `name`, which specifies the volume group name.

For additional non root or swap volumes the `systemdisk` element can contain the child element `volume`, with four possible attributes:

`name`
: A required attribute. The name of the volume. If `mountpoint` is not specified, a directory with the given name will be created by KIWI if it does not already exist inside the root tree. However, if the name contains the KIWI internal path field separator “_”, it is required to specify the path in an additional mount point attribute. The special value *@root* can be combined with the `size` or `freespace` attribute to control the size of the root volume.

`size`
: An optional attribute. Absolute size of the volume. If the size value is too small to store all data kiwi will exit. If no suffix is used, the value will be considered as Megabytes, otherwise add `M` (Megabyte) or `G` (Gigabyte) as suffix.

`freespace`
: An optional attribute. Free space to be added to this volume. If no suffix is used, the value will be considered as Megabytes, otherwise add `M` (Megabyte) or `G` (Gigabyte) as suffix.

`mountpoint`
: An optional attribute. Specifies a path which needs to exist inside the root directory.

Example 5.2. Examples for Configuring LVM

The following example will create a logical volume named `LVtmp` with minimal size to store the content of `/tmp` of the image at build time. The volume is mounted to `/tmp`:

```
<image ...>
  <preferences>
    <type ...>
      <systemdisk name="vgroup-name">
        <volume name="tmp"/>
      </systemdisk>
      ...
    </type>
    ...
  </preferences>
  ...
</image>
```

To set the volume size to 200 MB use:

```
<image ...>
  <preferences>
    <type ...>
      <systemdisk name="vgroup-name">
        <volume name="tmp" size="200M"/>
      </systemdisk>
      ...
    </type>
    ...
  </preferences>
  ...
</image>
```

To create the logical volume named `foo` with 500 MB of free space mounted as `/tmp`, use:

```
<image ...>
  <preferences>
    <type ...>
      <systemdisk name="vgroup-name">
        <volume name="tmp" freespace="500M" mountpoint="tmp"/>
      </systemdisk>
      ...
    </type>
    ...
  </preferences>
  ...
</image>
```

The volumes `LVRoot` and `LVSwap` for the root file system and for swap, will always be generated. To control the size if `LVRoot`, use the special name `@root`.

```
<image ...>
  <preferences>
    <type ...>
      <systemdisk name="vgroup-name">
        <volume name="@root" size="2M"/>
      </systemdisk>
      ...
    </type>
    ...
  </preferences>
  ...
</image>
```

5.3.2. Custom Partitioning via Btrfs

If Btrfs is used as a file system, the subvolume management is configured via the same `systemdisk` element as explained in Section 5.3.1, “Custom Partitioning via LVM”. Also the same rules as explained for lvm volumes applies to Btrfs subvolumes with the following exception;

There is no @root volume and no size setup. The Btrfs file system is created with an initial size which can be specified by the `size` element All subvolumes are part of the file system itself and managed by a namespace. The overall size is shared across the entire file system and the size of an entity can be controlled by a Btrfs quota which is not applied by kiwi at the moment

5.4. KIWI Encryption Support

KIWI supports Linux Unified Key Setup (LUKS) encrypted images. To set up an encrypted volume , add the attribute `luks` to the `type` element in `config.xml`. The value of the attribute represents the password string which will be required to mount the file system while booting:

```
<image ...>
  <preferences>
    <type ... luks="password"/>
  </preferences>
  ...
</image>
```

6 Maintaining Appliance Images

Table of Contents

Whenever you create an appliance as described in Chapter 14, *Creating Appliances*, you are using a snapshot of the software repositories. As time goes by, the software continues to develop and the image becomes outdated. Something similar applies to the appliance configuration, which may become outdated when, for example, the network setup has changed or services have been replaced. Last, it may be necessary to update the image description itself, for example to add or change repositories. To prevent an image from becoming outdated, KIWI provides means to *maintain* existing images.

kiwi --upgrade
: The `--upgrade` option can be used to update a previously built image without making any changes to the image description. It requires an existing unpacked root tree. This option can be used to update software packages within an image. Since it only modifies an existing root tree, it is faster than re-building an image from scratch.

kiwi --prepare
: In case the appliance configuration or the image description needs to be changed, the image root tree needs to be rebuilt with `--prepare` to make these changes permanent. We using this method to upgrade an existing image, it is recommended to put the image description under a version control system such as git or subversion. This allows to track the changes and to easily switch between image versions.

6.1. Image Maintenance: Updating Software Packages

The quickest way to produce an updated image containing the latest software versions from the repositories configured in the image description is to use KIWI's `--upgrade` option. It optionally allows to specify additional repositories (such as an update repository) or packages on the command line. Using `--upgrade` requires an existing unpacked image tree from a previous run of **kiwi --prepare**.

1. Make sure you know the path to the existing unpacked root tree. In the following we assume it is located at `/tmp/myISO`. If you need to add additional repositories, note down the paths and types.

2. Run the following command to update the unpacked root tree from the repositories configured in the image description:

   ```
   kiwi --upgrade /tmp/myISO
   ```

   You can optionally specify additional repositories not yet configured in the image description, or additional packages and/or patterns. The latter two can be part of the added repositories or the ones already configured.

   ```
   kiwi --upgrade /tmp/myISO --add-repo REPO --add-repotype REPO_TYPE \
   --add-package PACKAGE --add-pattern PATTERN
   ```

3. Run **kiwi --create** to build the final image. The following example assumes that the image is created in `/tmp`:

   ```
   kiwi --create /tmp/myISO --destdir /tmp
   ```

Adding Repositories, Packages or Patterns

Repositories, packages or patterns that you specify with the --add-* commands are not added to the image description! Therefore these additions are not permanent and need to be specified again with the next run. If you want to make these additions permanent, proceed as described in Section 6.2, “Image Maintenance: Modifying the Configuration”.

Configuration Changes

Running KIWI with the `--upgrade` option does not rebuild the root tree—it only manipulates the existing version that was created with the last `--prepare`-run. Changes to the image description that have been made after the last `--prepare`-run, have no effect when running the `--upgrade` option.

In case of un-applied configuration changes, rebuild the unpacked root tree from scratch with **kiwi --prepare**.

6.2. Image Maintenance: Modifying the Configuration

In case you need to change the configuration of the appliance (see Section 14.1, “The KIWI Model”) or the image description (see Chapter 4, *KIWI Image Description* you need to rebuild the complete image tree by running **kiwi --prepare**, to make your changes permanent. It is strongly recommended to put your image description under a version control system to enable tracking of changes and different version builds.

1. Put your image description directory under a version control system such as git. This step is optional, but strongly recommended.

2. Change the image description as needed. Stick to the following general rules:

   a. To add packages or patterns, modify the respective entries in `configuration.xml`. See Section 4.1.8, “`packages` Element” for details.

   b. To change the list of repositories, modify the respective entries in `configuration.xml`. See Section 4.1.7, “`repository` Element” for details.

c. To change the configuration of your appliance, modify the respective files in the overlay tree or provide an updated archive.

d. To change the user-defined tasks that will be carried out after the installation (for example activating services), adjust the `configuration.sh` script.

3. Create an unpacked root tree by running **kiwi --prepare**. The following example assumes that the image description is located under `/tmp/myISO_config` and that the result is written to `/tmp/myISO`:

```
kiwi --prepare /tmp/myISO_config --root /tmp/myISO
```

4. Run **kiwi --create** to build the final image. The following example assumes that the image is created in `/tmp`:

```
kiwi --create /tmp/myISO --destdir /tmp
```

5. When the final version of your changes has been applied, make sure to check this state in to the version control system.

Do not Manipulate the Unpacked Root Image

The unpacked image directory is a directory, as far as the build system is concerned and you can manipulate the content of this directory according to your needs. Since it represents a system installation you can “chroot” into this directory for testing purposes. The file system contains an additional directory named /image that is not present in a regular system. It contains information KIWI requires during the create step, including a copy of the config.xml file.

Do not make any changes to the system, since they will get lost when re-running the prepare step again. Whats more, you may introduce errors that will occur during the create step, that are difficult to track. The recommended way to apply changes to the unpacked image directory is to change the configuration and re-run the prepare step as described above.

Part II. Usecases

Table of Contents

7 ISO Image / Live System

Table of Contents

A live system image is an operating System on CD, DVD or removable USB storage. It can bee booted directly from the media. A CD/DVD live system is read-only—all changes to the system are done in RAM will be lost as soon as the computer shuts down. A removable USB storage can optionally be set up with an additional partition (hybrid ISO image) which can be used for writing data.

ISO Image Description Templates

KIWI comes with many image description templates. It is recommended to use them as a basis for your own image descriptions. To do so, copy the respective directory containing the image description of your choice to you working directory and adjust it according to your needs.

ISO image templates are shipped with the package kiwi-desc-isoboot. They are installed to `/usr/share/kiwi/image/isoboot`.

Just Enough Operating System (JeOS) Templates

The package kiwi-templates contains ready-to-use JeOS templates. They are installed to `/usr/share/kiwi/image/*-JeOS/`. These templates can be used without any modification to build images containing a minimal operating system. The template directory is in KIWI's search path, therefore it is sufficient to only specify the name of the `*-JeOS` directory on the KIWI command line. Get a list of available templates with the following command:

```
(cd /usr/share/kiwi/image/ && ls -d1 *-JeOS)
```

7.1. Building Live CD/DVD Images

The following example shows how to build a Just enough Operating System (JeOS) based on SUSE Linux Enterprise 12:

```
kiwi --build suse-SLE12-JeOS -d /tmp/myiso-result --type iso
```

There are two possibilities to use the ISO image generated above:

1. Burn the `.iso` file on a CD or DVD with your preferred burn program. Plug in the CD or DVD into a test computer and (re)boot the machine. Make sure the computer boots from the CD drive as first boot device.

2. Use a virtualization system to test the image directly. Testing an ISO image can be done with any full virtual system for example `QEMU`:

   ```
   qemu -cdrom /tmp/myiso-result/LimeJeOS-SLE12.x86_64-1.13.1.iso
   ```

KIWI supports different flavors of file systems and boot methods along with the ISO image type. The template `suse-SLE12-JeOS` from the example above, uses an `overlayfs`-based compressed root file system. Set the file system type with the `flags` attribute in `config.xml`:

```
<image ...>
  <preferences>
    <type image="iso" flags="FSTYPE" .../>
    ...
  </preferences
  ...
</image>
```

The following values can be set for *FSTYPE*:

Value	**Description**
`compressed`	Does file system compression with squashfs, but does not use an overlay file system for write support. A symbolic link list is used instead and thus a `split` element is required in `config.xml`. See Section 7.1.1, “Split mode” for details.
`overlay`	`overlayfs` allows to combine two file systems into one. The root file system exists as a compressed `squashfs` file system and all write operations are redirected to the RAM or to a persistent area on a disk. The result is a fully writable live-system.
`seed`	Creates a `Btrfs` image and allows write operations into a cow (seed) file. In case of an ISO image the seed device is created on a RAM disk.
unset	If the `flags` attribute is not set, no compressed nor overlay file system will be used. The root tree will be directly part of the ISO file system and the paths: `/bin`, `/boot`, `/lib`, `/lib64`, `/opt`, `/sbin`, and `/usr` will be read-only.

7.1.1. Split mode

If no overlay file system is in use but the image file system is based on a compressed file system, KIWI allows toc onfigure which files and directories should be writable in a so-called split section. To allow login in to the system, at least the `/var` directory needs to be writable. This is because the PAM authentication requires to be able to report any login attempt to `/var/log/messages` which therefore needs to be writable. The following example ensures that `/boot`, `/etc`, `/home`, and `/var` are writable:

```
<image ...>
  ...
  <preferences>
    <type image="iso" flags="compressed" .../>
    <split>
      <persistent>
        <file name="/var"/>
        <file name="/var/*"/>
```

```
        <file name="/boot"/>
        <file name="/boot/*"/>
        <file name="/etc"/>
        <file name="/etc/*"/>
        <file name="/home"/>
        <file name="/home/*"/>
        <file name="/tmp"/>
        <file name="/tmp/*"/>
      </persistent>
    </split>
    ...
  </type>
</image>
```

7.1.2. Hybrid mode

A hybrid image is an ISO image including a partition table. therefore it cab be attached as a CD/DVD *and* as a normal disk to the system. This has the advantage that a hybrid ISO live system can be burned to a CD/DVD and uploaded to a flash disk. To activate the hybrid feature the `hybrid` attribute must be set to `true` as follows:

```
<image ...>
  <preferences>
    <type image="iso" hybrid="true" .../>
    ...
  </preferences
  ...
</image>
```

7.2. Building Live Images for Removable USB Devices

KIWI supports two types of images for removable USB devices. Hybrid ISO images are the same as the live CD/DVD images. The second type are OEM virtual disk images. The deployment of both types can be performed from any OS including Windows as long as a tool to dump data onto a disk device exists and is used.

7.2.1. Hybrid ISO Image

As indicated above a hybrid ISO image also works as an image for removable USB devices. If a hybrid ISO image is used like a disk image on a writable medium like a flash disk, it is possible to write into a persistent area on the stick instead of the RAM. KIWI creates an additional Ext2 partition for this purpose if the attribute `hybridpersistent` is set to `true`.

```
<image ...>
  <preferences>
    <type image="iso" hybrid="true" .../>
    ...
  </preferences
  ...
</image><
```

7.2.2. In RAM ISO Image

Any live CD iso image supports the *`toram`* mode. In this mode the contents on the ISO are copied into a tmpfs on the system. In the standard live image mode the ISO is only mounted to the system. Reading the data from a tmpfs is however much faster than reading the data from CD/DVD drive or flash disk. An additional benefit of the toram mode is the ability to disconnect the CD/DVD drive or USB slot the live system originally existed. However the costs for the performance win are a slower initial start-up time and more RAM requirements. A minimum 2GB of RAM is recommended to boot 700M compressed live image. To activate the toram mode place the following option at the boot prompt when the system starts up.

```
toram
```

7.2.3. OEM Virtual Disk Image

In contrast to the hybrid ISO image it is also possible to create an OEM virtual disk image. This image type allows to create the live operating system plus a data partition for custom data. The data partition is a fat partition also recognized by Windows. To create such a partition use the option `--fat-storage` *`size-in-MB`* on the KIWI command line:

```
kiwi --create ... --fat-storage 500
```

If this option is set, KIWI will use the syslinux boot loader for the image and for the first FAT partition with the specified size. The live operating system itself will be hosted on a logical volume (LVM), which allows to easily manipulate the logical root volume. For further information about the OEM image type refer Chapter 12, *OEM Image / Preload Systems*.

7.2.3.1. OEM compressed / Read-only removable USB Media

If a compressed file system type like `overlayfs` is used for the image root directory, it is also possible to choose whether to write on a partition or to the RAM. KIWI provides the attribute `ramonly` for this purpose. To create a read-only image with compressed root file system for a removable USB medium the following configuration is required:

```
<image ...>
  <preferences>
    <type image="iso" file system="overlayfs" ramonly="true" .../>
    ...
  </preferences
  ...
</image>
```

8 VMX Image / Virtual Disks

Table of Contents

A VMX image is a virtual disk image for use in full virtualization systems like Qemu or VMware. The image is a file containing the system (represented by the configured packages in `config.xml`) as well as partition data and boot loader information. VMX images have a fixed disk size and geometry that cannot be changed (see Chapter 12, *OEM Image / Preload Systems* for images with expandable disks).

VMX Image Description Templates

KIWI comes with many image description templates. It is recommended to use them as a basis for your own image descriptions. To do so, copy the respective directory containing the image description of your choice to you working directory and adjust it according to your needs.

VMX image templates are shipped with the package kiwi-desc-vmxboot. They are installed to `/usr/share/kiwi/image/vmxboot`.

Just Enough Operating System (JeOS) Templates

The package kiwi-templates contains ready-to-use JeOS templates. They are installed to `/usr/share/kiwi/image/*-JeOS/`. These templates can be used without any modification to build images containing a minimal operating system. The template directory is in KIWI's search path, therefore it is sufficient to only specify the name of the `*-JeOS` directory on the KIWI command line. Get a list of available templates with the following command:

```
(cd /usr/share/kiwi/image/ && ls -d1 *-JeOS)
```

8.1. Building VMX Images

The following example shows how to build a Just enough Operating System (JeOS) based on SUSE Linux Enterprise 12:

```
kiwi --build suse-SLE12-JeOS -d /tmp/myvm-result --type vmx
```

The command creates a virtual disk in the `.raw` format that can be directly booted with any virtualization system. The following example shows how to boot it with QEMU:

```
qemu /tmp/myvm-result/LimeJeOS-SLE12.x86_64-1.13.1.raw -m 1024
```

KIWI always generates a file in the `.raw` format. The `.raw` file is a disk image with an equivalent to the structure of a physical hard disk. `.raw` images are supported by any hypervisor, but are rather big in size (not compressed) and do not offer the best performance.

Therefore each virtualization system supports its own proprietary format supporting compression and improved I/O performance. To build an image in a format other than `.raw`, add the `format` attribute to the type definition in `config.xml`:

```
<image ...>
  <preferences>
    <type format="FORMAT" .../>
    ...
  </preferences
  ...
</image>
```

The following values can be set for *FORMAT*:

Table 8.1. Supported Virtual Disk Formats

Name	Description
vmdk	Disk format for VMware
vhd|vhd-fixed	Disk format for Microsoft HyperV
ovf|ova	Open Virtual Format requires VMware's ovftool
qcow2	QEMU virtual disk format
vdi	Disk format for VirtualBox
vagrant	Vagrant Box Format
gce	Google Cloud Format

8.2. VMware support

A VMware image is accompanied by a guest configuration file. This file includes information about the hardware to be represented by the guest as well as specifications of resources such as memory.

It is possible to specify the VMware configuration settings in `config.xml`. Additionally, it is possible to also include selected packages in the created image that are specific to the VM image generation. This is done in the *machine* section in the configuration file. The following example will create a VMware guest configuration with 512 MB of RAM and an IDE disk controller:

```
<image ...>
  <preferences>
    <type format="vmdk" ...>
      <machine memory="512">
        <vmdisk controller="ide" id="0"/>
      </machine>
      ...
    </type>
    ...
  </preferences
  ...
</image>
```

The guest configuration can be loaded via the VMware user interface and may be modified through the GUI. The configuration file has the extension `.vmx` and the same basename as the image:

```
/tmp/myvm-result/LimeJeOS-SLE12.x86_64-1.13.1.vmx
```

8.3. LVM Support

Support for LVM has been added for all image types which are disk-based. In order to use LVM for the vmx type just add the `--lvm` option as part of the KIWI create/build step. Alternatively add the attribute lvm="true" as part of the type section in your `config.xml` file.

```
<image ...>
  <preferences>
    <type lvm="true" .../>
    ...
  </preferences
  ...
</image>
```

When using modern file systems like Btrfs or zfs, KIWI also supports using their native volume management system. For more information how to setup custom partitions/volumes, see Section 5.3, “KIWI Custom Partitions”.

8.4. Extra Boot Partition

Depending on the selected root file system and the capabilities of the boot loader, KIWI automatically decides whether to set up a separate boot partition. The format of a boot partition is set to ext2 by default.

To manually control the attributes `bootpartition` (can be set to `true` or `false`) and `bootfilesystem` (can be set to `ext2`, `ext3`, `ext4`, `fat32`, or `fat16`). A runtime check at build time will test whether the given configuration can be implemented. The following example will create an etx3-formatted boot partition:

```
<image ...>
  <preferences>
    <type bootpartition="true" bootfilesystem="ext3" .../>
    ...
  </preferences
  ...
</image>
```

9 Docker images

Table of Contents

Docker is a shipping container system for code that can run virtually everywhere. It is an extension of LXC's capabilities. Since Docker is based on LXC, a Docker container does not include a separate operating system. It relies on the functionality provided by the underlying infrastructure. As such, it can package the application and all its dependencies in a virtual container which can be run on any Linux server.

Docker not only makes it possible to deploy portable containers across machines. It also includes versioning capabilities for tracking different versions of a container, it allows re-using containers as a base for other specialized components, and much more. Find more information about Docker on its home page at http://www.docker.io.

Just Enough Operating System (JeOS) Templates

The package kiwi-templates contains ready-to-use JeOS templates. They are installed to `/usr/share/kiwi/image/*-JeOS/`. These templates can be used without any modification to build images containing a minimal operating system. The template directory is in KIWI's search path, therefore it is sufficient to only specify the name of the `*-JeOS` directory on the KIWI command line. Get a list of available templates with the following command:

```
(cd /usr/share/kiwi/image/ && ls -d1 *-JeOS)
```

9.1. Building Docker Images

The following example shows how to build a Just enough Operating System (JeOS) based on SUSE Linux Enterprise 12:

```
kiwi --build suse-SLE12-JeOS --add-profile docker --type docker -d /tmp/my-container
```

The image is packed into a TAR archive, `/tmp/my-container/LimeJeOS-SLE12-docker.x86_64-1.13.1.tar.xz` in this example. To use this image with Docker it must be imported via the **docker** command. The package docker needs to be installed and the daemon `dockerd` needs to run:

```
cat /tmp/my-container/LimeJeOS-SLE12-docker.x86_64-1.13.1.tar.xz |\
docker import - sle12-jeos:new
```

When imported, a container instance can be started as follows:

```
docker run --privileged=true -t -i sle12-jeos:new /bin/bash
```

Unpacking the TAR archive

LXC images created by KIWI are packed into a TAR archive and need to be unpacked at the root level (`/`) of the host system. *Never do this with a Docker TAR archive* since it would overwrite data on the host system. Always use the **docker** command as described above to import the image.

9.2. Image Configuration Details

The configuration for a container does not need to contain a kernel package. The container represents the user space that runs on top of the kernel of the container host system. However, the container itself must include the Linux user space container tools. *2015-07-29 - fs: Is this a package? What needs to be done to include these tools?*

to configure the network for the container use the `vmnic` in the `config.xml` file as shown below. The `mode` attribute configures the network mode, with *veth* being the default. *2015-07-29 - fs: which other modes are valid?*

Although it is possible to configure multiple network interfaces in the `config.xml` file, only the first one is used in the container. Prerequisite for a working network in the container is a network bridge named `br0` configured on the host system. For complex network setups is necessary to edit the configuration file for the container. *2015-07-29 - fs: How? Unpack TAR archive, edit and tar again?*

```
<image ...>
  <preferences>
    <type ...>
      <machine ...>
        <vmnic interface="0" mode="veth"/>
        ...
      </machine>
      ...
    </type>
  </preferences>
  ...
</image>
```

The generated configuration file restricts the device access of the container according to a generally accepted best practice security model. The device access permissions may be modified by editing the `config` file for the container.

10 Vagrant boxes

Table of Contents

Vagrant [http://vagrantup.com/] is a nice framework to implement consistent processing/testing work environments based on Virtualization technologies. To run a system, Vagrant needs so-called “boxes”. A box is a TAR archive containing a virtual disk image and some metadata.

To build Vagrant boxes, you can use veewee [https://github.com/jedi4ever/veewee] which builds boxes based on AutoYaST, or Packer [http://packer.io], which is provided by Vagrant itself.

Both tools are based on the official distribution media (DVDs). If such media does not exist (for example if the distribution is still under development or you want to use a collection of your own repositories), the KIWI way of building images might be helpful. In addition you can use the KIWI image description as source for the Open Build Service [http://openbuildservice.org/] which then allows building and maintaining boxes in the Build Service as a plus.

10.1. Building Vagrant Boxes

The following example shows how to build a Just enough Operating System (JeOS) based on SUSE Linux Enterprise 12:

```
kiwi --build suse-SLE12-JeOS --add-profile vagrant --type vmx -d /tmp/my-box
```

The build result is written to `/tmp/my-box`. The `.box` and `.json` files are needed to add and run the box in Vagrant. The `.box` file is a TAR archive containing the virtual disk image for the selected virtualization provider. In this example it is a qcow2 image to be used with libvirt and some metadata which mostly duplicates the information from the '.json' file to have it packaged in one place, too. *2015-07-29 - fs: not sure what the last part of the sentence is supposed to say... .*

The system installed on the virtual disk needs to fulfill some requirements which are documented at http://docs.vagrantup.com/v2/boxes/base.html. The KIWI template makes sure these requirements are met:

- installation of mandatory packages: sudo, openssh and rsync
- users `root` and `vagrant` are both configured to use `vagrant` as password

- integration of the Vagrant public SSH key from https://github.com/mitchellh/vagrant/tree/master/keys
- starting `sshd` daemon at boot time with `UseDNS` set to `no`
- sudo configured to allow passwordless root permissions for the `vagrant` user

Using the box requires a correct Vagrant installation on your machine. The `libvirtd` daemon and the libvirt default network need to be running.

Adding the box to Vagrant can be done in two ways. Either by using the `.box` file and providing a name at the command line:

```
cd /tmp/my-box
vagrant box add my-box LimeJeOS-SLE12.x86_64-1.13.1.libvirt.box
```

or by using the `.json` file to provide metadata such as a version number (similar to the boxes downloaded from https://vagrantcloud.com/):

```
cd /tmp/my-box
vagrant box add LimeJeOS-SLE12.x86_64-1.13.1.libvirt.json
```

2015-07-29 - fs: Are the cd commands really needed? What about vagrant box add /tmp/my-box/ LimeJeOS-SLE12.x86_64-1.13.1.libvirt.json ?

With either method, you can now boot the box and log in:

```
root # cd /tmp/my-box
root # vagrant init my-box
root # vagrant up --provider libvirt
root # vagrant ssh
 This is the Lime-JeOS SLE12 Linux System...
 vagrant@linux:~>
```

Vagrant with Docker

Building boxes for the libvirt, VMware or VirtualBox providers requires to build images with the disk format required by the provider. A Docker based box, however, has no such requirements. Therefore building a Docker based box for Vagrant in KIWI does not differ from building a regular Docker image as described in Chapter 9, *Docker images*.

11 PXE Image / Thin Clients

Table of Contents

PXE is a network boot protocol that is shipped with most BIOS implementations. The protocol sends a DHCP request to get an IP address. When an IP address is assigned, it uses the TFTP protocol to download a Kernel and boot instructions. Contrary to other images built with KIWI, a PXE image consists of separate boot and system images, since both images need to be made available in different locations on the network boot server.

PXE Image Description Templates

KIWI comes with many image description templates. It is recommended to use them as a basis for your own image descriptions. To do so, copy the respective directory containing the image description of your choice to you working directory and adjust it according to your needs.

PXE image templates are shipped with the package kiwi-desc-netboot. They are installed to `/usr/share/kiwi/image/netboot`.

Just Enough Operating System (JeOS) Templates

The package kiwi-templates contains ready-to-use JeOS templates. They are installed to `/usr/share/kiwi/image/*-JeOS/`. These templates can be used without any modification to build images containing a minimal operating system. The template directory is in KIWI's search path, therefore it is sufficient to only specify the name of the `*-JeOS` directory on the KIWI command line. Get a list of available templates with the following command:

```
(cd /usr/share/kiwi/image/ && ls -d1 *-JeOS)
```

11.1. Building PXE Images

The following example shows how to build a Just enough Operating System (JeOS) based on SUSE Linux Enterprise 12:

```
kiwi --build suse-SLE12-community-JeOS --add-profile netboot --type pxe -d /tmp/mypxe-result
```

This command generates a compressed root file system image which is deployed as an overlayfs-based union system. To use the image, all image parts need to be copied to a PXE boot server. If you have not set up such a server, refer to Appendix B, *Setting Up a Network Boot Server* for instructions. The following example assumes you have created the PXE image on the boot server itself (if not, use **scp** to copy the files on the remote host).

1. Change into the build directory:

   ```
   cd /tmp/mypxe-result
   ```

2. Copy the initrd and the kernel to `/srv/tftpboot/boot/initrd`:

   ```
   cp initrd-netboot-suse-SLES12.x86_64-2.1.1.gz /srv/tftpboot/boot/initrd
   cp initrd-netboot-suse-SLES12.x86_64-2.1.1.kernel /srv/tftpboot/boot/linux
   ```

3. Copy the system image and its md5 sum to `/srv/tftpboot/image`:

   ```
   cp LimeJeOS-SLE12-Community.x86_64-1.13.1 /srv/tftpboot/image
   cp LimeJeOS-SLE12-Community.x86_64-1.13.1.md5 /srv/tftpboot/image
   ```

4. Copy the image boot configuration to `/srv/tftpboot/KIWI/config.default`. See Section 11.2, “PXE Configuration Files” for details.

   ```
   cp LimeJeOS-SLE12-Community.x86_64-1.13.1.config \
    /srv/tftpboot/KIWI/config.default
   ```

5. Adjust the PXE configuration file. it controls which kernel and initrd is loaded and which kernel parameters are set. A template has been installed at `/srv/tftpboot/pxelinux.cfg/default` with the kiwi-pxeboot package. The minimal configuration required to boot the SLE 12 JeOS looks like to following:

   ```
   DEFAULT KIWI-Boot

   LABEL KIWI-Boot
       kernel boot/linux
       append initrd=boot/initrd
       IPAPPEND 1

   LABEL Local-Boot
       localboot 0
   ```

   2015-07-31 - fs: The content above differs from what is installed with kiwi-pxeboot.

6. Connect the client to the network and boot it.

11.2. PXE Configuration Files

2015-12-02 - fs: Removed everything concerning hwtype.MAC, since it did not seem relevant All PXE boot based deployment methods are controlled by configuration files located in `/srv/tftpboot/KIWI` on the PXE server. Such a configuration file can either be client-specific (`config.`*`MAC_ADDRESS`*, for example `config.00.AB.F3.11.73.C8`), group-specific (`config.`*`GROUP`*, see Section 11.4, “Hardware Grouping” for details) or generic (`config.default`). In an environment with heterogeneous clients, this allows to have a default configuration suitable for the majority of clients, to have configurations suitable for a group of clients (for example machines with similar or identical hardware) and individual configurations for selected machines.

Configuration files are assigned to a client in the following order, the first matching file is used:

1. `config.`*`MAC_ADDRESS`*
2. `config.`*`GROUP`*
3. `config.default`

When building the OEM image a template configuration file named *`IMAGE_NAME.config`* (for example `LimeJeOS-SLE12.x86_64-1.13.1.config`) is generated. Copy this file to `/srv/tftpboot/KIWI/config.default` on the PXE server and adjust it according to your needs. If you need client- or group-specific configuration files, use the final version of `config.default` as a template.

11.3. The PXE Client Configuration File Syntax

The configuration file contains data about the image and about configuration, synchronization, and partition parameters. The configuration file has got the following general format:

```
AOEROOT=device
COMBINED_IMAGE=1
CONF="src;dest;srvip;bsize;[hash],...,src;dest;srvip;bsize;[hash]"
DISK="device"
FORCE_KEXEC=1
IMAGE="device;name;version;srvip;bsize;compressed,...,"
KIWI_BOOT_TIMEOUT="seconds"
KIWI_INITRD="path-to-initrd"
KIWI_KERNEL="path-to-kernel"
KIWI_KERNEL_OPTIONS="opt1 opt2 ..."
NBDROOTNBDROOT="ip-address;export-name;device;swap-export-name;swap-device;write-export-name;wr
NFSROOT="ip-address;path"
PART="size;id;Mount,...,size;id;Mount"
RAID="raid-level;device1;device2;..."
REBOOT_IMAGE=1
RELOAD_CONFIG=1
RELOAD_IMAGE=1
UNIONFS_CONFIGURATION="rw-partition,compressed-partition,container-fs"
```

Quoting the Values

`config.`*`MAC_ADDRESS`* is sourced by the Bash, so the same quoting rules as for the Bash apply.

Not all configuration options need to be specified. The following configuration is an example for an image based on a read-write file system stored on a local disk:

```
DISK="/dev/sda"
PART="5;S;x,x;L;/"
IMAGE="/dev/sda2;suse-##.#-pxe-client.i686;1.2.8;192.168.100.2;4096"
```

Refer to the following list for details on each configuration option:

`AOEROOT`
: Mount the system image root file system remotely via AoE (ATA over Ethernet). This requires a server exporting a block device representing the root directory of the system image via the AoE subsystem. The block device could be a partition of a real or a virtual disk. To use the AoE subsystem it is recommended to install the aoetools and vblade packages from http://download.opensuse.org/repositories/server:/ltsp.

When these packages are installed, partitions can be exported with the **vbladed** command. The following example shows how to export `/dev/sdb1` via AoE with a major value of 0 and minor of 1 on the eth0 interface.:

```
vbladed 0 1 eth0 /dev/sdb1
```

To be able to use the device KIWI needs the information which AoE device contains the root file system. In this example this is the device `/dev/etherd/e0.1`:

```
AOEROOT=/dev/etherd/e0.1
```

In case of a compressed read-only image with `overlayfs`, the `AOEROOT` variable can also contain a second device for the write actions:

```
AOEROOT=/dev/etherd/e0.1,/dev/ram1
```

Writing to RAM as in the example above, is the default. You also can specify another AoE location or a local device for writing the data.

`COMBINED_IMAGE`

If set to a non-empty string, indicates that the boot and the system image need to be combined into a single bootable image. The first image defines the read-write part and the second image defines the read-only part.

`CONF`, `VENDOR_CONF`

Specifies a comma-separated list of source:target configuration file names. The `source` corresponds to the path on the TFTP server. It is downloaded to `target` on the client. The download is only done when the file is missing on the client or has a different md5-sum (in case the md5sum hash is supplied with `hash`).

To achieve this, a list of `CONF` files (and `VENDOR_CONF`) files that generated by KIWI and stored on the client (`/etc/KIWI/InstalledConfigFiles`) is compared to the `CONF` data gathered from the configuration file (for example `config.default`), if supplied.

Configuration files selected for comparison are those with same destination path (dest). If the destination path is same for more than one configuration file, only the last one is used (and `VENDOR_CONF` always take s precedence over `CONF`). By comparing configuration file lists present in the current `CONF` and `VENDOR_CONF` variables and with the local list, the following actions can result:

Table 11.1. Configuration Files Synchronization Possibilities

File from `CONF`, `VENDOR_CONF`	`InstalledConfigFiles`	Action
hash_a	hash_a	nothing, keep
hash_a	hash_b	download from server
none	hash	download from server
hash	none	download from server
none	none	nothing, keep
present	not present	download from server (regardless of hash)
not present	present	delete on client (regardless hash)

Note that actual configuration files (or their md5sum hashes) on the client machine are not tested—only data from the list file `/etc/KIWI/InstalledConfigFiles` is used to determine which files need to be synchronized. This means that actual configuration files can be altered or even be deleted without triggering any action. On the other hand, if the list file is changed or deleted, an action could be triggered although the actual configuration files have not changed.

`DISK`

Specifies the storage device. Only to be used together with `PART`, for example:

```
DISK=/dev/hda
```

`FORCE_KEXEC`

During the initial deployment process KIWI checks if the running Kernel is the same as the Kernel installed via the system image. If there is a mismatch, KIWI activates the installed kernel by calling **kexec**. Kexec is a tool to boot to another kernel from the currently running one. The system boots faster, because the hardware initialization phase and the boot loader are skipped..

If `FORCE_KEXEC` is set to a non-empty string kiwi will also perform kexec if the Kernel versions matches.

`IMAGE`

Specifies the image to be downloaded and the device for the root file system.

```
IMAGE='device;name;version;srvip;bsize;compressed'
```

The following parameters are supported:

`device`

2015-11-24 - fs: Please check whether the following is correct

The device the root file system should be installed on. Can either be a RAM disk (for example `/dev/ram1` or a block device (for example `/dev/hda2`).

When using a RAM disk, note that you cannot use `/dev/ram0`, since it is already reserved for the initial RAM disk set up by the installation system. Use `/dev/ram1` instead.

When using a block device, always make sure to also create a corresponding `PART` entry defining the partitioning. Note that a partitioning scheme defined with `PART` always defines the first partition as swap and the second partition as the root file system. Therefore `device` needs to point to the second partition (for example `/dev/hda2`).

`name`

2015-11-24 - fs: Is "##.#" used as a wild card for the version string?

File name of the image. Use the character “#” as a wild card for the version number, for example `LimeJeOS-SLE12-Community.x86_64-#.##.#`.

`version`

Version string, for example `1.13.1`.

`srvip`

Specifies the server IP address for the TFTP download. Must always be indicated, except in `PART`.

`bsize`
: Specifies the block size for the TFTP download. Must always be indicated, except in `PART`. If the block size is too small according to the maximum number of data packages (32768), linuxrc will automatically calculate a new block size for the download.

`compressed`
: Specifies if the image file on the TFTP server is compressed and handles it accordingly. To specify a compressed image download only the keyword `compressed` needs to be added. If compressed is not specified the standard download workflow is used.

  The name of the compressed image needs to contain the suffix `.gz` and the image needs to be compressed with the **gzip** tool. Using a compressed image will automatically *deactivate* the multicast download option of atftp.

```
IMAGE='/dev/sda2;suse.i686.gz;1.2.8;192.168.100.2;4096;compressed'
```

  The download will fail if you specify `compressed` and the image is not compressed. It will also fail if you don't specify `compressed` but the image is compressed.

`KIWI_BOOT_TIMEOUT`
: Specifies the number of seconds to wait at the boot loader screen when doing a local boot before booting the default boot entry. The default is 10.

`KIWI_INITRD`
: Specifies the KIWI initrd to be used for a local boot of the system. The value must be set to the name of the initrd file which is used via PXE network boot. If the standard TFTP setup suggested with the kiwi-pxeboot package is used all initrd files reside in `/var/lib/tftpboot/boot/`. However, because the TFTP server does a change root (**chroot /var/lib/tftpboot**) you need to specify the initrd file as in the following example:

```
KIWI_INITRD=/boot/name-of-initrd-file
```

`KIWI_KERNEL`
: Specifies the Kernel to be used for a local boot of the system The same path rules as described for `KIWI_INITRD` apply for the kernel setup:

```
KIWI_KERNEL=/boot/name-of-kernel-file
```

`KIWI_KERNEL_OPTIONS`
: Specifies additional command line options to be passed to the Kernel when booting from disk. For instance, to enable a splash screen, you might use

```
KIWI_KERNEL_OPTIONS="vga=0x317 splash=silent"
```

`NBDROOT`
: Mount the system image root file system remotely via NBD (Network Block Device). This requires a server which exports the root directory of the system image via a specified export name. The Kernel provides the block layer, together with a remote port that uses the nbd-server program. For more information on how to set up the server, see **man 1 nbd-server**. The Kernel on the remote client can set up a special network block device named `/dev/nb0` using the nbd-client command. After this device exists, the mount program is used to mount the root file system. To allow the KIWI boot image to use that, the following information must be provided:

```
NBDROOT=NBD.Server.IP.address;\
NBD-Export-Name;/dev/NBD-Device;\
NBD-Swap-Export-Name;/dev/NBD-Swap-Device;\
NBD-Write-Export-Name;/dev/NBD-Write-Device
```

The server IP and the export name are mandatory parameters, all other parameters are optional. The default device names are:

- NBD-Device: `/dev/nbd0`,
- NBD-Swap-Device: `/dev/nbd1`
- NBD-Write-Device: `/dev/ram1`

Defining a swap device is optional. It is only set up if the client has less than 48 MB of RAM. The optional NBD-Write-Export-Name and NBD-Write-Device define a write copy-on-write (COW) location for the root file system. A separate write device is only used together with a union setup based on, for example, overlayfs.

`NFSROOT`

Mount the system image root file system remotely via NFS (Network File System). This requires a server which exports the root file system of the network client in such a way that the client can mount it read/write. To do that, the boot image must know the server IP address and the path name where the root directory exists on this server. The information must be provided like follows:

```
NFSROOT=NFS.Server.IP.address;/path/to/root/tree
```

`PART`

Specifies the partitioning data. Comma-separated blocks contain size, partition type and mount point:

```
PART='size;type;mountpoint,...,size;type;mountpoint'
```

In addition, the following rules apply:

- *`size`* is measured in megabytes (MB).
- *`type`* can be one of the following values:

  `S` or `82`: swap partition
  `L` or `83`: linux partition
  `V` or `8e`: LVM partition
  `fd`: RAID partition
  `41`: prep-partition for IBM POWER

- The first element of the list must define the swap partition.

  The swap partition must not contain a mount point. A lowercase letter `x` needs to be set instead.

- The second element of the list must define the root partition.
- If a partition should take all the space left on a disk one can set a lower `x` letter as size specification.

The following example defines a 2 GB swap partition (`2000;S;x`) and a root partition occupying the remaining space on the disk (`x;L;/`):

```
PART='2000;S;x,x;L;/'
```

RAID
: In addition to the PART line it is also allowed to add a raid array setup. The first parameter of the RAID line is the raid level. So far only raid1 (mirroring) is supported. The second and third parameter specifies the raid disk devices which make up the array. If a RAID line is present all partitions in PART will be created as raid partitions. The first raid is named md0 the second one md1 and so on. It is required to specify the correct raid partition in the IMAGE line according to the PART setup. A typical raid image setup could look like this:

```
DISK=/dev/sda
RAID='1;/dev/sda;/dev/sdb'
IMAGE='/dev/md1;LimeJeOS-openSUSE-##.#.i686;1.11.3;192.168.100.2;4096'
PART='5;S;x,2000;L;/'
```

REBOOT_IMAGE
: If set to a non-empty string, this will reboot the system after the initial deployment process is done. This means the system is rebooted before the system init process is activated. If the machine's default boot setup is to boot via PXE it will again boot from the network.

RELOAD_CONFIG
: If set to a non-empty string, this forces all configuration files to be loaded from the server. The primary purpose of this setting is to aid debugging. The option only applies to disk-based systems.

RELOAD_IMAGE
: If set to a non-empty string, this forces the image to be loaded from the server even if the image on the disk is up-to-date. The primary purpose of this setting is to aid debugging. The option only applies to disk-based systems.

UNIONFS_CONFIG
: Netboot images may use overlayfs as a container file system in combination with a compressed system image. The recommended compressed file system type for the system image is **overlayfs**.

```
UNIONFS_CONFIG=/dev/sda2,/dev/sda3,overlayfs
```

In this example the first device /dev/sda2 represents the read/write file system and the second device /dev/sda3 represents the compressed system image file system.

The union file system overlayfs is then used to cover the read/write layer with the read-only device to one read/write file system. If a file on the read-only device is going to be written, the changed inodes are part of the read/write file system. Note the device specifications in UNIONFS_CONFIG must correspond to the IMAGE and PART information. The following example should explain the interconnections:

```
DISK=/dev/sda
IMAGE='/dev/sda3;image/myImage;1.1.1;192.168.1.1;4096'
PART='200;S;x,300;L;/,x;L;x'
UNIONFS_CONFIG=/dev/sda2,/dev/sda3,overlayfs
```

As the second element of the PART list must define the root partition it is absolutely important that the first device in UNIONFS_CONFIG matches this device as read/write device. The second device of UNIONFS_CONFIG needs to match the given IMAGE device name.

11.3.1. Use a Different Download Protocol

By default all downloads controlled by the KIWI linuxrc code are performed by an atftp call using the TFTP protocol. With PXE the download protocol is fixed and thus you cannott change the way how the kernel and the boot image (initrd) is downloaded. As soon as Linux takes over, the download protocols HTTP, HTTPS and FTP are supported too. KIWI uses of the **curl** program to support the additional protocols.

To select one of the additional download protocols the following kernel parameters need to be specified ion `/srv/tftpboot/pxelinux.cfg/default`:

`kiwiserver`
: Name or IP address of the server who implements the protocol

`kiwiservertype`
: Name of the download protocol which could be one of `http`, `https` or `ftp`

To set up this parameters edit the file `/srv/tftpboot/pxelinux.cfg/default` on your PXE boot server and change the append line accordingly. Note that all downloads except for kernel and initrd are now controlled by the given server and protocol. You need to make sure that this server provides the same directory and file structure as initially provided by the kiwi-pxeboot package.

11.3.2. RAM Only Image

If there is no local or remote storage for mounting the root file system, the image can be stored into the main memory of the client. The machine needs to be equipped with sufficient memory to host the RAM disk and provide enough additional memory for system operation. Set up the machine similar to the following example:

1. Use a read-write file system in `config.xml`, for example `filesystem="ext3"`
2. Create `config.`*`MAC`*

   ```
   IMAGE='/dev/ram1;suse-##.#-pxe-client.i686;1.2.8;192.168.100.2;4096'
   ```

11.3.3. Union Image

As shown in `UNIONFS_CONFIG` it is possible to use the union file system with overlayfs, to combine two file systems into one. In case of thin clients there is often the need for a compressed file system because of space limitations. However, all supported compressed file systems only allow read-only access. Combining a read-only file system with a read-write file system is a solution for this problem. KIWI uses squashfs compressing file systems. To create an image with a compressed root file system, make sure the file system attribute in `config.xml` contains `squashfs`. *2015-11-25 - fs: where should this attribute be set? A link to an example or an example right here is needed.*

When an image contains a compressed root file system, it can either be downloaded to the local machine or be mounted remotely. The following setups can be configured:

- Example 11.1, “ Download Compressed Image to Local Storage, Write to Local Storage ”
- Example 11.2, “ Download Compressed Image to Local Storage, Write to RAM ”
- Example 11.3, “ Mount Compressed Image from Remote, Write to Local Storage ”
- Example 11.4, “ Mount Compressed Image from Remote, Write to RAM ”
- Example 11.5, “ Mount Compressed Image from Remote, Write to Remote ”

Example 11.1. Download Compressed Image to Local Storage, Write to Local Storage

```
DISK=/dev/sda
PART='5;S;x,400;L;/,x;L;x'
IMAGE='/dev/sda2;suse-##.#-pxe-client.i386;1.2.8;192.168.100.2;4096'
UNIONFS_CONFIG=/dev/sda3,/dev/sda2,overlayfs
KIWI_INITRD=/boot/initrd
```

Example 11.2. Download Compressed Image to Local Storage, Write to RAM

```
DISK=/dev/sda
PART='5;S;x,400;L;/'
IMAGE='/dev/sda2;suse-##.#-pxe-client.i386;1.2.8;192.168.100.2;4096'
UNIONFS_CONFIG=tmpfs,/dev/sda2,overlayfs
```

Example 11.3. Mount Compressed Image from Remote, Write to Local Storage

Depending on whether the remote image is served via AoE, NBD, or NFS, the configuration differs:

AoE

```
PART='5;S;x,x;L;x'
AOEROOT=/dev/etherd/e0.1,/dev/sda2
UNIONFS_CONFIG=/dev/sda2,aoe,overlayfs
```

NBD

```
PART='5;S;x,x;L;x'
NBDROOT=192.168.100.7;root1;/dev/nbd0;;;;/dev/sda2
UNIONFS_CONFIG=/dev/sda2,nbd,overlayfs
```

NFS

```
PART='5;S;x,x;L;x'
NFSROOT="192.168.100.2;/srv/kiwi-read-only-path"
UNIONFS_CONFIG=/dev/sda2,nfs,overlayfs
```

Example 11.4. Mount Compressed Image from Remote, Write to RAM

Depending on whether the remote image is served via AoE, NBD, or NFS, the configuration differs:

AoE

```
AOEROOT=/dev/etherd/e0.1
UNIONFS_CONFIG=tmpfs,aoe,overlayfs
```

NBD

```
NBDROOT=192.168.100.7;root1;/dev/nbd0
UNIONFS_CONFIG=tmpfs,nbd,overlayfs
```

NFS

```
NFSROOT="192.168.100.2;/srv/kiwi-read-only-path"
UNIONFS_CONFIG=tmpfs,nfs,overlayfs
```

Example 11.5. Mount Compressed Image from Remote, Write to Remote

AoE

```
AOEROOT=/dev/etherd/e0.1,/dev/etherd/e1.1
UNIONFS_CONFIG=aoe,aoe,overlayfs
```

NBD

```
NBDROOT=192.168.100.7;root1;/dev/nbd0;swap1;/dev/nbd1;write1;/dev/nbd2
UNIONFS_CONFIG=nbd,nbd,overlayfs
```

NFS

```
NFSROOT="192.168.100.2;/srv/kiwi-read-only-path"
UNIONFS_CONFIG=/srv/kiwi-read-write-path,nfs,overlayfs
```

Check Remote Access

It is recommended to check the accessibility of the read and, if applicable, the read-write devices from a client machine in the same network. If data can be read from and, if applicable, written to these devices, the image should also be able to access these devices when booting. If the PXE boot fails, device accessibility problems can be ruled out upfront.

11.3.4. Split Image

An alternative to a Union Image (see Section 11.3.3, “Union Image”)is a split image that combines the read and read-write partitions with the `COMBINED_IMAGE` method. This allows to use different file systems without the need for an overlay file system.

Edit `config.xml` and add a split type plus a split section describing the `temporary` and `persistent` parts:

```
<type fsreadonly="squashfs"
   image="split" fsreadwrite="ext3" boot="netboot/suse-..."/>
  <split>
    <temporary>
      <!-- allow RAM read/write access to: -->
      <file name="/mnt"/>
      <file name="/mnt/*"/>
    </temporary>
    <persistent>
      <!-- allow DISK read/write access to: -->
      <file name="/var"/>
      <file name="/var/*"/>
      <file name="/boot"/>
      <file name="/boot/*"/>
      <file name="/etc"/>
      <file name="/etc/*"/>
      <file name="/home"/>
      <file name="/home/*"/>
    </persistent>
  </split>
  ...
```

```
</type>
```

Create a `config.MAC` file similar to the following::

```
IMAGE='/dev/sda2;suse-##.#-pxe-client.i686;1.2.8;192.168.100.2;4096,\
      /dev/sda3;suse-##.#-pxe-client-read-write.i686;1.2.8;192.168.100.2;4096'
PART='200;S;x,500;L;/,x;L'
DISK=/dev/sda
COMBINED_IMAGE=yes
KIWI_INITRD=/boot/initrd
```

11.3.5. Mounting the Root File System from a Remote Server

Instead of installing the image on a local storage device, it is also possible to mount the root file system remotely via AoE, NBD, and NFS.

Example 11.6. Root Tree Over AoE

Use the **vbladed** command on the remote server to bind a block device to an Ethernet interface. The block device can be a disk partition or a loop device (losetup) but not a directory. For example:

```
vbladed 0 1 eth0 blockdevice
```

Create a `config.MAC` pointing to the exported AoE device. For the example above, this would be:

```
AOEROOT=/dev/etherd/e0.1
```

Example 11.7. Root Tree Over NBD

Export the KIWI prepared tree on the NBD server and use a `config.MAC` file similar to the following example:

```
NBDROOT=192.168.100.7;root1;/dev/nbd0
```

Example 11.8. Root Tree Over NFS

Export the KIWI prepared tree via NFS and use a `config.MAC` file similar to the following example:

```
NFSROOT=192.168.100.7;/tmp/kiwi.nfsroot
```

11.4. Hardware Grouping

As explained in the section Section 11.2, “PXE Configuration Files”, three different types of configuration files containing image and deployment information exist: generic, client-specific and group-specific. This section explains how to set up groups and group-specific configuration files.

Creating groups is useful if you have a subgroup of identical or similar clients that will use the same configuration file in an otherwise heterogeneous group of clients. Instead of creating and maintaining multiple `config.MAC_ADDRESS` with identical content, you create a single `config.GROUP` file.

To add one or more groups requires to create a group definition file (`config.group` defining the groups and the clients belonging to each group, and a configuration file `config.`*`GROUP`* for each group containing image and deployment information.

11.4.1. The Group Definition File

The group definition file defines one or more groups and assigns client machines by MAC address to these groups. The file is name `config.group` and needs to be placed in `/srv/tftpboot/KIWI/` on the PXE boot server.

The following example `config.group` defines three groups *`group1`*, *`group2`*, and *`group3`* and assigns two client machines to each group:

```
KIWI_GROUP="group1, group2, group3"

group1_KIWI_MAC_LIST="11:11:11:11:11:11, 00:11:00:11:22:CA"
"group2_KIWI_MAC_LIST="00:22:00:44:00:4D, 99:3F:21:A2:F4:32"
"group3_KIWI_MAC_LIST="00:54:33:FA:44:33, 84:3D:45:2F:5F:33"
```

The following parameters can be set in `config.group`:

`KIWI_GROUP`
: Contains a list of groups that should be defined. At least one group name needs to be specified. The names are separated by a comma and a space. Although there is no limit for the number of groups, it should be kept to a minimum for reasonable manageability.

  Valid group names are made up of uppercase and lowercase letters, and may use numeric, and underscore characters. The same rules used to define Bash or sh variable names apply. The following example contains valid names:

  ```
  KIWI_GROUP="group1, group2, group3"
  ```

`GROUP_NAME``_KIWI_MAC_LIST`
: This parameter is used to assign MAC addresses to the groups defined with `KIWI_GROUP`. The name of this parameter depends on the group it represents. In our example the groups `group1, group2, group3` are defined, so the corresponding parameters are:

  ```
  group1_KIWI_MAC_LIST
  group2_KIWI_MAC_LIST
  group3_KIWI_MAC_LIST
  ```

  These parameters contain a comma-separated list of MAC addresses that should be assigned to the specified group. MAC addresses always need to be specified with uppercase letters, otherwise they will not match. If the list of addresses is very long (several thousand entries), the client's boot process my be slowed down.

11.4.2. The Group Configuration File

In addition to the group definition file `config.group`, each defined group requires a `config.`*`GROUP_NAME`* file containing the image and deployment information. Content-wise this file is identical to the `config.default` or `config.`*`MAC_ADDRESS`*. See Section 11.3, “The PXE Client Configuration File Syntax” for details. These files need to be placed in `/srv/tftpboot/KIWI/` on the PXE server. For our example, the following files would be needed:

```
/srv/tftpboot/KIWI/config.group1
/srv/tftpboot/KIWI/config.group2
```

```
/srv/tftpboot/KIWI/config.group3
```

11.4.2.1. Hardware-Specific Configuration Files

Some scenarios may require to provide different system configuration files to clients belonging to the same hardware group: If grouping clients with similar hardware, but for example different graphic cards, it may be necessary to provide graphic card specific configuration files. If setting up a PXE server supporting different locations within your organization, you may want to deliver country specific configurations for the system language and the keyboard layout.

The concept is similar to the one used with the group definition a configuration files. A list of MAC addresses that will receive individual configuration files is specified with two hardware mapping element in the group configuration file. A list of system configuration files for each hardware mapping element is specified in a hardware mapping configuration file.

Using hardware specific configuration files within a group is optional. It allows to use a single group configuration file, but to provide additional configuration files that will override the defaults provided by the `CONF` parameter. The same functionality could theoretically also be achieved by specifying different groups using different CONF parameters, but that would be less flexible and produce an unnecessary maintenance overhead.

11.4.2.1.1. The Hardware Mapping Elements

To use the hardware linking mechanism, two additional elements needs to be added to the group details file (`config.`*`GROUP_NAME`*. These two elements link hardware specific configurations to the appropriate systems by MAC address. A general example would look like this:

```
HARDWARE_MAP="VENDORNAME_MODEL"
VENDORNAME_MODEL_HARDWARE_MAP="MAC1MAC2"
```

The following parameters can be set in `config.group`:

`HARDWARE_MAP`
: This element follows the same rules as defined by the KIWI_GROUP element. However, this variable will create sub-groups used to ensure multiple types of hardware vendors can be used within the same group. The name of the group(s) should be clearly defined, and a good convention to follow would be to use a combination of the vendor name with the model number or type. This would allow for cases where the same vendor is used, but differences between alternative models requires different maps to be used.

```
HARDWARE_MAP="myvendor_foo1000 myvendor_foo2000"
```

`HARDWARE_MAP_NAME``_HARDWARE_MAP`
: This element behaves identical to the *`GROUP_NAME`*_KIWI_MAC_LIST element. It lists all MAC addresses that need to be linked to a hardware map. Any host defined within the list will receive configuration files that have been specifically defined in a hardware_config.*`HARDWARE_MAP`* file (in addition to any files defined within a CONF element).

```
myvendor_foo1000_HARDWARE_MAP="11:11:11:11:11:11"
myvendor_foo2000_HARDWARE_MAP="00:11:00:11:22:CA"
```

11.4.2.1.2. The Hardware Mapping Configuration File

When the hardware map has been defined, the last step is to ensure configuration specific elements are linked to the host(s) in question. This is done by creating a new hardware_config.*hardware_map* file. The content of the file only contains one element VENDOR_CONF:

```
VENDOR_CONF='CONFIGURATIONS/xorg.conf.hardware_name_model;/etc/X11/xorg.conf;192.168.100.2;40
```

The format of the VENDOR_CONF values is identical to the CONF variable used in the standard host and group configurations (see Section 11.3, “The PXE Client Configuration File Syntax”). In addition, files defined within this list will over-write any files defined in the group configuration, if and only if, the following requirements are met:

- The host is assigned to the current hardware map
- The file is defined within the CONF *and* VENDOR_CONF elements

NOTE: If a file is not defined in the CONF element, but is defined in the VENDOR_CONF element, it is simply downloaded to the host as if it was a CONF file. In this case, no overwriting will take place as it is considered a new file.

11.4.2.2. A Complete Example

The following is an example of a group that is using hardware from multiple vendors. For this example, lets assume the group will have 10 defined hosts, seven are *myvendor_foo1000* thin client, while the remaining three are *myvendor_foo2000* thin clients. We will also assume that the differences between the two hardware models are specific to the video card and therefore require different X drivers.

The following configuration files in `/srv/tftpboot/KIWI`are required:

```
config.group❶
config.myvendor❷
hardware_config.myvendor_foo2000❸
```

❶ Example 11.9, “The Group Definition File `config.group`”
❷ Example 11.10, “ The Group Configuration File `config.myvendor` ”
❸ Example 11.11, “ The Hardware Mapping Details File `hardware_config.myvendor_foo2000` ”

Example 11.9. The Group Definition File `config.group`

```
KIWI_GROUP="myvendor"
myvendor_KIWI_MAC_LIST=
  "00:00:00:00:00:01 00:00:00:00:00:02 \
   00:00:00:00:00:03 00:00:00:00:00:04 \
   00:00:00:00:00:05 00:00:00:00:00:06 \
   00:00:00:00:00:07 00:00:00:00:00:08
   00:00:00:00:00:09 00:00:00:00:00:0A"
```

The example group definition file contains a single group (`myvendor`) containing the ten thin clients.

Example 11.10. The Group Configuration File `config.myvendor`

```
KIWI_INITRD=/boot/initrd
KIWI_KERNEL=/boot/linux
DISK=/dev/sda
PART='5;S;x,769;L;/,x;L;x'
IMAGE='/dev/sda2;exmaple-kiosk-opensuse-##.#-pxe-client.i686;0.0.1;192.168.1.2;4096'
```

```
UNIONFS_CONFIG=/dev/sda3,/dev/sda2,overlayfs
RELOAD_IMAGE=yes
RELOAD_CONFIG=yes
CONF='prefs.js;/home/kioskuser/.mozilla/firefox/07xvl1ty.default/prefs.js;192.168.1.2;4096,\
xorg.conf;/etc/X11/xorg.conf;192.168.1.2;4096'❶
HARDWARE_MAP='myvendor_foo2000'❷
myvendor_foo2000_HARDWARE_MAP='00:00:00:00:00:02 00:00:00:00:00:03 00:00:00:00:00:04'❸
```

The first seven lines define a standard KIWI configuration, while the last three lines set up a hardware-specific configuration.

❶ Specifies two default configuration files that will be copied to all clients defined in this group: `prefs.js` (for Mozilla Firefox), and `xorg.conf` (for X Window).
❷ Defines the hardware map(s) (`myvendor_foo2000` in this case) that are to be used to provide overrides for the configuration files defined in the previous lines.
❸ Defines the list of hosts (by MAC address) that will receive the configuration file overrides (the three myvendor_foo2000 clients).

Example 11.11. The Hardware Mapping Details File `hardware_config.myvendor_foo2000`

```
VENDOR_CONF='xorg.conf.myvendor_foo2000;/etc/X11/xorg.conf;192.168.1.2;4096,\
someconfig.cfg;/etc/sysconfig/someconfig.cfg;192.168.1.2;4096'
```

When the VENDOR_CONF definition is used, we are telling KIWI that all files defined within this element, are specific to the hardware map they are linked to. As a result, all files listed here will be transferred to a host if, and only if, the host has been linked to the hardware map via the myvendor_foo2000_HARDWARE_MAP element. In our example the only systems that will receive the xorg.conf.myvendor_foo2000 file will be the three myvendor_foo2000 thin clients listed in the hardware map itself.

In this VENDOR_CONF element, two files are defined. An override `xorg.conf` file and an additional file called `someconfig.cfg`. `xorg.conf.myvendor_foo2000` will overwrite the `xorg.conf` file that was previously transferred via the CONF element. In addition to that, `someconfig.cfg` will be copied to the three myvendor_foo2000 thin clients.

As a result of this example, all ten thin clients will receive the `prefs.js` file defined in `CONF`. The seven myvendor_foo1000 clients will receive the `xorg.conf` defined in `CONF`, while the three myvendor_foo2000 clients will receive the specific `xorg.conf` defined in `VENDOR_CONF`. The myvendor_foo2000 clients will also get the file `someconfig.cfg`.

12 OEM Image / Preload Systems

Table of Contents

An OEM image is a virtual disk image representing all partitions and boot loader information the same way as on a physical disk. All flavors discussed previously in Chapter 8, *VMX Image / Virtual Disks* also apply to the OEM image type. Compared to the VMX image type, an OEM image comes with additional features. It can expand itself to a custom disk geometry and KIWI can create installation images which embeds the OEM image for deployment from CD/DVD/ Stick and over the network via PXE.

The basic idea behind an OEM image is to provide the virtual disk data for OEM vendors to support easy deployment of the system to physical storage media.

OEM Image Description Templates

KIWI comes with many image description templates. It is recommended to use them as a basis for your own image descriptions. To do so, copy the respective directory containing the image description of your choice to you working directory and adjust it according to your needs.

OEM image templates are shipped with the package kiwi-desc-oemboot. They are installed to `/usr/share/kiwi/image/oemboot`.

12.1. Building an OEM System and an Installation Image

The image creation process creates two images: An OEM disk image and an Installation ISO image containing the OEM disk image. The disk image can be dumped on a physical disk on the target system (using for example **dd**). The installation image can be dumped or burned to a bootable installation medium (for example a flash disk or a DVD). When a machine is booted from such a medium, an image deployment process (which can optionally be configured to run without user interaction) is started. The following example shows how to build a Just enough Operating System (JeOS) based on SUSE Linux Enterprise 12:

```
kiwi --build suse-SLE12-JeOS -d /tmp/myoem-result --type oem
```

12.2. Testing the Images

The images can be tested using virtualization software such as QEMU, VMware, or VirtualBox. The OEM disk image file can b identified by the extension `.raw`, the installation image either has the `.iso` or the `.raw.install` extension (also see Section 12.3, “Installation Image Flavors”).

To test the OEM disk image using **qemu**, run the following commands:

```
cd /tmp/myoem-result
qemu LimeJeOS-SLE12.x86_64-1.13.1.raw
```

Alternatively, use the **dd** command to dump the image onto a spare hard disk or a flash disk (this will wipe all existing data on the target device). To boot the image, select the appropriate device for booting device in the BIOS/EFI.

```
cd /tmp/myoem-result
dd if=LimeJeOS-SLE12.x86_64-1.13.1.raw of=/dev/device
```

Note, when testing an OEM image using the virtual disk image, for example the `.raw` file, the geometry of the disk image is not changed and therefore retains the disk geometry of the host system. This implies that the re-partitioning performed for a physical disk install during the OEM boot workflow will be skipped. *2015-11-30 - fs: Does this mean you are asked whether you want to perform the repartitioning or not during the initial boot process?*

The installation image can also be tested using virtualization software. Note that the hard disk will be re-partitioned in this case. The following example uses **qemu** for testing. A virtual hard disk is created with **qemu-img** prior to starting the image.

```
cd /tmp/myoem-result
qemu-img create /tmp/mydisk 20G
qemu -hda /tmp/mydisk -cdrom LimeJeOS-SLE12.x86_64-1.13.1.iso -boot d
```

12.3. Installation Image Flavors

The installation image is a bootable, self-installing image that deploys the OEM image onto the selected storage device. The installation process is a simple image dump using the **dd** command. During this process the target system remains in terminal mode.

The installation image can be created in two formats: a hybrid image suitable for CD/DVD media and flash disks and a second one suitable for a flash disks only. The latter format can be used if the BIOS/EFI does not support booting from hybrid images. The following configuration snippets show the use of the `installiso` to create a CD/DVD iso image and `installstick` attributes in `configuration.xml` to create a USB installation image format.

`installiso`
: Creates an `.iso` file which can be burned onto a CD or a DVD or dumped on a flash disk. The attribute `hybrid="true"` makes sure a hybrid iso image is created that is suitable for both purposes. A hybrid image is the recommended format for an installation image. If the target hardware is not able to boot from such an image, try the `installstick` variant described below.

```
<image ...>
```

```
  <preferences>
    ...
    <type image="name" installiso="true" hybrid="true" ....>
      ...
    </type>
  </preferences>
  ...
</image ...>
```

`installstick`
: Creates a `.raw.install` file which can be dumped (**dd**) onto a flash disk. Use this format if your machine cannot boot from a hybrid image.

```
<image ...>
  <preferences>
    ...
    <type image="name" installstick="true" ....>
      ...
    </type>
  </preferences>
  ...
</image ...>
```

12.4. Customizing the OEM Images

KIWI not only allows to customize the boot process of an OEM image by adding scripts to the image but also provides many configuration options that let you customize the boot loader, partitioning and other aspects of the image.

12.4.1. Customizing the OEM Install Process

It is possible to customize the OEM install process by providing shell scripts with the following names. For more information on how to include the scripts into the boot image and make them work in the boot code, see the chapter Section 3.2.1, “Boot Image Hook-Scripts”.

`preHWdetect.sh`
: This script is executed prior to the hardware scan on the target machine.

`postHWdetect.sh`
: This script is executed after the hardware scan on the target machine.

`preImageDump.sh`
: This script is executed immediately prior to the OEM image dump onto the target storage device.

`postImageDump.sh`
: This script is executed directly after the OEM image dump onto the target storage device when the image checksum has been successfully verified.

12.4.2. OEM Customizing Parameters

2015-12-01 - fs: oem-home is/are? missing. Any others?* All OEM customizing parameters reside in the `oemconfig` tag in `configuration.xml`:

```
<image ...>
  <preferences>
```

```
    ...
    <type image="oem" ....>
      <oemconfig>
        <oem-.../>
      </oemconfig>
      ...
    </type>
  </preferences>
  ...
</image ...>
```

`<oem-boot-title>`text`</oem-boot-title>`
: By default, the string `OEM` > will be used as the boot manager menu entry when KIWI creates the GRUB configuration during deployment. The `oem-boot-title` element allows you to set a custom name for the GRUB menu entry. This value can also be set at the boot prompt by the parameter `kiwi_oemtitle="`*`TITLE`*`"`.

`<oem-bootwait>`true|false`</oem-bootwait>`
: Specify if the system should wait for user interaction prior to continuing the initial boot process after (default value is `false`). This value can also be set at the boot prompt by the parameter `kiwi_oembootwait=`*`TRUE_OR_FALSE`*.

`<oem-inplace-recovery>`true|false`</oem-inplace-recovery>`
: Specify if the recovery archive is stored as part of the image or if the archive is to be created at the time the image is deployed to the target storage device. This value can also be set at the boot prompt by the parameter `kiwi_oemrecoveryInPlace=`*`TRUE_OR_FALSE`*.

`<oem-kiwi-initrd>`true|false`</oem-kiwi-initrd>`
: If this element is set to `true` (default value is `false`) the oemboot boot image (initrd) will *not* be replaced by the initrd created by the system. This option is useful when the system is installed on a flash disk. When booting from such a drive it is usually necessary to detect the storage location on each boot. This detection process is part of the oemboot boot image. This value can also be set at the boot prompt by the parameter `kiwi_oemkboot=`*`TRUE_OR_FALSE`*.

`<oem-partition-install>`true|false`</oem-partition-install>`
: By default (`false`), an OEM image is installed on the specified disk on the target system. During this process the disk is being overwritten and the original data is lost. Setting this parameter to `true`, installs the image into an empty partition (that is a partition without a file system). If the device already contains a swap partition, it will be used, otherwise a swap file will be created. The empty partition needs to exist prior to booting the KIWI image, otherwise the installation will fail. Setting this parameter to `true` also makes KIWI ignore any other partitioning-related setting (for example `oem-swap`). See Section 12.4.3, “Partition Based Installation” for more details. This value can also be set at the boot prompt by the parameter `kiwi_oempartition_install=`*`TRUE_OR_FALSE`*.

`<oem-reboot>`true|false`</oem-reboot>`
: If set to `true`, the system reboots after the OEM image has been deployed. By default this parameter is set to `false`. This value can also be set at the boot prompt by the parameter `kiwi_oemreboot=`*`TRUE_OR_FALSE`*.

`<oem-reboot-interactive>`true|false`</oem-reboot-interactive>`
: If set to `true`, the system reboots after the OEM image has been deployed. A message, which the user needs to confirm to start the reboot is displayed. By default this parameter is set to `false`. This value can also be set at the boot prompt by the parameter `kiwi_oemrebootinteractive=`*`TRUE_OR_FALSE`*.

`<oem-recovery>`true|false`</oem-recovery>`
: If this element is set to `true` (default value is `false`), KIWI will create a recovery archive from the prepared root tree. The archive will appear as `/recovery.tar.bz2` in the image file. During the first boot of the image a single recovery partition will be created and the recovery archive will be moved to that partition. A boot menu entry for recovery, that will restores the original root tree on the system, is created. User data stored in `/home` will not be affected by the recovery process. This value can also be set at the boot prompt by the parameter `kiwi_oemrecovery=`*`TRUE_OR_FALSE`*.

`<oem-recoveryID>`partition-id`</oem-recoveryID>`
: Specify the partition type for the recovery partition. The default is to create a Linux partition (id = 83). This value can also be set at the boot prompt by the parameter `kiwi_oemrecoveryID=`*`ID`*.

`<oem-silent-boot>`true|false`</oem-silent-boot>`
: Specify if the system should show boot messages (`false`) on the very first boot after having deployed the OEM image, or whether all boot messages should be suppressed (`true`). This value can also be set at the boot prompt by the parameter `kiwi_oemsilentboot=`*`TRUE_OR_FALSE`*.

`<oem-shutdown>`true|false`</oem-shutdown>`
: Specify if the system is to be powered down after the OEM image has been deployed (the default value is `false`). This value can also be set at the boot prompt by the parameter `kiwi_oemshutdown=`*`TRUE_OR_FALSE`*.

`<oem-shutdown-interactive>`true|false`</oem-shutdown-interactive>`
: Specify if the system is to be powered down after the OEM image has been deployed. A message, which the user needs to confirm to start the shutdown process, is displayed. This value can also be set at the boot prompt by the parameter `kiwi_oemshutdowninteractive=`*`TRUE_OR_FALSE`*.

`<oem-swap>`true|false`</oem-swap>`
: Specify if a swap partition should be created (the default is `true`). This value can also be set at the boot prompt by the parameter `kiwi_oemswap=`*`TRUE_OR_FALSE`*.

`<oem-swapsize>`size in MB`</oem-swapsize>`
: Set the size of the swap partition in megabytes. If a swap partition is to be created and the size of the swap partition is not specified with this optional element, KIWI will create a swap partition of a size equal to two times of the RAM size at initial boot time. This value can also be set at the boot prompt by the parameter `kiwi_oemswapMB=`*`SIZE_IN_MB`*.

`<oem-systemsize>`size in MB`</oem-systemsize>`
: Set the size the operating system is allowed to consume on the target disk. The size limit does not include any consideration for swap space or a recovery partition. In a setup *without* a `systemdisk` element this value specifies the size of the root partition. In a setup *including* a `systemdisk` element this value specifies the size of the LVM partition which contains all specified volumes. Thus, the sum of all specified volume sizes plus the sum of the specified free space for each volume must be smaller or equal than the size specified with the `oem-systemsize`. This value can also be set at the boot prompt by the parameter `kiwi_oemrootMB=`*`SIZE_IN_MB`*.

`<oem-unattended>`true|false`</oem-unattended>`
: If set to `true`, the image deployment is done without requiring user interaction. If the target system contains multiple disks, the first device (/dev/sda, for example) is au-

tomatically selected. This value can also be set at the boot prompt by the parameter `kiwi_oemunattended=`*`TRUE_OR_FALSE`*.

12.4.3. Partition Based Installation

The default installation method of an OEM image is to dump the entire virtual disk onto the target disk and to re-partition the disk to match the real geometry. All data that was previously stored on the disk will be erased.

Alternatively KIWI supports the installation into already existing partitions. This requires to set up *empty* (no file system) partitions prior to deploying the image. This way already existing data will not be touched. To activate the partition based install mode the following option needs to be set in `config.xml`:

```
<image ...>
  <preferences>
    ...
    <type image="oem" ....>
      <oemconfig>
        <oem-partition-install>true</oem-partition-install>
      </oemconfig>
      ...
    </type>
  </preferences>
  ...
</image ...>
```

With a partition-based installation, the setup differs from the default, disk-based installation in the following ways:

- The boot loader will be set up to boot the installed system only. If a multi-boot setup is required, it needs to be manually configured by the user after the initial boot.
- The parameter `oem-home*`, `oem-swap*`, and `oem-systemsize` for system, swap and home are ignored. In this mode KIWI will not create additional partitions. If a swap partition exists, it will automatically be used, if not, a swap-file will be created.
- There is no support for a remote (PXE) OEM installation, because KIWI needss to loop-mount the disk image and need to address specific regions inside of the image. Such operations are not implemented for remote access

12.5. Network Based Installation

Instead of manually dumping the OEM image on the target device or creating a KIWI installation CD or flash disk, the image can alternatively be downloaded from a PXE boot server over the network. This requires a PXE network boot server to be setup as explained in Chapter 11, *PXE Image / Thin Clients*. If your PXE server is running the following steps are required to set up the installation process over the network:

1. Make sure to create an installation PXE TAR archive along with your OEM image by setting the following option in `configuration.xml`:

   ```
   <image ...>
     <preferences>
       ...
       <type image="oem" installpxe="true"....>
         ...
   ```

```
    </type>
  </preferences>
  ...
</image ...>
```

2. Create the image, unpack the resulting `IMAGE_NAME.tgz` file to a temporary directory and copy the `initrd` and kernel images to the PXE server:

```

      mkdir /tmp/pxe && cd /tmp/pxe
      tar -xf PATH_TO/IMAGE_NAME.tgz
scp initrd-oemboot-*.install.* PXE_SERVER_IP:/srv/tftpboot/boot/initrd
scp initrd-oemboot-*.kernel.*  PXE_SERVER_IP:/srv/tftpboot/boot/linux
```

3. Also copy the system image and the md5 sum to the PXE boot server:

```
scp IMAGE_FILE.xz  PXE_SERVER_IP:/srv/tftpboot/image/
scp IMAGE_FILE.md5 PXE_SERVER_IP:/srv/tftpboot/image/
```

4. Copy the kernel command line parameters from `IMAGE_FILE.append`. Edit your PXE configuration (for example `pxelinux.cfg/default`) on the PXE server and add these parameters to the append line.

   Optionally the image and its md5sum can be stored on an FTP or HTTP server specified via the parameters **kiwiserver**=***IP_ADRESS*** and **kiwiservertype**=***HTTP_HTTPS_OR_FTP***. In this case make sure to copy the system image and md5 file to the correct location on the server. KIWI searches the image at *SERVER_ROOT*/image (for example http://www.example.com/image/*IMAGE_FILE*.xz). Note that initrd and Linux Kernel are always loaded via PXE.

13 Xen Para- and Full virtual Images

Table of Contents

A Xen image is a virtual disk like a vmx but with the Xen kernel installed for dom0 or para virtual guest images. For fully virtualized guest images any Kernel, for example `kernel-default`, my be used together with the Xen kernel modules.

A Xen image can only be booted on a Xen dom0 server. A Xen guest is booted via a boot infrastructure. For paravirtual images `pyGrub` or `pvGrub` can be used, while for HVM (fully virtualized) a special hvmloader is used. Xen extracts boot information from the given image and boots the guest. Depending on the guest type, also the boot loader configuration needs to be read. This puts some constraints on the configuration which are addressed by KIWI.

Xen Image Description Templates

KIWI comes with many image description templates. It is recommended to use them as a basis for your own image descriptions. To do so, copy the respective directory containing the image description of your choice to you working directory and adjust it according to your needs.

There are no special templates for Xen images. You may either use OEM or VMX image templates. OEM image templates are shipped with the package kiwi-desc-oemboot. They are installed to `/usr/share/kiwi/image/oemboot`. VMX image templates are shipped with the package kiwi-desc-vmxboot. They are installed to `/usr/share/kiwi/image/vmxboot`.

13.1. Building a Dom0 Image

The following example shows how to build a Just enough Operating System (JeOS) based on SUSE Linux Enterprise 12. The example adds a `xenFlavour` profile which builds a dom0 image for the OEM image type.

```
kiwi --build suse-SLE12-JeOS -d /tmp/myoem-result --type oem \
     --add-profile xenFlavour
```

13.2. Testing the Dom0 Image

The dom0 represents the most privileged layer with access to the hardware. Running such an image in a fully virtualized system like Qemu, as shown below, is only suitable for testing purposes. For production system this is not supported and suffers from a major performance penalty. To test the image with **qemu**, run the following commands:

```
cd /tmp/myoem-result
qemu-img create mydom0 10g
qemu -cdrom LimeJeOS-SLE12.x86_64-1.13.1.install.iso -hda mydom0 -boot d
```

When booted *mydom0* is a Xen dom0 from which other Xen guests can be started.

13.3. Building a Paravirtualized Xen Guest Image

The following example shows how to build a Just enough Operating System (JeOS) based on SUSE Linux Enterprise 12. The example again uses the `xenFlavour` profile but builds a simple vmx image. The result is a disk image with kernel-xen prepared for paravirtual boot via GRUB2. To boot such a guest a `pvGrub` or `pyGrub` machine configuration supporting GRUB2 must be provided.

```
kiwi --build suse-SLE12-JeOS -d /tmp/myvmx-result --type vmx \
     --add-profile xenFlavour
```

13.4. Building a Fully Virtualized Xen Guest

The following example shows how to build a Just enough Operating System (JeOS) based on SUSE Linux Enterprise 12. Contrary to the paravirtual guest image this example builds a simple vmx image including the standard kernel plus some kernel modules required by Xen. To boot such a guest, a hvmloader machine configuration must be provided.

```
kiwi --build suse-SLE12-JeOS -d /tmp/myvmx-result --type vmx \
     --add-profile xenFlavourHVM
```

13.5. Using the Guest Images

To run a domain U the Xen tool **xl** needs to be called in with a domain U configuration file:

```
xl create -f CONFIG-FILE
```

For paravirtual guest images KIWI supports the creation of the configuration file according to information provided with the `machine` element of the KIWI configuration file `config.xml`:

```
<image ...>
  <preferences ...<
    <machine memory="512" domain="domU">
      <vmdisk id="0" device="/dev/xvda" controller="ide"/>
      <vmnic interface=""/>
    </machine>
    ...
  </preferences>
  ...
</image>
```

If this information exists, KIWI creates a file with the extension `.xenconfig`. Note that not all possible configuration options are supported by the KIWI Xen configuration file creator. For fully virtualized images there is currently no support to create the configuration from KIWI. However tools like **virt-manager** support you setting up the machine configuration. Refer to the SUSE Linux Enterprise Virtualization Guide [https://www.suse.com/documentation/sles-12/book_virt/data/book_virt.html] or the Xen Documentation [http://www.xenproject.org/help/documentation.html] for more information.

14 Creating Appliances

Table of Contents

With the traditional model of application delivery, applications such as a word processor or an e-mail program are installed by a user or an administrator on individual machines. When deploying the application on multiple machines, this often requires to start the application installation on each machine. Furthermore, in case of machines solely dedicated to a single application or a defined set of applications, it is good practice, to adjust the operating system to optimize resource management and maximize security and performance. When multiple machines are affected, the steps for adjusting the operating system need to be performed on each machine, too.

An alternative to the traditional model of application delivery is to provide a so-called “appliance”. An appliance is the combination of the parts of a general purpose OS needed by a given application and the application itself, bundled and delivered as one unit. This unit may be delivered in a variety of formats, for example a ready to run virtual machine or a self-installing system on an optical media or a flash disk.

The appliance model has a many advantages. Operating system and application installation are no longer separate steps. The application is installed together with the operating system and the installation does not require manual intervention. The appliance provider can preconfigure the application to be ready-to-run directly after installation. Furthermore, the appliance provider can customize the operating system in terms of performance, resources and security, so it best fits the given appliance.

14.1. The KIWI Model

KIWI supports building appliances. When building appliances with KIWI the following procedure has proven to work reliably. Nevertheless it is a recommendation only and can be adapted to special needs and environments.

1. Choose an appropriate image description from the KIWI example templates. They are provided by the packages kiwi-templates, kiwi-desc-vmxboot, kiwi-desc-netboot, and kiwi-desc-oemboot. The templates are installed to `/usr/share/kiwi/images`. Add or adapt repositories and/or package names, according to the distribution you want to build an image for.

2. Allow the image to create an in-place git repository to allow tracking non-binary changes. This is done by adding the following line into the **config.sh** script:

   ```
   baseSetupPlainTextGITRepository
   ```

3. Set up a testing environment. A physical machine supporting to boot from a flash disk is recommended. All image templates provided by KIWI contain a hybrid iso type setup which is suitable for such a machine.

4. Build the live stick appliance by running

   ```
   kiwi --build template-name
   ```

5. Transfer the image generated in the previous step to the flash disk. Note that all data on the disk will be erased!

   ```
   dd if=PATH_TO_ISO_IMAGE
       of=FLASH_DISK_DEVICE bs=4M
   ```

6. Plug the flash disk into your test machine and boot it from the disk containing the appliance.

7. After your test system has successfully booted the image, log in to your appliance and start to tweak the system according to your needs. This includes all actions required to configure the appliance as needed. Before you start take care for the following:

   a. Create an initial package list. This can be done by calling:

      ```
      rpm -qa | sort > /tmp/deployPackages
      ```

   b. Check the output of the command **git** `status` and add all files matching the following criteria to `/.gitignore`:

      - the file is not yet part of the repository
      - you do not plan to change the file
      - the file will not be included by one of the image description overlay files

   When the initial package list exist and the git repository is set up, you can start to configure the system.

   ### Installing Additional Software

   Do not add additional software by installing a package being part of a repository not listed in `configuration.xml` or by compiling it from the sources. It is very hard to find out what binary files have been installed that way and it is also not architecture-safe.

   If there is really no other way for the software to become part of the image, you should address this issue directly in your image description and the **config.sh** script, but not after the initial deployment has happened.

8. As soon as the appliance on your test system works as expected it is ready to enter the final stage. At this point you have done several changes to the system which are tracked by the git repository and the package list. To include them into your image description, use the following process:

   a. Check the differences between the currently installed packages and the initial deployment list. This can be done by running:

      ```
      rpm -qa | sort > /tmp/appliancePackages
      ```

```
diff -u /tmp/deployPackages /tmp/appliancePackages
```

Add all packages labeled with `(+)` to the `<packages type="image">` section of your config.xml file and remove packages labeled with `(-)`.

In case you want to keep packages that have been automatically removed by the package manager, make sure you address these packages in the `config.sh` script. If you have installed packages from repositories which are not yet configured in `config.xml`, add them to allow KIWI to install the packages.

b. Check the differences made in the configuration files by running:

```
git diff >/tmp/appliancePatch
```

The patch created with this command (`/tmp/appliancePatch`) should become part of your image description. To make sure it is applied when preparing the image, add the following line to **config.sh**:

```
patch -p0 < appliancePatch
```

c. Check for new non binary files that have been added. This can be done by calling:

```
git status
```

All files not yet tracked, will be listed under `Untracked files`. Make sure to add all files from this list *which are not created automatically* to your image description. To do this, clone (copy) these files with regards to the file system structure as overlay files in your image description `root/` directory.

9. All customization work you did within your appliance is now stored in the image description. The image description you created can be re-used for all image types supported by KIWI.

To make sure the appliance works as expected prepare a new image tree and create an image from the new tree. You may optionally disable the creation of the git repository within this new image tree to save disk space. If this appliance is a server, you should keep it, because it allows you to keep track of changes during the live time of this appliance.

Cross Platform Appliance Building

Building appliances for a specific processor architecture on a different processor architecture is in generally *not* possible with KIWI. This limitation is based on the requirement that KIWI needs to be able to execute installed software inside the unpacked image tree. If the software installed inside the unpacked image tree does not run on the architecture of the build platform then KIWI cannot build the appliance. The only exception is building 32-bit x86 appliances on a 64-bit x86-64 architecture.

KIWI's option `--target-arch` is not intended to support cross-platform appliance builds. It rather tells the package manager to install packages for the specified architecture.

15 System Analysis/Migration

KIWI provides a module which allows you to analyze the running system for creating a report and an image description representing the current state of the machine. Among others, this allows you to clone your currently running system into an image. The process has the following limitations at the moment:

- Works for SUSE systems only (with zypper on board)
- The process works semi-automatically—depending on the complexity of the system, some manual postprocessing might be necessary

When calling KIWI's analysis module it tries to find the base version of your operating system by using the active repositories specified in the zypper database to identify the software packages currently installed. The result is a list of packages and patterns representing your system. Files not belonging to any packages, such as user data or configuration files, are provided as custom data by KIWI. In addition, KIWI offers different data visualization e.g unmanaged binary data. Along with the software analysis, KIWI also checks for enabled systemd services, augeas configuration inventory and more. The process will not go beyond the scope of local file systems.

To create an image of your running system, proceed as follows:

1. Create a report listing all packages and repositories currently installed by running

   ```
   sudo kiwi --describe workstation
   ```

   The result will be written to /var/cache/kiwi/describe/workstation

2. Check the result of the previous step for packages that cannot be assigned to a repository. If you do not need these packages within your image, proceed with the next step. If you want them to be in the image, either add the repositories containing these packages to your system (for example with **zypper addrepo**) and run the previous command again. Alternatively, tell KIWI which additional repos to check, by using the previous command together with the `--add-repo` and `--add-repotype` options (refer to **man 1 kiwi** for more information).

3. Rerun the report generating command. List all packages that are not part of a repository or should not be included with the `--skip` parameter. In case you have previously used the `--add-repo` and `--add-repotype` options, use them again with this command: *2015-12-22 - fs: --skip and --nofiles are not mentioned in "man 1 kiwi"*

   ```
   kiwi --describe workstation --nofiles \
   --skip "LIST_OF_PAKAGES" \
   --add-repo REPO --add-repotype TYPE
   ```

4. Next check the list of files not belonging to any packages in `/var/cache/kiwi/describe/workstation/custom.files`. Among others, this list contains user data, con-

figuration files, and database files. Make sure to only add files to the image that are really needed. In case you want to publish the image, double check the list for password files, configuration files containing plain text passwords, user data, databases and other sensitive data. All files that should become part of the image description need to be moved from `/var/cache/kiwi/describe/worksation/custom` to `/var/cache/kiwi/describe/worksation/root`. For additional information also check `/var/cache/kiwi/describe/workstation/custom.files.readme`.

5. Adjust the image description according to your needs by checking the following items:

   - Change author and contact in `config.xml`.
   - Set appropriate name for your image in `config.xml`.
   - Add or modify the image type (`oem` by default) set in `config.xml` if needed.
   - If you want to access any remote file system its a good idea to let AutoYaST add them on first boot of the system.
   - Check the network setup in `/etc/sysconfig/network`. Is this setup still possible in the cloned environment? Make sure you check for the MAC address of the card first.

6. Check the size of the image description. It's good practice to keep the image as small as possible. The size of a migrated image description mainly depends on how many overlay files exists in the `root/` directory. You should make sure to maintain only required overlay files.

7. Create an image from the description. By default an OEM image, containing a virtual disk that can also be deployed to a physical machine, is created. In addition to that, an ISO image containing an installable version of the image, is also generated. Refer to Chapter 12, *OEM Image / Preload Systems* for details.

   ```
   kiwi --build /var/cache/kiwi/describe/workstation -d /tmp/myResult
   ```

8. Test the image as described in Section 12.2, “Testing the Images”.

A KIWI Man Pages

Table of Contents

The following pages will show you the man page of KIWI and the functions which can be used within **config.sh** and **index.sh**

kiwi

kiwi — Creating Operating System Images

Synopsis

`kiwi` { -l | --list }

`kiwi` { -o | --clone } *`image-path`* { -d } *`destination`*

`kiwi` { -b | --build } *`image-path`* { -d } *`destination`*

Basics

KIWI is a complete imaging solution that is based on an image description. Such a description is represented by a directory which includes at least one `config.xml` file and may as well include other files like scripts or configuration data. The `kiwi-templates` package provides example descriptions based on a JeOS system. JeOS means *Just enough Operating System*. KIWI provides image templates based on that axiom which means a JeOS is a small, text only based image including a predefined remote source setup to allow installation of missing software components at a later point in time.

Detailed description of the kiwi image system exists in the system design document in file:///usr/share/doc/packages/kiwi/kiwi.pdf. KIWI always operates in two steps. The KIWI `--build` option just combines both steps into one to make it easier to start with KIWI. The first step is the preparation step and if that step was successful, a creation step follows which can create different image output types. If you have started with an example and want to add you own changes it might be a good idea to clone of from this example. This can be done by simply copying the entire image description or you can let KIWI do that for you by using the **kiwi** `--clone` command.

In the preparation step, you prepare a directory including the contents of your new file system based on one or more software package source(s) The creation step is based on the result of the preparation step and uses the contents of the new image root tree to create the output image. If the image type ISO was requested, the output image would be a file with the suffix `.iso` representing a live system on CD or DVD. Other than that KIWI can create images for virtual and para-virtual (Xen) environments as well as for USB stick, PXE network clients and OEM customized Linux systems.

General Options

`[-h | --help]`
Display help.

`[--version]`
Display the KIWI version.

`[--check-config` *`path-to-the-configuration-file`*`]`
Checks the XML configuration file.

`[--nocolor]`
Do not use colored output.

Image Preparation and Creation

`kiwi` { -p | --prepare } *`image-path`*
{ -r | --root } *`image-root`* [--cache *`directory`*]

`kiwi` { -c | --create } *`image-root`*
{ -d | --destdir } *`destination`* [--type *`image-type`*]

Image Upgrade

If the image root tree is stored and not removed, it can be used for upgrading the image according to the changes made in the repositories used for this image. If a distributor provides an update channel for package updates and an image `config.xml` includes this update channel as repository, it is useful to store the image root tree and upgrade the tree according to changes on the update channel. Given that the root tree exists it's also possible to add or remove software and recreate the image of the desired type.

`kiwi` { -u | --upgrade } *`image-root`* [--add-package *`name`*] [--add-pattern *`name`*]

System Analysis

KIWI provides a module which allows you to analyze the running system and create a report and an image description representing the current state of the machine. Among others this allows you to clone your currently running system into an image. The system requires the zypper back-end in order to work properly.

The process will always place it's result into the `/tmp/$OptionValueOf--describe` directory. The reason for this is because `/tmp` is always excluded from the analysis and therefore we can safely place new files there without influencing the process itself. You should have at least 100 MB free space for the cache file and the image description all the rest are just hard links.

As one result a HTML based report file is created which contains important information about the system. You are free to ignore that information but with the risk that the image from that description does not represent the same system which is running at the moment. The less issues left in the report the better is the result. In most cases a manual fine tuning is required. This includes the repository selection and the unmanaged files along with the configuration details of your currently running operating system. You should understand the module as a helper to analyze running linux systems.

`kiwi` { --describe } *`name`*

Image Postprocessing Modes

The KIWI post-processing modes are used for special image deployment tasks, like installing the image on a USB stick. So to say they are the third step after preparation and creation. KIWI calls the postprocessing modules automatically according to the specified output image type and attributes but it's also possible to call them manually.

`kiwi` --bootvm *`initrd`* --bootvm-system *`systemImage`* [--bootvm-disksize *`size`*]

`kiwi` --bootcd *`initrd`*

`kiwi` --bootusb *`initrd`*

`kiwi` --installcd *`initrd`* --installcd-system *`raw-system-image`*

`kiwi` --installstick *`initrd`* --installstick-system *`raw-system-image`*

`kiwi` --installpxe *`initrd`* --installpxe-system *`raw-system-image`*

Image Format Conversion

The KIWI format conversion is useful to perform the creation of another image output format like vmdk for VMware or ovf the open virtual machine format. Along with the conversion KIWI also creates the virtual machine configuration according to the format if there is a machine section specified in the XML description

`kiwi` --convert *`systemImage`* [--format *`vmdk|ovf|qcow2|vhd`*]

Helper Tools

The helper tools provide optional functions like creating an encrypted password string for the users section of the `config.xml` file as well as signing the image description with an md5sum hash and adding splash data to the boot image used by the boot loader.

`kiwi` --createpassword

`kiwi` --createhash *`image-path`*

`kiwi` { -i | --info } *`ImagePath`* {--select *`repo-patterns|patterns|types|sources|size|profiles|packages|version`* }

`kiwi` --setup-splash *`initrd`*

The following list describes the helper tools more detailed

`[--createpassword]`
: Create a crypted password hash and prints it on the console. The user can use the string as value for the pwd attribute in the XML users section

`[--createhash` *`image-path`*`]`
: Sign your image description with a md5sum. The result is written to a file named `.checksum.md` and is checked if KIWI creates an image from this description.

`[-i | --info` *`image-path`*`--select` *`selection`*`]`
: List general information about the image description. So far you can get information about the available patterns in the configured repositories with *`repo-patterns`*, a list of used patterns for this image with *`patterns`*, a list of supported image types with *`types`*, a list of source URLs with *`sources`*, an estimation about the install size and the size of the packages marked as to be deleted with *`size`*, a list of profiles with *`profiles`*, a list of solved packages to become installed with *`packages`*, and the information about the appliance name and version with *`version`*

`[--setup-splash` *`initrd`*`]`
: Create splash screen from the data inside the initrd and re-create the initrd with the splash screen attached to the initrd cpio archive. This enables the kernel to load the splash screen at boot time. If splashy is used only a link to the original initrd will be created

Global Options

[`--add-profile`*`profile-name`*]
: Use the specified profile. A profile is a part of the XML image description and therefore can enhance each section with additional information. For example adding packages.

[`--set-repo`*`URL`*]
: Set/Overwrite the repo URL for the first repo listed in the configuration file that does not have a "fixed" status. The change is temporary and will not be written to the XML file.

[`--set-repotype`*`type`*]
: Set/Overwrite repo type for the first listed repo. The supported repo types depends on the package manager. Commonly supported are rpm-md, rpm-dir and yast2. The change is temporary and will not be written to the XML file.

[`--set-repoalias`*`name`*]
: Set/Overwrite alias name for the first listed repo. Alias names are optional free form text. If not set the source attribute value is used and builds the alias name by replacing each “/” with a “_”. An alias name should be set if the source argument doesn't really explain what this repository contains. The change is temporary and will not be written to the XML file.

[`--set-repoprio`*`number`*]
: Set/Overwrite priority for the first listed repo. Works with the smart package manager only. The Channel priority assigned to all packages available in this channel (0 if not set). If the exact same package is available in more than one channel, the highest priority is used.

[`--add-repo` *`URL`*, `--add-repotype` *`type`*`--add-repoalias` *`name`*`--add-repoprio` *`number`*]
: Add the given repository and type for this run of an image prepare or upgrade process. Multiple `--add-repo`/`--add-repotype` options are possible. The change will not be written to the `config.xml` file

[`--ignore-repos`]
: Ignore all repositories specified so far, in XML or elsewhere. This option should be used in conjunction with subsequent calls to `--add-repo` to specify repositories at the command line that override previous specifications.

[`--logfile` *`Filename`* | `terminal`]
: Write to the log file *`Filename`* instead of the terminal.

[`--gzip-cmd` *`cmd`*]
: Specify an alternate command to run when compressing boot and system images. Command must accept **gzip** options.

[`--package-manager` *`smart`*|*`zypper`*]
: Set the package manager to use for this image. If set it will temporarily overwrite the value set in the xml description.

[`-A` | `--target-arch` *`i586`*|*`x86_64`*|*`armv5tel`*|*`ppc`*]
: Set a special target-architecture. This overrides the used architecture for the image-packages in `zypp.conf`. When used with smart this option doesn't have any effect.

[`--disk-start-sector` *`number`*]
: The start sector value for virtual disk based images. The default is 2048. For newer disks including SSD this is a reasonable default. In order to use the old style disk layout the value can be set to 32.

`[--disk-sector-size` *`number`*`]`
: Overwrite the default 512 byte sector size value. This will influence the partition alignment.

`[--disk-alignment` *`number`*`]`
: Align the start of each partition to the specified value. By default 4096 bytes are used.

`[--debug]`
: Prints a stack trace in case of internal errors

`[--verbose` *`1|2|3`*`]`
: Controls the verbosity level for the instsource module

`[-y | --yes]`
: Answer any interactive questions with yes

`[--create-instsource` *`path-to-config.xml`*`]`
: Using this option, it is possible to create a valid installation repository from blank RPM file trees. The created tree can be used directly for the image creation process afterwards.

`[--bundle-build]`
: This option bundles the build results to be suitable for publishing it in the Build Service. It allows adding a build-number in combination with the `--bundle-id` option as well as a SHA key to the results. It also removes intermediate build results not relevant for users if they don't want to rebuild the image.

`[--bundle-id` *`build-number`*`]`
: The build-number/string in combination with `--bundle-build`

Image Preparation Options

`[-r | --root` *`RootPath`*`]`
: Set up the physical extend, chroot system below the given root-path path. If no `--root` option is given, KIWI will search for the attribute defaultroot in `config.xml`. If no root directory is known, a **mktemp** directory will be created and used as root directory.

`[--force-new-root]`
: Force creation of new root directory. If the directory already exists, it is deleted.

Image Upgrade/Preparation Options

`[--cache`*`directory`*`]`
: When specifying a cache directory, KIWI will create a cache each for patterns and packages and re-use them, if possible, for subsequent root tree preparations of this and/or other images

`[--init-cache`*`image description`*`]`
: Creates a cache from a KIWI image description.

`[--recycle-root]`
: Uses an existing root tree and base the kiwi prepare step on top of it. This is used to speed things up.

[`--force-bootstrap`]
: In combination with recycle-root this option forces to call the bootstrap phase of kiwi, which is not considered necessary under normal circumstances.

[`--add-package`*`package`*]
: Add the given package name to the list of image packages multiple `--add-package` options are possible. The change will not be written to the XML description.

[`--add-pattern`*`name`*]
: Add the given pattern name to the list of image packages multiple `--add-pattern` options are possible. The change will not be written to the xml description. Patterns can be handled by SUSE based repositories only.

[`--del-package`*`package`*]
: Removes the given package by adding it the list of packages to become removed. The change will not be written to the xml description.

Image Creation Options

[`-d` | `--destdir` *`DestinationPath`*]
: Specify destination directory to store the image file(s) If not specified, KIWI will try to find the attribute *`defaultdestination`* which can be specified in the *`preferences`* section of the `config.xml` file. If it exists its value is used as destination directory. If no destination information can be found, an error occurs.

[`-t` | `--type` *`Imagetype`*]
: Specify the output image type to use for this image. Each type is described in a *`type`* section of the preferences section. At least one type needs to be specified in the `config.xml` description. By default, the types specifying the *primary* attribute will be used. If there is no primary attribute set, the first type section of the preferences section is the primary type. The types are only evaluated when KIWI runs the `--create` step. With the option `--type` one can distinguish between the types stored in `config.xml`

[`-s` | `--strip`]
: Strip shared objects and executables - only makes sense in combination with `--create`

[`--prebuiltbootimage` *`Directory`*]
: Search in *`Directory`* for pre-built boot images.

[`--isocheck`]
: in case of an iso image the checkmedia program generates a md5sum into the ISO header. If the `--isocheck` option is specified a new boot menu entry will be generated which allows to check this media

[`--lvm`]
: Use the logical volume manager to control the disk. The partition table will include one lvm partition and one standard ext2 boot partition. Use of this option makes sense for the create step only and also only for the image types: vmx, oem, and usb

[`--fs-blocksize` *`number`*]
: When calling KIWI in creation mode this option will set the block size in bytes. For ISO images with the old style ramdisk setup a block size of 4096 bytes is required

[`--fs-journalsize` *`number`*]
: When calling KIWI in creation mode this option will set the journal size in mega bytes for ext[23] based file systems and in blocks if the Reiser file system is used

[`--fs-inodesize` *number*]
: When calling KIWI in creation mode this option will set the inode size in bytes. This option has no effect if the Reiser file system is used

[`--fs-inoderatio` *number*]
: Set the bytes/inode ratio. This option has no effect if the Reiser file system is used

[`--fs-max-mount-count` *number*]
: When calling kiwi in creation mode this option will set the number of mounts after which the file system will be checked. Set to 0 to disable checks. This option applies only to ext[234] file systems.

[`--fs-check-interval` *number*]
: When calling kiwi in creation mode this option will set the maximal time between two file system checks. Set to 0 to disable time-dependent checks. This option applies only to ext[234] file systems.

[`--fat-storage` *size in MB*]
: if the syslinux boatload is used this option allows to specify the size of the fat partition. This is useful if the fat space is not only used for booting the system but also for custom data. Therefore this option makes sense when building a USB stick image (image type: usb or oem)

[`--partitioner` *parted|fdasd*]
: Select the tool to create partition tables. Supported are parted and fdasd (s390). By default parted is used

[`--check-kernel`]
: Activates check for matching kernels between boot and system image. The kernel check also tries to fix the boot image if no matching kernel was found.

[`--mbrid` *number*]
: Specifies a custom mbrid. The number value is treated as decimal number which is internally translated into a 4byte hex value. The allowed range therefore is from 0x0 to max 0xffffffff. By default kiwi creates a random value

[`--edit-bootconfig` *script*]
: Specifies the location of a custom script which is called right before the boot loader is installed. This allows to modify the boot loader configuration file written by kiwi. The scripts working directory is the one which represents the image structure including the boot loader configuration files. Please have in mind that according to the image type, architecture and boot loader type the files/directory structure and also the name of the boot loader configuration files might be different.

[`--edit-bootinstall` *script*]
: Specifies the location of a custom script which is called right after the boot loader is installed.

[`--archive-image`]
: When calling kiwi `--create` this option allows to pack the build result(s) into a tar archive.

[`--targetdevice`*device*]
: Use an alternative block device instead of the loop device. The given location must be a block device node, not a symlink or other linux device node type.

For More Information

More information about KIWI, its files can be found at:

https://opensuse.github.io/kiwi/
: KIWI wiki

`config.xml`
: The configuration XML file that contains every aspect for the image creation.

file:///usr/share/doc/packages/kiwi/kiwi.pdf
: The system documentation which describes the supported image types in detail.

file:///usr/share/doc/packages/kiwi/schema/kiwi.xsd.html
: The KIWI RELAX NG XML Schema documentation.

kiwi::config.sh

KIWI::config.sh — Customization File for KIWI image description

Description

The KIWI image description allows to have an optional `config.sh` bash script in place. It can be used for changes appropriate for all images to be created from a given unpacked image (since config.sh runs prior to create step) Basically the script should be designed to take over control of adding the image operating system configuration. Configuration in that sense means all tasks which runs once in an os installation process like activating services, creating configuration files, prepare an environment for a firstboot workflow, etc. The `config.sh` script is called *after* the following kiwi built in configuration tasks: User/Groups, copy of overlay root tree and setup of AutoYaST If `config.sh` exits with an exit code != 0 the kiwi process will exit with an error too.

Example A.1. Template for config.sh

```
#======================================
# Functions...
#--------------------------------------
test -f /.kconfig && . /.kconfig
test -f /.profile && . /.profile

#======================================
# Greeting...
#--------------------------------------
echo "Configure image: [$kiwi_iname]..."

#======================================
# Mount system filesystems
#--------------------------------------
baseMount

#======================================
# Call configuration code/functions
#--------------------------------------
...

#======================================
# Umount kernel filesystems
#--------------------------------------
baseCleanMount

#======================================
# Exit safely
#--------------------------------------
exit 0
```

Common functions

The `.kconfig` file allows to make use of a common set of functions. Functions specific to SUSE Linux specific begin with the name *suse*. Functions applicable to all linux systems starts with the name *base*. The following list describes the functions available inside the `config.sh` script.

[baseCleanMount]
    Umount the system filesystems `/proc`, `/dev/pts`, and `/sys`.

[baseDisableCtrlAltDel]
Disable the **Ctrl-Alt-Del** key sequence setting in `/etc/inittab`

[baseGetPackagesForDeletion]
Return the name(s) of packages which will be deleted

[baseGetProfilesUsed]
Return the name(s) of profiles used to build this image

[baseSetRunlevel {value}]
Set the default run level

[baseSetupBoot]
Set up the linuxrc as init

[baseSetupBusyBox {-f}]
activates busybox if installed for all links from the `busybox/busybox.links` file—you can choose custom apps to be forced into busybox with the `-f` option as first parameter, for example:

```
baseSetupBusyBox -f /bin/zcat /bin/vi
```

[baseSetupInPlaceGITRepository]
Create an in place git repository of the root directory. This process may take some time and you may expect problems with binary data handling

[baseSetupInPlaceSVNRepository {path_list}]
Create an in place subversion repository for the specified directories. A standard call could look like this baseSetupInPlaceSVNRepository `/etc`, `/srv`, and `/var/log`

[baseSetupPlainTextGITRepository]
Create an in place git repository of the root directory containing all plain/text files.

[baseSetupUserPermissions]
Search all home directories of all users listed in `/etc/passwd` and change the ownership of all files to belong to the correct user and group.

[baseStripAndKeep {list of info-files to keep}]
helper function for strip* functions read stdin lines of files to check for removing params: files which should be keep

[baseStripDocs {list of docu names to keep}]
remove all documentation, except one given as parameter

[baseStripInfos {list of info-files to keep}]
remove all info files, except one given as parameter

[baseStripLocales {list of locales}]
remove all locales, except one given as parameter

[baseStripMans {list of manpages to keep}]
remove all manual pages, except one given as parameter example: baseStripMans more less

[baseStripRPM]
remove rpms defined in `config.xml` in the image type=delete section

[suseRemovePackagesMarkedForDeletion]
: remove rpms defined in `config.xml` in the image type=delete section. The difference compared to baseStripRPM is that the suse variant checks if the package is really installed prior to passing it to rpm to uninstall it. The suse rpm exits with an error exit code while there are other rpm version which just ignore if an uninstall request was set on a package which is not installed

[baseStripTools {list of toolpath} {list of tools}]
: helper function for suseStripInitrd function params: toolpath, tools

[baseStripUnusedLibs]
: remove libraries which are not directly linked against applications in the bin directories

[baseUpdateSysConfig {filename} {variable} {value}]
: update sysconfig variable contents

[Debug {message}]
: Helper function to print a message if the variable `DEBUG` is set to 1

[Echo {echo commandline}]
: Helper function to print a message to the controlling terminal

[Rm {list of files}]
: Helper function to delete files and announce it to log

[Rpm {rpm commandline}]
: Helper function to the RPM function and announce it to log

[suseConfig]
: Setup keytable language, timezone and hwclock if specified in `config.xml` and call SuSEconfig afterwards SuSEconfig is only called on systems which still support it

[suseInsertService {servicename}]
: This function calls baseInsertService and exists only for compatibility reasons

[suseRemoveService {servicename}]
: This function calls baseRemoveService and exists only for compatibility reasons

[baseInsertService {servicename}]
: Activate the given service by using the chkconfig or systemctl program. Which init system is in use is auto detected

[baseRemoveService {servicename}]
: Deactivate the given service by using the chkconfig or systemctl program. Which init system is in use is auto detected

[baseService {servicename} {on|off}]
: Activate/Deactivate a service by using the chkconfig or systemctl program. The function requires the service name and the value on or off as parameters. Which init system is in use is auto detected

[suseActivateDefaultServices]
: Activates the following sysVInit services to be on by default using the chkconfig program: boot.rootfsck, boot.cleanup, boot.localfs, boot.localnet, boot.clock, policykitd, dbus, consolekit, haldaemon, network, atd, syslog, cron, kbd. And the following for systemd systems: network, cron

[suseSetupProduct]
This function creates the baseproduct link in /etc/products.d pointing to the installed product

[suseSetupProductInformation]
This function will use zypper to search for the installed product and install all product specific packages. This function only makes sense if zypper is used as package manager

[suseStripPackager {-a}]
Remove smart or zypper packages and db files Also remove rpm package and db if `-a` given

Profile environment variables

The .profile environment file contains a specific set of variables which are listed below. Some of the functions above use the variables.

[$kiwi_compressed]
The value of the compressed attribute set in the type element in `config.xml`

[$kiwi_delete]
A list of all packages which are part of the packages section with `type="delete"` in `config.xml`

[$kiwi_drivers]
A comma separated list of the driver entries as listed in the drivers section of the `config.xml`.

[$kiwi_iname]
The name of the image as listed in `config.xml`

[$kiwi_iversion]
The image version string major.minor.release

[$kiwi_keytable]
The contents of the keytable setup as done in `config.xml`

[$kiwi_language]
The contents of the locale setup as done in `config.xml`

[$kiwi_profiles]
A list of profiles used to build this image

[$kiwi_size]
The predefined size value for this image. This is not the computed size but only the optional size value of the preferences section in `config.xml`

[$kiwi_timezone]
The contents of the timezone setup as done in `config.xml`

[$kiwi_type]
The basic image type. Can be a simply file system image type of ext2, ext3, reiserfs, squashfs, cpio, or one of the following complex image types: iso, split, usb, vmx, oem, xen, or pxe.

kiwi::images.sh

KIWI::images.sh — Customization File for KIWI image description

Description

The KIWI image description allows to have an optional `images.sh` bash script in place. It can be used for changes appropriate for certain images/image types on case-by-case basis (since it runs at beginning of create step) Basically the script should be designed to take over control of handling image type specific tasks. For example if building the oem type requires some additional package or config it can be handled in images.sh. Please keep in mind there is only one unpacked root tree the script operates in. This means all changes are permanent and will not be automatically restored. It is also the script authors tasks to check if changes done before do not interfere in a negative way if another image type is created from the same unpacked image root tree If `images.sh` exits with an exit code != 0 the kiwi process will exit with an error too.

Example A.2. Template for images.sh

```
#======================================
# Functions...
#--------------------------------------
test -f /.kconfig && . /.kconfig
test -f /.profile && . /.profile

#======================================
# Greeting...
#--------------------------------------
echo "Configure image: [$kiwi_iname]..."

#======================================
# Call configuration code/functions
#--------------------------------------
...

#======================================
# Exit safely
#--------------------------------------
exit
```

Common functions

The `.kconfig` file allows to make use of a common set of functions. Functions specific to SUSE Linux specific begin with the name *suse*. Functions applicable to all linux systems starts with the name *base*. The following list describes the functions available inside the `images.sh` script.

[baseCleanMount]
: Umount the system file systems `/proc`, `/dev/pts`, and `/sys`.

[baseGetProfilesUsed]
: Return the name(s) of profiles used to build this image.

[baseGetPackagesForDeletion]
: Return the list of packages setup in the packages `type="delete"` section of the `config.xml` used to build this image.

[suseGFXBoot {theme} {loadertype}]
: This function requires the gfxboot and at least one bootsplash-theme-* package to be installed to work correctly. The function creates from this package data a graphics boot screen for the isolinux and grub boot loaders. Additionally it creates the bootsplash files for the resolutions 800x600, 1024x768, and 1280x1024

[suseStripKernel]
: This function removes all kernel drivers which are not listed in the *drivers sections of the `config.xml` file.

[suseStripInitrd]
: This function removes a whole bunch of tools binaries and libraries which are not required to boot a suse system with KIWI.

[Rm {list of files}]
: Helper function to delete files and announce it to log.

[Rpm {rpm commandline}]
: Helper function to the rpm function and announce it to log.

[Echo {echo commandline}]
: Helper function to print a message to the controlling terminal.

[Debug {message}]
: Helper function to print a message if the variable `DEBUG` is set to 1.

Profile environment variables

The .profile environment file contains a specific set of variables which are listed below. Some of the functions above use the variables.

[$kiwi_iname]
: The name of the image as listed in config.xml

[$kiwi_iversion]
: The image version string major.minor.release

[$kiwi_keytable]
: The contents of the keytable setup as done in `config.xml`

[$kiwi_language]
: The contents of the locale setup as done in `config.xml`

[$kiwi_timezone]
: The contents of the timezone setup as done in `config.xml`

[$kiwi_delete]
: A list of all packages which are part of the packages section with `type="delete"` in `config.xml`

[$kiwi_profiles]
: A list of profiles used to build this image

[$kiwi_drivers]
: A comma separated list of the driver entries as listed in the drivers section of the `config.xml`.

[$kiwi_size]
: The predefined size value for this image. This is not the computed size but only the optional size value of the preferences section in `config.xml`

[$kiwi_compressed]
: The value of the compressed attribute set in the type element in config.xml

[$kiwi_type]
: The basic image type. Can be a simply file system image type of ext2, ext3, reiserfs, squashfs, and cpio or one of the following complex image types: iso split usb vmx oem xen pxe

kiwi::kiwirc

KIWI::kiwirc — Resource file for the Kiwi imaging system

Description

The KIWI imaging tool chain supports the use of an optional resource file named `.kiwirc` located in the users home directory.

The file is sourced by a Perl process and thus Perl compatible syntax for the supported variable settings is required.

Example A.3. Template for .kiwi.rc

```
$BasePath='/usr/share/kiwi';
$Gzip='bzip2';
$LogServerPort='4455';
$System='/usr/share/kiwi/image';
```

Supported Resource Settings

KIWI recognizes the BasePath, Gzip, LogServerPort, LuksCipher, and System settings in the `.kiwirc` file.

[BasePath]
: Path to the location of the KIWI image system components, such as modules, tests, image descriptions etc.

  The default value is `/usr/share/kiwi`

[Gzip]
: Specify the compression utility to be used for various compression tasks during image generation.

  The default value is **gzip** `-9`

[LogServerPort]
: Specify a port number for log message queuing.

  The default value is off

[LuksCipher]
: Specify the cipher for the encrypted Luks file system.

[System]
: Specify the location of the KIWI system image description.

  The default value is the value of BasePath concatenated with /image.

B Setting Up a Network Boot Server

To be able to deploy PXE bot images created with KIWI, you need to set up a network boot server providing the services DHCP and atftp.

Procedure B.1. Installing and Configuring atftp

1. Install the packages atftp and kiwi-pxeboot.

2. Edit the file `/etc/sysconfig/atftpd`. Set or modify the following variables:

   ```
   ATFTPD_OPTIONS="--daemon --no-multicast"
   ATFTPD_DIRECTORY="/srv/tftpboot"
   ```

3. Start the `atftpd` service by running:

   ```
   systemctl start atftpd
   ```

Procedure B.2. Installing and Configuring DHCP

Contrary to the atftp server setup the following instructions can only serve as an example. Depending on your network structure, the IP addresses, ranges and domain settings need to be adapted to allow the DHCP server to work within your network. If you already have a DHCP server running in your network, make sure that the `filename` and `next-server` are correctly set in `/etc/dhcpd.conf` on this server.

The following steps describe how to set up a new DHCP server instance:

1. Install the package dhcp-server.

2. Create the file `/etc/dhcpd.conf` and include the following statements. Note that all *values* listed below are examples, make sure to replace them with data fitting your network setup.

   ```
   option domain-name "example.org";
   option domain-name-servers 192.168.100.2;
   option broadcast-address 192.168.100.255;
   option routers 192.168.100.2;
   option subnet-mask 255.255.255.0;
   default-lease-time 600;
   max-lease-time 7200;
   ddns-update-style none; ddns-updates off;
   log-facility local7;

   subnet 192.168.100.0 netmask 255.255.255.0 {
      filename "pxelinux.0";
      next-server 192.168.100.2;
      range dynamic-bootp 192.168.100.5 192.168.100.20;
   }
   ```

3. Edit the file `/etc/sysconfig/dhcpd` and setup the network interface the server should listen on:

   ```
   DHCPD_INTERFACE="eth0"
   ```

4. Run the `dhcp` server by calling:

   ```
   systemctl start wickedd-dhcp4
   ```

C GNU Licenses

Table of Contents

This appendix contains the GNU Free Documentation License version 1.2.

C.1. GNU Free Documentation License

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggre-

8. TRANSLATION

9. TERMINATION

10. FUTURE REVISIONS OF THIS LICENSE

ADDENDUM: How to use this License for your documents

```
Copyright (c) YEAR YOUR NAME.
Permission is granted to copy, distribute and/or modify
this document under the terms of the GNU Free Documentation
License, Version 1.2 or any later version published by the
```

```
Free Software Foundation; with no Invariant Sections,
no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled
“GNU Free Documentation License”.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

```
with the Invariant Sections being LIST THEIR TITLES,
with the Front-Cover Texts being LIST, and with
the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index