openSUSE-KIWI Image System

Cookbook

Marcus Schäfer

openSUSE-KIWI Image System: Cookbook

by Marcus Schäfer
Thomas Schraitle <toms@suse.de>
Robert Schweikert <rjschwei@suse.com>

KIWI Version 7.02

License

Table of Contents

Part I. Concepts and Basics

Table of Contents

1 Introduction

Table of Contents

1.1. What is KIWI?

KIWI is an image build system for Linux.

A Linux image may present itself in many different formats, for example the `*.iso` file you download to burn a distribution installation file to optical media 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, you can think of an image as a Linux system in a file. Depending on the type of the image you are dealing with you have different options for using the image. For example you can burn an ISO image to optical media and then boot your computer from the CD/DVD, or you can run a Virtual Machine from the `*.iso` file (image) stored on your hard drive.

1.2. What does KIWI do?

KIWI builds images in a variety of formats.

As an image build tool, KIWI builds images in a relatively large number of supported image formats. The details of the image creation process are explained in the Chapter 3, *Basic Workflow* chapter. The image format of the image produced by KIWI is defined within a configuration file named `config.xml` or `*.kiwi` as described in Chapter 5, *KIWI Image Description*.

Note that not all elements and attributes that may be used in the KIWI `config.xml` configuration file are listed or described in this document. The complete schema documentation can be accessed on the web at http://doc.opensuse.org/projects/kiwi/schema-doc/, latest version, or on your local system using the `file:///usr/share/doc/packages/kiwi/schema/kiwi.html` path as the URL in the browser.

1.3. How do I 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.

An image is created in a two step process as described in the Chapter 3, *Basic Workflow* chapter. Use **kiwi --prepare** for the first step and **kiwi --create** for the second step. For user convenience KIWI also has the `--build` that combines the *prepare* and *create* steps.

2 Installation

Table of Contents

2.1. Installing using Packages

Once you have added the appropriate repositories (more on this below) to your system you can search for the kiwi packages through the YaST interface or using **zypper** as shown below.

```
zypper se kiwi
```

The list of packages returned by zypper contains the main package, simply named `kiwi-`, the `-doc` package containing the documentation files, and the `-desc-` packages containing the boot descriptions for the various image types. Installing this set of packages is sufficient to build your images.

Adding repositories to your system can be accomplished using the YaST interface or the **zypper ar** command.

2.1.1. Package Repositories

The simplest and most straight forward way to install KIWI is to use the packages that are provided in the Virtualization:Appliances project on the web here: `http://download.opensuse.org/repositories/Virtualization:/Appliances/`

2.2. Running from Source

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

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

Before running from source you want to verify that all the dependencies are satisfied. The best way to accomplish this is to install all packages listed as *BuildRequires* in the `.spec` file found in the `rpm` directory. Once all dependent packages are installed change your working directory to the `kiwi` directory and run kiwi as follows:

```
./kiwi
```

The KIWI self tests are executed using:

```
make test
```

If you want to refresh your source with the latest checked in code you can simply pull the latest sources from the GitHub repository using the command shown below.

```
git pull
```

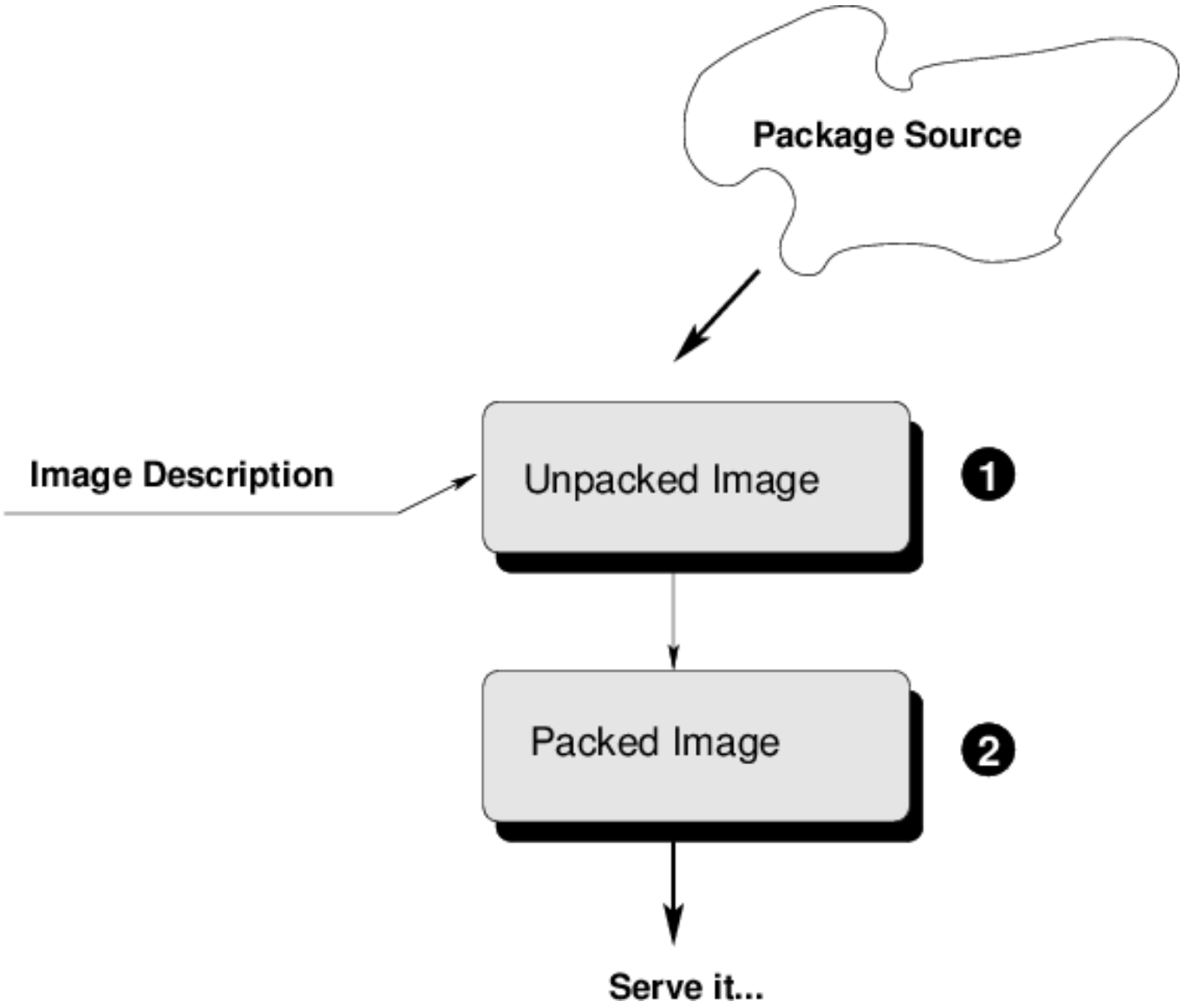
3 Basic Workflow

Table of Contents

3.1. Introduction

KIWI creates images in a two step process, as mentioned previously. 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` stored inside of the unpacked image below the `image/` directory.

Figure 3.1. Image Creation Architecture



❶ Encapsulated system as directory/files tree
❷ Encapsulated system as image file

Prior to building an image with KIWI it is important to understand the composition of an image, the general concepts of Linux, including the boot process, and distribution concepts such as package management.

Installation of a Linux system generally occurs by booting a target system from an installation source such as an install CD/DVD, a live CD/DVD, or entering the PXE boot environment. The installation process is often driven by an installer that interacts with the user to collect 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 necessary and specified software onto the target system using packages from the available software sources (repositories). After the installation is complete the system generally reboots and enters a configuration procedure upon startup. The configuration may be fully automatic or it may include user interaction.

A system image, or image, is a *complete installation* of a Linux system in a file. The image represents an operational system and may or may not contain the "final" configuration. The behavior of the image upon deployment varies depending on image type and image configuration. With KIWI it is possible to completely customize the initial start up behavior of the image. This may include behavior that allows the image to simply be deployed inside an existing virtual environment with no required configuration at start up. It is also possible to create images that automatically configure themselves in a known target environment. Further, the startup of an interactive configuration procedure can be integrated into the image to allow the user to configure the image when it is booted for the first time. The image configuration possibilities are practically unlimited. The image creation process with KIWI is automated and does not require any user interaction. The required information for the image creation process is provided in the primary configuration file named `config.xml`. The image can optionally be customized using the `config.sh` and `images.sh` scripts. Additional customization can be accomplished with the use of an optional *overlay tree (directory)* called `root`. The configuration information is stored in the so called *image description* or *configuration directory (tree)*.

3.2. Build Process

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 a new root tree or so called *unpacked image*. 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 target for any 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 *smart*, *zypper*, *yum* and *apt* package managers. The prepare step executes the following major stages:

- **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.

- **Install Packages.**
  Initially KIWI configures the package manager (zypper by default) to be used for the image creation to use the repositories specified in the configuration file and/or specified on the command line. Following the repository setup the packages specified in the `bootstrap` section 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 packages that you do not want in your image. Removing undesired packages can be accomplished by specifying the packages you would like to remove from the image as children of a `packages` element where the value of the `type` attribute of the `packages` element is set to `delete`.

- **Apply The Overlay Tree.**
  After the package installation with the package manager is complete, KIWI will apply all files and directories present in the overlay directory named *root* inside the configuration

directory to the target root tree. This allows you to over write any file that was installed by one of the packages installed during the installation phase. Files and directories will appear in the unpacked image tree in the same location as they are found in the directory named *root*.

- **Apply Archives.**
  Any archives specified with the `archive` element in the `config.xml` file are applied in the specified order (top to bottom) after the overlay tree copy operation is complete. Archives are unpacked at the top level of the new root tree and files will be located according to their path in the archive. As with the overlay tree, it is possible to over write any file in the target root tree.

- **Execute User Defined config.sh Script.**
  At the end of the preparation stage the optional script named `config.sh` is executed at the root level of the target root tree. The primary intended use of this script is to complete system configuration such as service activation. For detailed description pre-defined configuration functions consult the kiwi::config.sh(1) man page.

- **Manage The New Root Tree.**
  The unpacked image directory is just a directory, as far as the build system is concerned and you can manipulate the content of this directory to your liking. Further, as this directory represents a system installation you can chroot into this directory and run in the chroot environment to make changes. However, it is strongly discouraged to apply changes directly to the unpacked root, as any changes you apply will be lost when the *prepare* step for the image is repeated. In addition you may introduce errors into the unpacked root tree that may lead to very difficult to track kiwi build issues during the *create* step of the image build process. The best practice is to apply any necessary changes to the configuration directory followed by a new prepare operation. If you inspect the created unpacked root tree you will find a directory named `image` at the top level that you would not find on a system installed with the distribution installer. This directory contains information KIWI requires during the create step, including a copy of the `config.xml` file. You can make modifications to data in this directory to influence the create step, however, as mentioned previously this is discouraged and changes will be lost once the prepare step is repeated.

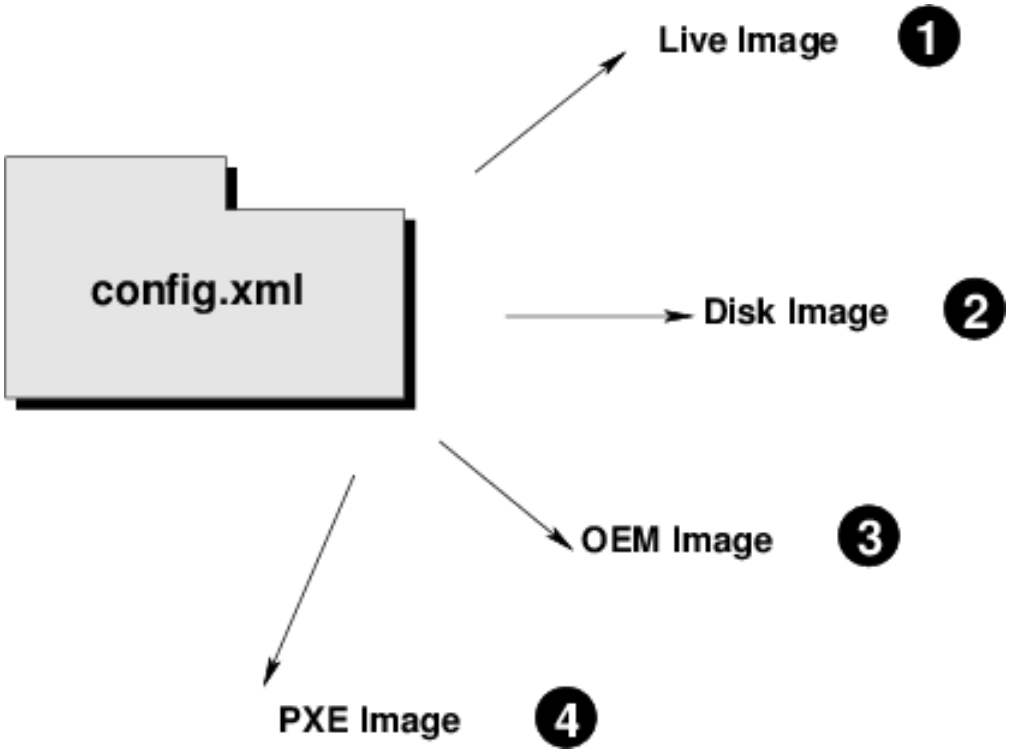
Successful completion of the *prepare* step is a the pre-requisite for the *create* step of the image build process. With the successful completion of the image preparation the unpacked root tree is considered complete and consistent. Creating the packed, or final image requires the execution of the *create* step. Multiple images can be created using the same unpacked root tree, for example it is possible to create a self installing OEM image and a virtual machine image from one unpacked root tree, under the condition that both image types are specified in the `config.xml` when the prepare step is executed. During the *create* step the following major operations are performed by kiwi:

- **Execute User Defined images.sh Script.**
  At the beginning of the image creation process the optional `images.sh` script is executed at the top level of the unpacked image directory. Unlike the `config.sh` script, the `images.sh` script does not have a target use case. The script is most often used to remove files that are no needed in the final image. For example if an appliance is being built that is targeted for specific hardware one can remove all unnecessary kernel drivers from the image 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.

- **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 contain at least one `type` element. The figure below shows the currently image types:

Figure 3.2. Image Types



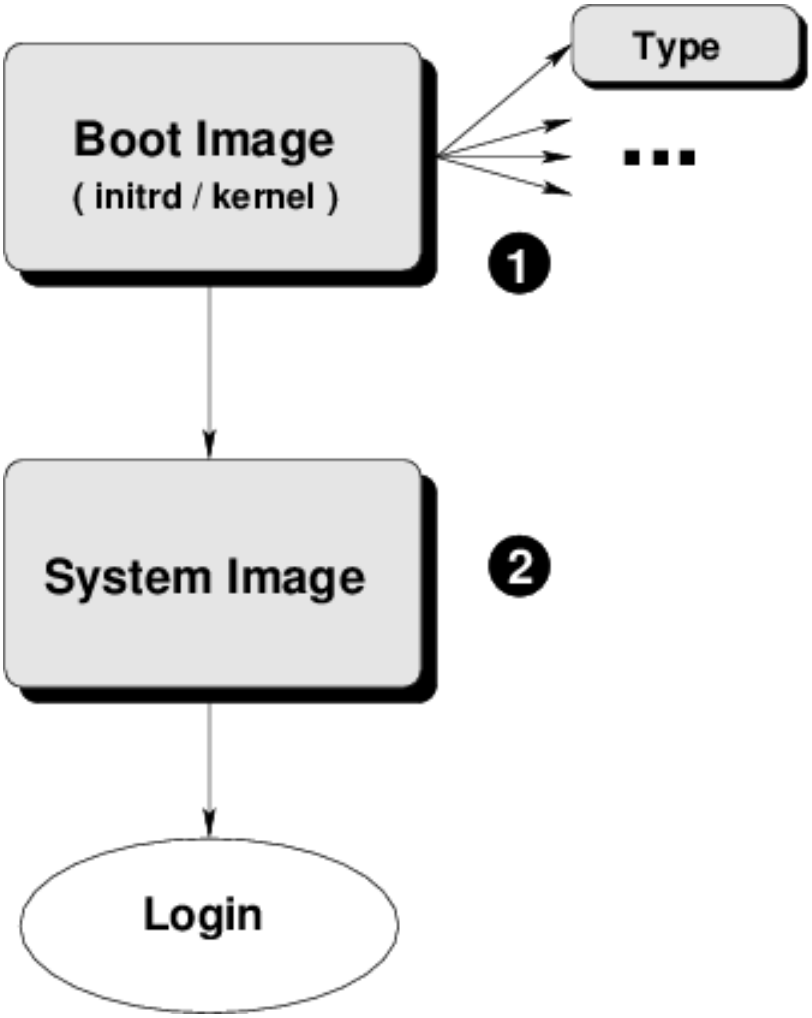
❶ Live Image on CD, DVD or USB stick
❷ Virtual system than can be used in VMware, Xen, Amazon Cloud, KVM, etc. virtual environments. Depending on the format a guest configuration file is created.
❸ Preload system for install media CD/DVD or USB stick
❹ Network boot image. KIWI also provides the bootp environment via the package kiwi-pxeboot

Detailed information, including step by step instructions about using the **kiwi** command and building specific images, as well as the configuration of the supported image types is provided later.

3.3. 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 ramdisk, into RAM and executes *init* or, if present, *linuxrc*. KIWI creates the boot image automatically depending on the image type as part of the *create* step in the image build process. Each image type has a specialized image description that describes 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 and are provided by KIWI. The boot image descriptions provided by KIWI cover almost all use cases and it should not be necessary for the majority of KIWI users to implement their own boot descriptions.

Figure 3.3. Image Descriptions



❶ Boot image descriptions are provided by KIWI, use is recommended but not required
❷ 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 to build 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 value of the `boot` attribute should be `oemboot/suse-SLES12`. The boot image description only represent the initrd and as such serves a limited purpose. The system image description created by the person building the image is ultimately the image that runs on the target system. Boot image descriptions are complete and consistent descriptions that allow you to build the boot image outside of the system image build process.

3.4. 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 is different for the various image types and provides hooks to execute user defined custom shell scripts. The shell scripts provided by the user may extend the first boot process and are expected to exist inside the boot image in a specific location with specific names. The naming and timing of the execution of the hook scripts is

image type dependent and described later. The instructions below explain the concepts of hook scripts, which is common to all image types, and how to include the scripts in the initrd.

- 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 tarball 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 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 have to have the executable bit set. Encoding the interpreter location with the #! comment is superfluous.

- 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 tarball 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. In order 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.

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. The hook 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 exist only for the *pxe* image type.

- **predownload|postdownload.** The hooks are called before and after the client image receives the root filesystem, respectively. The hooks exist only 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 exist only 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 exist only for the *pxe* image type.

- **premount|postmount.** The hooks are called before and after the client root filesystem is mounted, respectively. The hooks exist only for the *pxe* image type.
- **prenetwork|postnetwork.** The hooks are called before and after the client network is setup, respectively. The hooks exist only 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 exist only 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 exist only for the *pxe* image type.
- **preswap|postswap.** The hooks are called before and after the creation of the swap space, respectively. The hooks exist only for the *pxe* image type.
- **preactivate.** This hook is called before the root filesystem is moved to / The hook exists only for the *pxe* image type.
- **preCallInit.** This hook is called before the initialization process, init or systemd, is started. At call time the root filesystem has already been moved to /. The hook exists only 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 exists only 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 exists only for the *oem* image type.
- **preException.** This hook is called before a system error is handled, the actual 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 hook exists only for the *oem* image type.
- **preNetworkRelease.** This hook is called before the network connection is released. The hook exists only for the *pxe* image type.

The execution of hooks can be globaly deactivated by passing the following variable to the kernel commandline:

```
KIWI_FORBID_HOOKS=1
```

In addition to the hook script itself it's 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 one can pass to the kernel commandline to extend an existing hook script. There are the following rules for the processing of these information

- **The hook must activate the command post processing.** Post hook commands are only processed if the corresponding hook script activates this. The variable the hook script has to set follows the naming schema: KIWI_ALLOW_HOOK_CMD_|HOOKNAME| = 1 For example:

  ```
  KIWI_ALLOW_HOOK_CMD_preHWdetect=1
  ```

If this is set as part of the preHWdetect.sh hook script code the post command execution is activated

- **KIWI_HOOK_CMD_|HOOKNAME|.** The variable containing the command to become executed must match the following naming schema. For example:

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

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

- **KIWI_FORBID_HOOK_CMDS.** If this variable is set to something non empty the post hook command execution is deactivated however the basic hook script invocation is still active unless KIWI_FORBID_HOOKS is also set

3.5. 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. For the circumstances that require customized boot images KIWI provides mechanisms in the system image `config.xml` file to influence the boot image content. Using these mechanisms allows the user to still base the boot image on the KIWI provided descriptions rather than defining a completely new and custom boot image description. Creating a custom boot image that is not based on the KIWI provided descriptions is also possible. The following question and answer section provides solutions to the most common customization needs fro the initrd created by kiwi.

- **Why is the boot image so big and can I reduce it's size ?** KIWI includes all required tools and libraries to boot the image in all circumstances in the target environment for the image type. If target environment is well defined it is possible to remove data from that is known not to be needed. This will decrease the size of the initrd to and decrease boot time. Removing files in the boot image is accomplished by adding a `strip` section to the system image `config.xml` file, with the `type` attribute set to `delete`, as shown below.

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

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

- **Can drivers be added to the boot image?** KIWI uses a subset of the kernel. Should you encounter problems due to a missing driver that is part of the "standard" kernel but has not bee included by the kiwi build process you can add the driver by adding a `drivers` section to the system image `config.xml` file, as shown below.

```
<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 and it must be marked for boot image inclusion by setting the value of the `bootinclude` attribute of the `package` element to `true`, as shown below.

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

- **How to add missing tools/libraries?** Additional software can be added to the boot image with the use of the `bootinclude` attribute of the `package` element 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, any detectable dependencies of the files listed are preserved as well. The list of known 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 in your specific use case you need to instruct kiwi to not strip the files you have added to the boot image. This is accomplished by adding a `strip` section to the system image `config.xml` file, with the `type` attribute set to `tools`, as shown below.

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

the removal/preservation of files is name base and the path is immaterial. Therefore, you only have to specify the file name that is to be preserved.

- **Is it possible to add boot code?** Yes, as described in the Section 3.4, “Boot Image Hook-Scripts” section above, KIWI supports the execution of boot code at various times for various image types using *hook* scripts.

- **Is it possible to include completely custom boot code?** No. In cases where the provided hooks are insufficient and the KIWI provided boot code needs to be replaced completed 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.

3.6. 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.

- ***kiwidebug=1.*** If the boot process encounters a fatal error, the default behavior is to reboot the system 120 seconds. The “exception” behavior is changed by setting the `kiwidebug` parameter. With the value of the parameter set to 1 the system will enter a limited shell environment should a fatal error occur during boot. The shell contains the standard basic commands. The `/var/log/kiwi.boot` boot log file may be consulted to develop a better understanding of the boot failure. In addition to the spawned shell process kiwi also starts the dropbear ssh server if the environment is suitable. Support for ssh into the boot image is possible in the netboot and oemboot (in PXE boot mode) boot images. For isoboot and vmxboot boot images there is no remote login support because they don't setup a network. In order 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's required that the repo setup provides dropbear. Once dropbear is there the kiwi boot code will start the service. In order to access the boot image via ssh it's required to provide a public key on the pxe server in the directory: `server-root/KIWI/debug_ssh.pub`. kiwi only searches for that filename so it's required to name it “debug_ssh.pub”. Adding more than one public key to this file is possible exactly like the common SSH file “authorized_keys”.

The path “server-root” depends on what server type was configured to download the image. By default this is done via tftp. In that case the complete path to put the public key to is `/srv/tftpboot/KIWI/debug_ssh.pub`. on the pxe server. If ftp or http is used the server-root path is different. If a public key was found you can login as follows:

```
ssh root@<ip>
```

It might be useful to have a copy tool like scp or rsync as part of the boot image as well. Adding rsync as bootincluded package does not increase the size of the initrd much and would allow to extract e.g the kiwi boot log as follows:

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

3.7. Common and Distribution Specific Code

KIWI is designed to be in principal distribution independent and the majority of the kiwi implementation follows this design principal. However, Linux distributions differ from each other, primarily in the package management area as well as the 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 “-”, **suse-linuxrc** for example. When kiwi creates a boot image for a SUSE distribution the **suse-linuxrc** file from the boot discription is used as the **linuxrc** file that the Linux kernel calls.

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

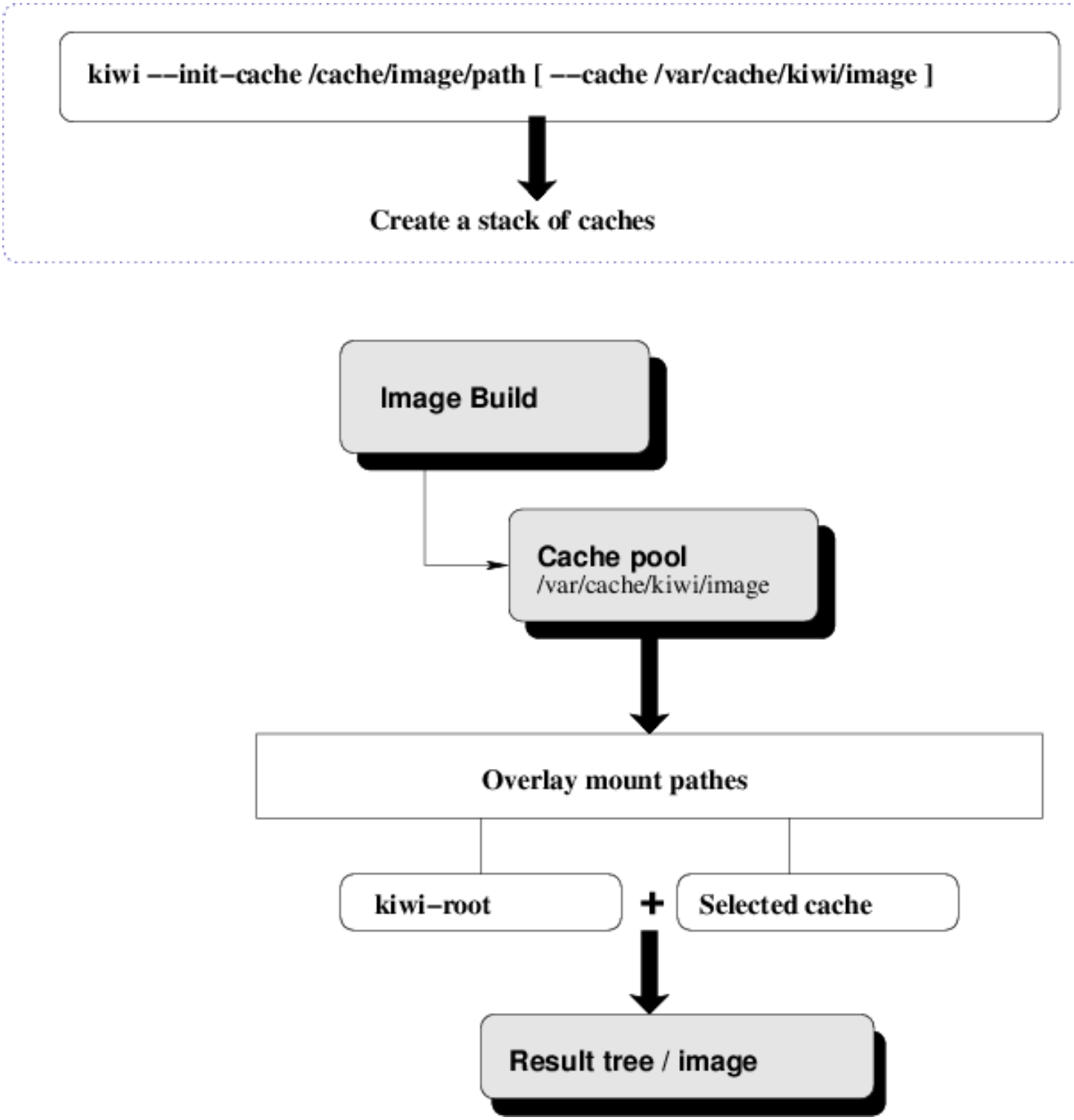
4 Image Caches

Table of Contents

4.1. Introduction

The process of creating an appliance could take quite some time and often the same software is installed over and over again. In order to speed up that process kiwi is able to 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 4.1. Image Caching Architecture



Before one can use a cache it needs to be created. A cache can be created from 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. But it might be more clever to create image descriptions for the purpose of caching. Such descriptions could represent a set of patterns for example. The less special 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 simultaniosly for other build processes. As result the build process doesn't start with an empty tree but with a tree almost complete. Only the missing parts are now installed and according to how much the cache already covered this process can speedup the build

4.2. Example

Let's say we know that we want to build some images of type 'vmx' and based on the SLES 12 JeOS image description. In order to create image caches for the system and the boot image the following steps needs to be done:

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 makes use of the cache the following command is used:

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

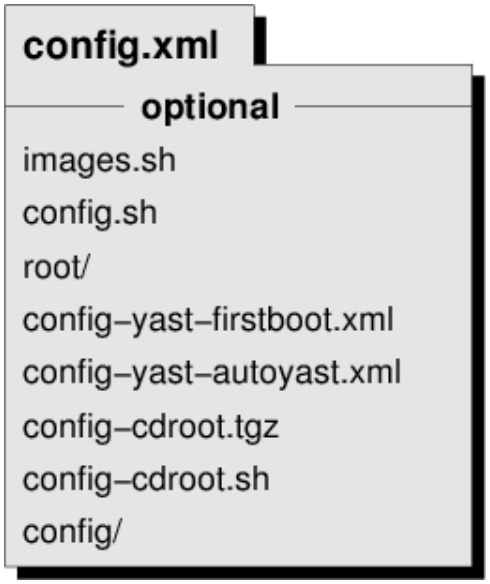
This call speeds up the build a lot compared to the creation without a cache. It's important to understand that a cache based build will create a root tree which contains only the differences compared to the used cache. Thus at any time you want to create an image out of it you have to make sure that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

5 KIWI Image Description

Table of Contents

In order 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 has 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`.

Figure 5.1. 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 in order 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.

5.1. 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`.

5.1.1. `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 bootloader. 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 bootloaders 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").

5.1.2. 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.

5.1.3. 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.

5.1.4. 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 11, *Linux Containers and Docker* 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 filesystems 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 bootloader 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 have 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. 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
: 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 priot 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 requiering 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. In order 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 verbatime.

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.

5.1.5. 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.

5.1.6. 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.

5.1.7. 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 5.1. Supported package manager repo 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 accomodate 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 primarly to install the license tarball if found on that repository. If no repository with a prefered 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 buildservice. 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 curent image.

5.1.8. 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.

5.1.8.1. 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 assigne one of the following values to the `patternType` attribute:

`onlyRequired`
: Incorporates only patterns and packages that the specifeid 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 explicitely. 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.

5.1.8.2. 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.

5.1.8.3. 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.

5.1.8.4. 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.

6 Creating Appliances with KIWI

Table of Contents

6.1. Overview

Traditionally, computing functions such as word processing or e-mail handling are delivered as software applications. These applications are targeted to run on a computer with an installed general purpose operating system. Applications often have a specialized installer that must be run by the consumer (whether home computer user or an administrator in an IT department of a company) to install the application on the computer in question. For installation of an application on multiple computers the installation program must often be run on each computer where the application is to be installed. In most cases a given application uses only a small part of the capabilities provided by the general purpose operating system running on a computer. Additionally if an application needs special settings to be applied to the general purpose operating system, these often have to be set by the consumer after the installation is complete. These settings are often documented in an installation guide that consumers may or may not read. Last but not least, running a general purpose operating system to support an application that only requires a small part of the functionality provided by the general purpose OS is a waste of computing resources.

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 optical media or a USB stick.

Compared to the traditional model of application delivery the appliance model has a number of advantages. The consumer no longer has to install a general purpose OS and the application in separate steps. The application is part of the appliance and the appliance provider, as the application expert, takes care of the application "installation". Further, the appliance provider takes care of any OS tuning that may benefit the application. Last but not least, the reduced size of the OS does not only consume fewer resources than a full blow "regular" install of a general purpose OS, but it also provides a reduced footprint for potential security exposure. From the application providers point of view there may be an opportunity to drop the implementation and maintenance of a specialized installer as the application installation no longer has to be "consumer friendly".

The traditional software delivery model certainly has it's place. However, for many purposes appliances present a more convenient mechanism for consumers.

6.2. The KIWI Model

With KIWI we started to use a different model. Instead of installing firewall software on top of a general purpose computer/operating system, the designers/engineers built images that are designed specifically for the task. These are so called appliances. When building appliances with KIWI the following proceeding has proven to work reliably. Nevertheless the following is just a recommendation and can be adapted to special needs and environments.

1. Choose an appropriate image description template from the provided KIWI examples. Add or adapt repositories, package names or both, 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 of non-binary changes. This is done by adding the following line into your **config.sh** script:

   ```
   baseSetupPlainTextGITRepository
   ```

3. Decide for a testing environment. In my opinion a real hardware based test machine which allows to boot from USB is a good and fast approach. All template images provided by us contains an hybrid iso type setup which is suitable for this environment.

4. Build the live stick appliance by calling

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

   After successful creation of the image files find an USB stick which is able to store your appliance and plug it into a free USB port on your image build machine. The deployment of the generated hybrid iso file can be performed from any OS including Windows as long as a tool to dump data onto a disk device exists and is used.

5. Plug in the stick on your test machine and boot it.

6. After your test system has successfully booted from stick login into your appliance and start to tweak the system according to your needs. This includes all actions required to make the appliance work as you wish. Before you start take care for the following:

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

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

   - Check the output of the command **git** `status` and include everything which is unknown to git and surely will not be changed by you and will not become part of the image description overlay files to the `/.gitignore` files

   After the initial package list exists and the git repository is clean you can start to configure the system. You never should install additional software just by installing an unmanaged archive or build and install from source. It's very hard to find out what binary files had been installed and it's 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.

7. As soon as your system works as expected your new appliance is ready to enter the final stage. At this point you have done several changes to the system but they are all tracked and should now become part of your image description. To include the changes into your image description the following process should be used:

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

```
rpm -qa | sort > /tmp/appliancePackages
diff -u /tmp/deployPackages /tmp/appliancePackages
```

  Add those packages which are labeled with (+) to the `<packages type="image">` section of your config.xml file and remove those packages which has been removed (−) appropriately. If there are packages which has been removed against the will of the package manager make sure you address the uninstallation of these packages in your **config.sh** script. If you have installed packages from repositories which are not part of your `config.xml` file you should also add these repositories in order to allow KIWI to install the packages

- Check the differences made in the configuration files. This can be easily done by calling:

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

  The created patch should become part of your image description and you should make sure the patch is applied when preparing the image. According to this the command:

```
patch -p0 < appliancePatch
```

  needs to be added as part of your **config.sh** script.

- Check for new non binary files added. This can be done by calling:

```
git status
```

  All files not under version control so far will be listed by the command above. Check the contents of this list make sure to add all files which are not created automatically to become part of your image description. To do this simply clone (copy) these files with respect to the filesystem structure as overlay files in your image description `root/` directory.

8. All your valuable work is now stored in one image description and can be re-used in all KIWI supported image types.

Congratulation! To make sure the appliance works as expected prepare a new image tree and create an image from the new tree. If you like you can deactivate the creation of the git repository which will save you some space on the filesystem. If this appliance is a server I recommend to leave the repository because it allows you to keep track of changes during the live time of this appliance.

6.3. Cross Platform Appliance Build

Building appliances for one processor architecture on another processor architecture is in general not possible with KIWI. The exception is that it is possible to build 32 bit (ix86) appliances on a 64 bit system running on the x86-64 architecture. This cross-platform limitation is based on the requirement that KIWI 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.

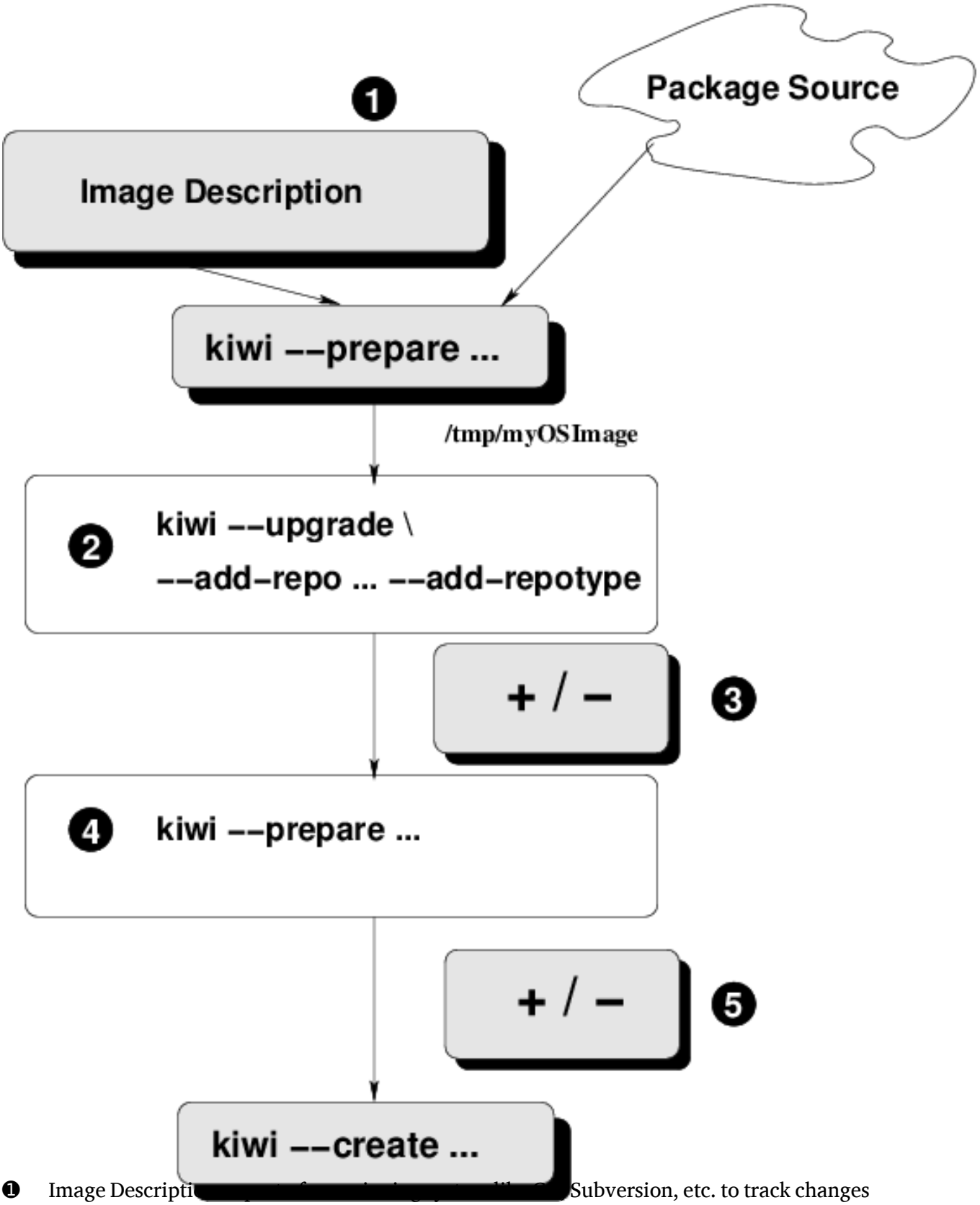
While KIWI has the `--target-arch` command line argument to instruct the package manager *zypper* to install packages for the specified architecture, this option is not intended to support cross-platform appliance builds.

Part II. Usecases

Table of Contents

7 Maintenance of Operating System Images

Creating an image often results in an appliance solution for a customer and gives you the freedom of a working solution at that time. But software develops and you don't want your solution to become outdated. Because of this together with an image people always should think of *image-maintenance*. The following paragraph just reflects ideas how to maintain images created by KIWI:



❶ Image Descripti Subversion, etc. to track changes
❷ Software package source changes
❸ Faster, because already prepared, cannot handle image description changes, requires free space to store `/tmp/myOSImage`
❹ Image Description changes
❺ Covers all possible changes, does not require storage for prepared trees, slower, because KIWI prepare runs again

The picture in Figure 7.1 shows two possible scenarios which requires an image to become updated. The first reason for updating an image are changes to the software, for example a new kernel should be used. If this change doesn't require additional software or changes in the configuration the update can be done by KIWI itself using its `--upgrade` option. In combination with `--upgrade` KIWI allows to add an additional repository which may be needed if the updated software is not part of the original repository. An important thing to know is that this additional repository is *not* stored into the original `config.xml` file of the image description.

Another reason for updating an image beside software updates are configuration changes or enhancements, for example an image should have replaced its browser with another better browser or a new service like apache should be enabled. In principle it's possible to do all those changes manually within the unpacked image root tree but concerning maintenance this would be a nightmare. Why, because it will leave the system in an unversioned condition. Nobody knows what has changed since the very first preparation of this image. So in short:

> Don't modify the unpacked image root tree manually!

Changes to the image configuration should be done within the image description. The image description itself should be part of a versioning system like git. All changes can be tracked down then and maybe more important can be assigned to product tags and branches. As a consequence an image must be prepared from scratch and the old unpacked image root tree could be removed.

8 System Analysis/Migration

Table of Contents

KIWI provides a module which allows you to analyse 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 process has the following limitations at the moment:

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

When calling KIWI's analysis mode it will try to find the base version of your operating system and uses the currently active repositories specified in the zypper database to match the software which exists in terms of packages and patterns. The result is a list of packages and patterns which represents your system so far. Of course there are normally some data which doesn't belong to any package. These are for example configurations or user data. KIWI collects all this information and provides it as custom data. In addition kiwi offers different data visualisations 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 filesystems.

8.1. Create a Clean Repository Set First

When starting with the analysis it is useful to let kiwi know about all the repositories from which packages has been installed to the system. In a first step call:

```
kiwi --describe workstation
```

This will create an HTML report where you can check which packages and patterns could be assigned to the given base repository. In almost all cases there will be information about packages which couldn't be assigned. You should go to that list and think of the repository which contains that packages (Packman, etc). If something is missing add it either to the zypper list on your system or use the KIWI options `--add-repo` … `--add-repotype`.

Continue calling the following command only if your list is clean and no skipped packages are used except you know that this package can't be provided or is not worth to become part of the description.

```
kiwi --describe workstation --nofiles [--skip package ... ]
```

8.2. Watch the Custom Files

Several reasons could lead to unmanaged data. In most cases these are user data like pictures, movies but also database files and external party software not installed as a package belongs to it. It's up to the user to decide if these data needs to be part of the description or not. Along with this important custom data there are unfortunately also a bunch of other custom data due to packaging inconsistencies or left over data as result of an upgrade process. These data taints your system and you are doing good in removing it. The quality of the description depends on how well the custom data tree is handled and how clean the system was when the analysis was started. Those data which should become part of the image description needs to be moved from the `/var/cache/kiwi/describe/worksation/custom` directory to the `/var/cache/kiwi/describe/worksation/root` directory

8.3. Checklist

After that you should walk through the following check list

- Change author and contact in `config.xml`
- Set appropriate name for your image in `config.xml`.
- Add/modify default type (oem) set in `config.xml` if needed
- If you want to access any remote filesystem it's a good idea to let AutoYaST add them on first boot of the system
- Check your 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.

8.4. Turn Into an Image...

After the process has finished you should check the size of the image description. The description itself shouldn't be that big. 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. Now let's try to create a clone image from the description. By default an OEM image which is a virtual disk which is able to run on real hardware too is created. On success you will also find a ISO file which is an installable version of the OEM image. If you burn the ISO on a DVD you can use that DVD to install your cloned image on another computer.

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

If everything worked well you can test the created OEM image in any full virtual operating system environment like Qemu or VMware™. Once created the image description can serve for all image types KIWI supports.

9 ISO Image / Live System

Table of Contents

A live system image is an operating System on CD or DVD. In principle one can treat the CD/DVD as the hard disk of the system with the restriction that you can't write data on it. So as soon as the media is plugged into the computer, the machine is able to boot from that media. After some time one can login to the system and work with it like on any other system. By default All write actions takes place in RAM space and therefore all changes will be lost as soon as the computer shuts down.

9.1. Building a live JeOS

This example is based on SLES version 12.

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

9.2. Using the Image

There are two ways to use the generated ISO image:

- 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 boot from the CD drive as first boot device.
- Use a virtualization system to test the image directly. Testing an iso can be done with any full virtual system for example:

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

9.3. Flavours

KIWI supports different filesystems and boot methods along with the ISO image type. The provided example by default uses an `overlayfs` based compressed root filesystem. overlayfs allows to combine two filesystems into one. The root filesystem exists as compressed squashfs filesystem and all write operations are redirected in RAM or in a persistent area on a disk.

The result is a full writable live-system. The flags attribute in `config.xml` exists to be able to have the following alternative solutions:

`flags="compressed"`
: Does filesystem compression with squashfs, but don't use an overlay filesystem for write support. A symbolic link list is used instead and thus a split element is required in `config.xml`. See the split mode section below for details.

`flags="seed"`
: Creates a `btrfs` image and allows write operations into a cow (seed) file. In case of an ISO the seed device is created on a ramdisk.

Flags Not Set
: If no `flags` attribute is set no compressed filesystem, no overlay filesystem will be used. The root tree will be directly part of the ISO filesystem and the paths: `/bin`, `/boot`, `/lib`, `/lib64`, `/opt`, `/sbin`, and `/usr` will be read-only.

9.3.1. Split mode

If no overlay filesystem is in use but the image filesystem is based on a compressed filesystem KIWI allows to setup which files and directories should be writable in a so called split section. In order to allow to login into the system, at least the `/var` directory should 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 split section can be used if the flag compressed is used:

```
<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>
```

9.3.2. Hybrid mode

A hybrid image is a iso image including a partition table and can therefore 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 as well as uploaded to a USB stick. In order to activate the hybrid feature the hybrid flag must be set to true as indicated below.

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

9.4. USB stick images

kiwi supports two types of USB stick images. The first type which are the hybrid ISO images and basically the same as the live ISO images and the second type which are the 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.

9.4.1. ISO Hybrid stick

As indicated above a hybrid iso image also works as USB stick image. If a hybrid iso is used like a disk image on a writable medium like a USB stick it's possible to write into a persistent area on the stick instead of the RAM. kiwi will create an additional ext2 partition to store that information on the disk if the attribute hybridpersistent is set to true.

```
<type image="iso" ... hybridpersistent="true"/>
```

9.4.2. OEM USB stick

In contrast to the hybrid iso image it's also possible to create a oem virtual disk image which is dumped on the stick. The big advantage with this approach is, that it's possible to create a stick which contains a live OS but also a data partition for custom data. The data partition is a fat partition also recognized by the Windows operating system. In order to create such a Windows friendly stick one has to pass the option `--fat-storage <size-in-MB>`.

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

If this option is set kiwi will use the syslinux bootloader for the image as well as the first partition as fat partition of the specified size. The live OS itself will live in a LVM which allows easy manipulation of the logical root volume. For further information about the OEM image type please refer to the OEM chapter Chapter 14, *OEM Image / Preload Systems*

9.4.2.1. OEM compressed / readonly USB stick

If a compressed filesystem type like overlayfs is used for the image root directory it's also possible to allow persistent writing on the USB stick or alternatively disallow that and let all write actions perfom in RAM only. kiwi provides the type attribute ramonly for this purpose. So in order to create a read-only oem stick with compressed root filesystem the following type section is required:

```
<type image="oem" filesystem="overlayfs" ramonly="true"/>
```

10 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 bootloader information. The size of this virtual disk can be specified by using the size element in the `config.xml` file or by adding the `--bootvm-disksize` command line argument. The virtual disk cannot expand to a different disk geometry. Be aware if you dump/copy the disk with another disk geometry, please refer to the Chapter 14, *OEM Image / Preload Systems* for additional details.

10.1. Building a JeOS disk

This example is based on SLES version 12. The JeOS system is based on the sles-Minimal pattern

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

10.2. Using the Image

The generated virtual disk image serves as the hard disk of the selected virtualization system (QEMU, VMware, etc.). The virtual hard disk format differs across virtualization environments. Some virtualization environments support multiple virtual disk formats. Using the QEMU virtualization environment test the created image with the following command:

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

10.3. Flavours

KIWI always generates a file in the `.raw` format. The .raw file is a disk image with a structure equivalent to the structure of a physical hard disk. Individual virtualization systems have specific formats to facilitate improved I/O performance to the virtual disk, represented by the image file, or additional specified virtual hard disk files. KIWI will generate a specific format when the format attribute of the type element is added.

```
<type image="vmx"... format="name"/>
```

The following table lists the supported virtual disk formats:

Table 10.1. 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

10.3.1. VMware support

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

Within the `config.xml` file it is possible to specify the VMware configuration settings. In addition it is possible to include selected packages in the created image that are specific to the VM image generation. The following *machine* section snippet provides general guidance on the VMware guest config which is defined below the *type* section in the `config.xml`.

```
<machine memory="512">
    <vmdisk controller="ide" id="0"/>
</machine>
```

Given the specification above KIWI will create a VMware guest configuration specifying the availability of 512 MB of RAM and an IDE disk controller interface for the VM guest. For additional information about the configuration settings please refer to the *machine* section.

The guest configuration can be loaded through VMware user interface and may be modified through the GUI. The configuration file has the `.vmx` extension as shown in the example below.

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

Using the `format="vmdk"` attribute of the `<type>` start tag will create the VMware formatted disk image (`.vmdk` file) and the required VMware guest configuration (`.vmx`) file.

10.3.2. 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 or add the attribute lvm="true" as part of the type section in your `config.xml` file. When using modern filesystems like btrfs, zfs kiwi also supports using their native volume management system. For more information how to setup custom partitions/volumes, see Chapter 17, *KIWI Custom Partitions* for a detailed explanation.

10.3.3. Extra Boot Partition

By default kiwi decides itself for using an extra boot partition or not. This basically depends on the selected root filesystem and the capabilities of the bootloader to directly read data from

it. By default kiwi will choose the save default even if the bootloader is able to read from e.g btrfs directly the default is still to use a simple ext filesystem for the /boot data. However it's possible to specify what kiwi should do with the two attributes `bootpartition="true|false"` and `bootfilesystem="ext2|ext3|ext4|fat32|fat16"`. A runtime check at build time will prove if the combination of attributes is technically possible.

11 Linux Containers and Docker

Table of Contents

Linux Containers (LXC) [http://lxc.sourceforge.net/] provide operating system-level virtualization, utilizing *Control Groups (cgroups)* [https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt]. The virtualization is similar to technologies in OpenVZ, Linux-VServer, FreeBSD jails, AIX Workload Partitions, and Solaris Containers. The network and process space of the container is separated from the host resources using namespaces. Kernel space information is mounted into the container root filesystem using the `fstab` file in the configuration directory. The container root filesystem provides the new user space on top of the running kernel of the host. A Linux container has two components: the container root file system stored in `/var/lib/lxc/`*`CONTAINER_NAME`* and the container configuration stored in `/etc/lxc/`*`CONTAINER_NAME`*. The kiwi created container image is packaged in a tarball containing the root filesystem and the configuration. The tarball is expected to be inflated at the root level (*/*) of the target system that functions as host.

Docker is a shipping container system for code that can run virtually everywhere. Basically is an extension of LXC's capabilities. As 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.

On top of LXC, Docker makes it possible to deploy portable containers across machines, shift focus on application rather than machines, includes versioning capabilities for tracking successive versions of a container, allows re-use of containers as a base for other specialized components, and much more. Find more information about Docker on its homepage at http://www.docker.io.

When building a docker image with kiwi *never* unpack the Docker tarball! If you unpack the tarball it will overwrite data on the host system. Use the **docker** command instead.

11.1. Building a docker image

This example is based on SLES version 12

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

11.2. Using the Image

The created container is packaged in a tarball in the destination directory, `my-container` and named `LimeJeOS-SLE12-docker.x86_64-1.13.1.tar.xz` In order to use this image with docker it must be imported via the **docker** command. The following requires to have docker installed and dockerd running.

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

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

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

11.3. 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.

The container itself must contain the Linux user space container tools.

Configure the network configuration for the container using the `vmnic` element. The `mode` attribute indicates the network mode, *veth* by default. While it is possible to configure multiple network interfaces in the `config.xml` file, the written container configuration will only reflect the information configured for the first `vmnic` element found in the `config.xml` file. The configuration for the container expects that the host has a configured network bridge with the name *br0*. For complex network setup implementations it is necessary to edit the config file.

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.

12 Vagrant boxes

Table of Contents

Vagrant [http://vagrantup.com] is a nice framework to implement consistent processing/testing work environments based on virtualisation technologies. In order to run a system vagrant needs so called boxes. A box is a tarball containing a virtual disk image and some metadata. If you need customized boxes you need to create them somehow. There is e.g. veewee [https://github.com/jedi4ever/veewee] which builds boxes based on `autoyast` , and Vagrant itself provides Packer [http://packer.io] . Both tools are based on the official distribution media (DVDs). If such media does not exist for reasons like the distro is still under development or you want to use a collection of your own repositories, the kiwi way of building images is 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.

12.1. Building a Base Box

This example is based on SLES version 12.

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

12.2. Box Configuration Details

The result in `/tmp/my-box` provides among other files the `.box` and the `.json` files which are needed in order to add and run the box in vagrant. The `.box` file is a tarball containing the actual virtual disk image for the selected virtualisation provider which is in our case a `.qcow2` image for use with `libvirt` and some metadata which mostly duplicates the information from the `.json` file to have it packaged in one place too.

The actual system inside of the virtual disk has to fulfill some requirements which are documented on the vagrant pages [http://docs.vagrantup.com/v2/boxes/base.html] . The kiwi template description already cared for this requirements which were:

- mandatory packages: `sudo` , `openssh` and `rsync`

- users `root` and `vagrant` both configured to use `vagrant` as password
- integration of `vagrant ssh` pubkey from here: https://github.com/mitchellh/vagrant/tree/master/keys
- setup of `sshd` with `UseDNS` set to no
- auto start of `sshd` at boot time
- `sudo` configured to allow passwordless root permissions for `vagrant` user

12.3. Using the box

This requires a correct vagrant installation on your machine including a running `libvirtd` and an up and running libvirt default network.

In order to add the box this can be done in two ways. First the traditional way by just using the `.box` file and providing a name at the commandline:

```
$ cd /tmp/my-box

$ vagrant box add my-box LimeJeOS-SLE12.x86_64-1.13.1.libvirt.box
```

Or secondly if you want the box to have metadata similar to the boxes downloaded from https://vagrantcloud.com/ (e.g. a version number), then instead of the above call:

```
$ cd /tmp/my-box

$ vagrant box add LimeJeOS-SLE12.x86_64-1.13.1.libvirt.json
```

With either method, you can now boot the box:

```
$ cd /tmp/my-box

$ vagrant init my-box

$ vagrant up --provider libvirt

$ vagrant ssh

This is the Lime-JeOS SLE12 Linux System...
vagrant@linux:~>
```

12.4. Vagrant with Docker

While it is required to build a specific disk image format for the libvirt, VMware or VirtualBox providers in vagrant, a docker base box would just be a tarball whose contents must match the vagrant box requirements listed above. Therefore building a docker base box for vagrant in kiwi is not different from just building a docker image as described in Chapter 11, *Linux Containers and Docker*

13 PXE Image—Thin Clients

Table of Contents

PXE is a boot protocol implemented in most BIOS implementations which makes it so interesting. The protocol sends DHCP requests to assign an IP address and after that it uses tftp to download kernel and boot instructions.

A PXE image consists of a boot image and a system image like all other image types too. But with a PXE image the image files are available separately and needs to be copied at specific locations of a network boot server.

13.1. Setting Up the Required Services

Before you start to build PXE images with KIWI, setup the boot server. The boot server requires the services atftp and DHCP to run.

13.1.1. Atftp Server

In order to setup the `atftp` server the following steps are required

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. Run `atftpd` by calling the command:

   ```
   rcatftpd start
   ```

13.1.2. DHCP Server

In contrast to the atftp server setup the following DHCP server setup can only serve as an example. Depending on your network structure, the IP addresses, ranges and domain settings needs to be adapted in order 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 information is provided by your server. The following steps describe how to setup a new DHCP server instance:

1. Install the package dhcp-server.

2. Create the file `/etc/dhcpd.conf` and include the following statements:

```
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:

```
rcdhcpd start
```

13.2. Building the suse-pxe-client Example

The example provided with KIWI is based on openSUSE and creates an image for a Wyse VX0 terminal with a 128MB flash card and 512MB of RAM. The image makes use of the `squashfs` compressed filesystem and its root tree is deployed as clicfs based overlay system.

```
cd /usr/share/doc/packages/kiwi/examples
==> select the example directory for the desired distribution change into it
cd suse-...
kiwi --build ./suse-pxe-client -d /tmp/mypxe-result --type pxe
```

13.3. Using the Image

In order to make use of the image all related image parts needs to be copied onto the boot server. According to the example the following steps needs to be performed:

1. Change working directory:

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

2. Copy of the boot and kernel image:

```
cp initrd-netboot-suse-*.splash.gz \
  /srv/tftpboot/boot/initrd
cp initrd-netboot-suse-*.kernel \
```

```
  /srv/tftpboot/boot/linux
```

3. Copy of the system image and md5 sum:

```
cp suse-*-pxe-client.* /srv/tftpboot/image
```

4. Copy of the image boot configuration. Normally the boot configuration applies to one client which means it is required to obtain the MAC address of this client. If the boot configuration should be used globally, copy the KIWI generated file as `config.default`:

```
cp suse-*-pxe-client.*.config \
  /srv/tftpboot/KIWI/config.MAC
```

5. Check the PXE configuration file. The PXE configuration controls which kernel and initrd are loaded and which kernel parameters are set. When installing the kiwi-pxeboot package, a default configuration is added. To make sure the configuration is valid according to this example, insert the following information into the file `/srv/tftpboot/pxelinux.cfg/default`:

```
DEFAULT KIWI-Boot

LABEL KIWI-Boot
    kernel boot/linux
    append initrd=boot/initrd vga=0x314
    IPAPPEND 1

LABEL Local-Boot
    localboot 0
```

6. Connect the client to the network and boot.

13.4. Flavours

All the different PXE boot based deployment methods are controlled by the `config.MAC` (or `config.default`) file. When a new client boots up and there is no client configuration file the new client is registered by uploading a control file to the TFTP server. The following sections informs about the control and the configuration file.

13.4.1. The PXE Client Control File

This section describes the netboot client control file:

```
hwtype.$<$MAC Address$>$
```

The control file is primarily used to set up new netboot clients. In this case, there is no configuration file corresponding to the client MAC address available. Using the MAC address information, the control file is created, which is uploaded to the TFTP servers upload directory `/var/lib/tftpboot/upload`.

13.4.2. The PXE Client Configuration File

This section describes the netboot client configuration file:

```
config.$<$MAC Address$>$
```

The configuration file contains data about image, configuration, synchronization, or partition parameters. The configuration file is loaded from the TFTP server directory `/var/lib/tftpboot/KIWI` via TFTP for previously installed netboot clients. New netboot clients are immedi-

ately registered and a new configuration file with the corresponding MAC address is created. The standard case for the deployment of a PXE image is one image file based on a read-write filesystem which is stored onto a local storage device of the client. Below, find an example to cover this case.

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

The following format is used:

```
IMAGE='device;name;version;srvip;bsize;compressed,...,'
CONF='src;dest;srvip;bsize;[hash],...,src;dest;srvip;bsize;[hash]'
PART='size;id;Mount,...,size;id;Mount'
DISK=device
```

`IMAGE`
: Specifies which image (name) should be loaded with which version (version) and to which storage device (device) it should be linked, e. g., `/dev/ram1` or `/dev/hda2`. The netboot client partition (device) hda2 defines the root file system `/` and hda1 is used for the swap partition. The numbering of the hard disk device should not be confused with the RAM disk device, where `/dev/ram0` is used for the initial RAM disk and can not be used as storage device for the second stage system image. SUSE recommends to use the device `/dev/ram1` for the RAM disk. If the hard drive is used, a corresponding partitioning must be performed.

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 blocksize 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. **Note:** The download will fail if you specify `"compressed"` and the image isn't compressed. It will also fail if you don't specify `"compressed"` but the image is compressed. The name of the compressed image has to contain the suffix `.gz` and needs to be compressed with the **gzip** tool. Using a compressed image will automatically *deactivate* the multicast download option of atftp.

`CONF`
: Specifies a comma-separated list of source:target configuration files. The source (src) corresponds to the path on the TFTP server and is loaded via TFTP. The download is made to the file on the netboot client indicated by the target (dest). Download only happens when configuration files are missing on the client or, if md5sum hash is supplied ([hash]), when different. To achieve this, list of CONF files (and VENDOR_CONF) files is kept on the client in the `/etc/KIWI/InstalledConfigFiles` backup file, and is compared to the CONF data gathered from the config.MAC and also from other configuration files, e.g. config.group, if supplied. Configuration files selected for comparison are those with same (dest) path. If destination path (dest) is same for more configuration files, only the last one is used (and VENDOR_CONF has always precedence to CONF). By comparing configuration file

lists present in the current CONF, VENDOR_CONF variables and stored in the backup file, following actions can result:

Table 13.1. Configuration files synchronization possibilities

cfg file in CONF,VENDOR_CONF	cfg file in InstalledConfig-Files backup	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 hash)
not present	present	delete on client (regardles hash)

Note that actual configuration files (or their md5sum hashes) on the client machine are not tested, only data from the backup file are used. This means that actual configuration files can be altered or even deleted without triggering any action, or, on the other hand, an action can be triggered without modifying the configuration files, only by modifying or removing of the backup file.

`PART`
Specifies the partitioning data. The comma-separated list must contain the size (size), the type number (id), and the mount point (Mount). The size is measured in MB by default. The mount specifies the directory the partition is mounted to.

- The first element of the list must define the swap partition.
- The second element of the list must define the root partition.
- The swap partition must not contain a mount point. A lowercase letter `x` must be set instead.
- If a partition should take all the space left on a disk one can set a lower `x` letter as size specification.

`RAID`
In addtion to the PART line it's also allowed to add a raid array setup. The first parmater 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 makes 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's required to specifiy 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;83;/'
```

`DISK`
Specifies the hard disk. Used only with PART and defines the device via which the hard disk can be addressed, e.g., `/dev/hda`.

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

`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. This is mostly the same as to perform a reboot but without the need of the BIOS or any bootloader. If FORCE_KEXEC is set to a non-empty string kiwi will also perform kexec if the kernel versions matches.

`RELOAD_IMAGE`
: If set to a non-empty string, this forces the configured 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 is sensible only for disk based systems.

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

`COMBINED_IMAGE`
: If set to a non-empty string, indicates that the both image specified needs to be combined into one bootable image, whereas the first image defines the read-write part and the second image defines the read-only part.

`KIWI_INITRD`
: Specifies the KIWI initrd to be used for local boot of the system. The variables 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 resides in the `boot/` directory below the tftp server path `/var/lib/tftpboot`. Because the tftp server do a chroot into the tftp server path you need to specify the initrd file as the following example shows:

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

`UNIONFS_CONFIG`
: For netboot images there is the possibility to use clicfs as container filesystem in combination with a compressed system image. The recommended compressed filesystem type for the system image is **clicfs**.

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

In this example the first device `/dev/sda2` represents the read/write filesystem and the second device `/dev/sda3` represents the compressed system image filesystem. The container filesystem clicfs is then used to cover the read/write layer with the read-only device to one read/write filesystem. If a file on the read-only device is going to be written the changes inodes are part of the read/write filesystem. Please note the device specifications in `UNIONFS_CONFIG` must correspond with 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,clicfs
```

As the second element of the PART list must define the root partition it's absolutely important that the first device in `UNIONFS_CONFIG` references this device as read/write device. The second device of `UNIONFS_CONFIG` has to reference the given IMAGE device name.

`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 `vga=0x317 splash=silent`.

`KIWI_BOOT_TIMEOUT`
Specifies the number of seconds to wait at the grub boot screen when doing a local boot. The default is 10.

`NBDROOT`
Mount the system image root filesystem remotely via NBD (Network Block Device). This means there is 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 the nbd-server man pages. 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 filesystem. 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 information. Whereas the other parameters are optional. The default device names are, NBD-Device = `/dev/nbd0`, NBD-Swap-Device = `/dev/nbd1` and NBD-Write-Device = `/dev/ram1` . The setup of swap and/R/W over nbd depends on if there are export names given or not. In addition a requested nbd swap space is only established if the client has less than 48 MB of RAM. The optional NBD-Write-Export-Name and NBD-Write-Device specifies a write COW location for the root filesystem. A separate write device is only used together with a union setup based on e.g overlayfs

`AOEROOT`
Mount the system image root filesystem remotely via AoE (ATA over Ethernet). This means there is a server which exports 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. In order to use the AoE subsystem I recommend to install the aoetools and vblade packages from here first: http://download.opensuse.org/repositories/server:/ltsp. Once installed the following example shows how to export the local `/dev/sdb1` partition via AoE:

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

Some explanation about this command, each AoE device is identified by a couple Major/Minor, with major between 0-65535 and minor between 0-255. AoE is based just over Ethernet on the OSI models so we need to indicate which ethernet card we'll use. In this example we export `/dev/sdb1` with a major value of 0 and minor of 1 on the eth0 interface. We are ready to use our partition on the network! To be able to use the device KIWI needs the information which AoE device contains the root filesystem. In our example this is the device `/dev/etherd/e0.1`. According to this the AOEROOT variable must be set as follows:

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

KIWI is now able to mount and use the specified AoE device as the remote root filesystem. In case of a compressed read-only image with `clicfs`, the AOEROOT variable can also contain a device for the write actions:

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

Writing to RAM is the default but you also can set another device like another aoe location or a local device for writing the data

`NFSROOT`
Mount the system image root filesystem remotely via NFS (Network File System). This means there is a server which exports the root filesystem of the network client in such a way that the client can mount it read/write. In order 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 as in the following example:

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

`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 the `/srv/tftpboot/boot/` directory. Because the TFTP server does a chroot into the TFTP server path, you must specify the initrd file as follows:

```
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` applies for the kernel setup:

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

`ERROR_INTERRUPT`
Specifies a message which is displayed during first deployment. Along with the message a shell is provided. This functionality should be used to send the user a message if it's clear the boot process will fail because the boot environment or something else influences the PXE boot process in a bad way.

13.4.3. User another than tftp as Download Protocol

By default all downloads controlled by the KIWI linuxrc code are performed by an atftp call and therefore uses the tftp protocol. With PXE the download protocol is fixed and thus you can't change the way how the kernel and the boot image (initrd) is downloaded. As soon as Linux takes over control the following download protocols http, https and ftp are supported too. KIWI makes use of the **curl** program to support the additional protocols.

In order to select one of the additional download protocols the following kernel parameters needs to be setup:

`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 setup this parameters edit the file `/srv/tftpboot/pxelinux.cfg/default` on your PXE boot server and change the append line accordingly. Please note 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.

13.4.4. RAM Only Image

If there is no local storage and no remote root mount setup the image can be stored into the main memory of the client. Please be aware that there should be still enough RAM space available for the operating system after the image has been deployed into RAM. Below, find an example:

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

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

13.4.5. Union Image

As used in the suse-pxe-client example it is possible to make use of the clicfs overlay filesystem to combine two filesystems into one. In case of thin clients there is often the need for a compressed filesystem due to space limitations. Unfortunately all common compressed filesystems provides only read-only access. Combining a read-only filesystem with a read-write filesystem is a solution for this problem. In order to use a compressed root filesystem based on clicfs make sure your `config.xml`'s filesystem attribute contains `clicfs`. As an alternative to clicfs kiwi also supports the fuse based unionfs utility. In contrast to clicfs which writes a block list on the write device, unionfs points all write operations into another filesystem which allows to mount and watch this location separately. In order to use a compressed root filesystem based on unionfs make sure your `config.xml`'s filesystem attribute contains `squashfs`. Below find examples for the different union modes.

13.4.5.1. Download 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,unionfs
KIWI_INITRD=/boot/initrd
```

13.4.5.2. Download 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,unionfs
```

13.4.5.3. Mount from Remote, Write to Local Storage

For all of the following modes I strongly recommend to check on a separate client machine in the network if it is possible to access the exported read-only and read-write device locations. If accessing devices works the image should also be able to access them on boot. If the boot fails it should be clear that the reason is not the exported device.

- NFSROOT

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

- AOEROOT

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

- NBDROOT

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

13.4.5.4. Mount from Remote, Write to RAM

- NFSROOT

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

- AOEROOT

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

- NBDROOT

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

13.4.5.5. Mount from Remote, Write to Remote

- NFSROOT

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

- AOEROOT

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

- NBDROOT

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

13.4.6. Split Image

As an alternative to the `UNIONFS_CONFIG` method it is also possible to create a split image and combine the two portions with the `COMBINED_IMAGE` method. This allows to use different filesystems without the need for an overlay filesystem to combine them together. Below find an example:

- Add a split type in `config.xml`, for example

```
<type fsreadonly="squashfs"
```

```
   image="split" fsreadwrite="ext3" boot="netboot/suse-..."/>
```

- Add a split section inside the type to describe the `temporary` and `persistent` parts. For example:

```
<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>
```

- Sample `config.`*`MAC`*:

```
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
```

13.4.7. Root Tree Over NFS

Instead of installing the image onto a local storage device of the client it is also possible to let the client mount the root tree via an `NFS` remote mount. Below find an example:

- Export the KIWI prepared tree via NFS.
- Sample `config.`*`MAC`*:

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

13.4.8. Root Tree Over NBD

As an alternative for root over NFS it is also possible to let the client mount the root tree via a special network block device. Below find an example:

- Use nbd-server to export the KIWI prepared tree.
- Sample `config.`*`MAC`*

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

13.4.9. Root Tree Over AoE

As an alternative for root over NBD it is also possible to let the client mount the root device via a special ATA over Ethernet network block device. Below find an example:

- Use the **vbladed** command 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 like with NBD.
- Sample `config.MAC`:

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

  This would require the following command to be called first:

  ```
  vbladed 0 1 eth0 blockdevice
  ```

13.5. Hardware Grouping

While the PXE standard takes care of the ability to create hardware groups via hardware or IP address groups, it does not take into account groups for non-contiguous hardware or IP addresses. The PXE standard makes the assumption that each hardware group will be clearly delineated by a range of IP addresses, or the hardware is from the same vendor. While an ideal scenario, this may not be the case in an established, slightly dated installation where the hardware itself has out-lived the vendors that made them.

KIWI has the ability to create groups for non-contiguous configurations where different hardware types may be involved due to newer equipment being rotated into production or older hardware failing and replacements are from different vendors. In addition, an organization might decide to organize their equipment by function, rather than by vendor, and may not be able to use the same hardware from one end to the other.

13.5.1. The Group Configuration File

To make use of the grouping functionality, some new configuration files will be required. These configuration files currently have to be manually managed rather than provided, however future versions of KIWI may provide a means of managing groups more effectively once this feature stabilizes. The number of configuration files required will depend on the number of hardware groups that will be created, rather than one configuration file for each MAC address that will reside on the network.

There will be one configuration file that will always be required if using groups, called:

```
/srv/tftpboot/KIWI/config.group
```

This file has a new static element that must exist, and one or more dynamic elements depending on the number of groups that will be created. For example, the config.group file defined below lists 3 distinct groups:

```
KIWI_GROUP="test1, test2, test3"

test1_KIWI_MAC_LIST="11:11:11:11:11:11, 00:11:00:11:22:CA"

test2_KIWI_MAC_LIST="00:22:00:44:00:4D, 99:3F:21:A2:F4:32"

test3_KIWI_MAC_LIST="00:54:33:FA:44:33, 84:3D:45:2F:5F:33"
```

Note: The above hardware addresses contain random entries, and may not reflect actual hardware.

As we can see in the above example the file contains 1 static element, KIWI_GROUP, and 3 dynamic elements "test1_KIWI_MAC_LIST, test2_KIWI_MAC_LIST and test3_KIWI_MAC_LIST". The definitions of these elements are as follows:

- KIWI_GROUP

  This element is the only static definition that needs to exist when using groups. While there is no implicit limit to the number of groups that can be configured, it should be kept to a minimum for reasonable management or it could quickly become un-manageable. It will need to contain one or more group names separated by comma's (,) and spacing (for readability). In the above example, our group names were:

  - test1
  - test2
  - test3

  Valid group names are made up of upper and lower case letters, and can use numeric, and underscore characters. The same rules used to define bash/sh variable names should apply here, as these names will have to be used as fully defined bash/sh variables when linking hardware addresses to an assigned group. The following is an example that contains valid names:

  ```
  KIWI_GROUP="test1, test_my_name, LIST_HARDWARE, Multple_Case_Group_1"
  ```

- <GROUP_NAME>_KIWI_MAC_LIST

  The name of this element is dynamic and depends entirely on the list of group names that were previously defined. Each group name that was used in the KIWI_GROUP variable, must contain a matching dynamic element, and have KIWI_MAC_LIST appended to the name. To continue with our previous example, to create hardware lists for the groups already defined, we need 3 dynamic elements called:

  - test1_KIWI_MAC_LIST
  - test2_KIWI_MAC_LIST
  - test3_KIWI_MAC_LIST

  These variables will contain a comma delimited list of the hardware addresses for all of the machines being assigned to the appropriate group, but there are some caveats that need to be kept in mind. The first caveat is for hardware addresses that contain the HEX characters A-F. The PXE standard uses capital letters for these characters, and as a result KIWI does upper case comparisons, so a MAC address that is defined with lower case letters in this list will never get matched.

  The second caveat is that as the list gets longer, it can be harder to maintain and it has the potential to slow down the booting process. However, testing has been completed with 1500+ hosts defined, and there was little delay when transferring the file to a single host. The file size will have a larger impact when trying to download it to 1500+ hosts, so some consideration will have to take that into account. The comparison itself still occurred in under half a second while searching through all 1500+ MAC addresses across 3 defined groups.

13.5.2. The Group Details File

In addition to the `config.group` file, each defined group will require a `config.<GROUP_NAME>` file. This file is exactly like a standard KIWI config.<MAC> file, but is assigned to a group

of hosts rather than a single unit. If we continue with the example we used in the previous section, we would need the following files:

```
/srv/tftpboot/KIWI/config.test1
/srv/tftpboot/KIWI/config.test2
/srv/tftpboot/KIWI/config.test3
```

The contents of these files is the same that would normally reside in a config.<MAC> file, and all definitions that would be supported for a single host, are supported for a group of hosts. In addition, if a host is matched to a group, yet the config.<GROUP_NAME> file does not exist, KIWI will error out.

For example, the following configuration file, called `config.test1` would be used for the group called "test1":

```
DISK=/dev/sda
PART='5;S;x,x;L;/'
IMAGE='/dev/sda2;suse-##.#-pxe-client.i686;1.2.8;192.168.100.2;4096'
CONF='CONFIGURATIONS/xorg.conf.test1;/etc/X11/xorg.conf;192.168.100.2;4096,\
     CONFIGURATIONS/syslog.conf;/etc/sysconfig/syslog.conf;192.168.100.2;4096'
```

As a result of this configuration file, the image would be configured consistently across all the hosts assigned to test1. The following file called `config.test2`, contains a small change that may be specific to a function:

```
DISK=/dev/sda
PART='5;S;x,x;L;/'
IMAGE='/dev/sda2;suse-##.#-pxe-client.i686;1.2.8;192.168.100.2;4096'
CONF='CONFIGURATIONS/xorg.conf.test2;/etc/X11/xorg.conf;192.168.100.2;4096,\
     CONFIGURATIONS/syslog.conf;/etc/sysconfig/syslog.conf;192.168.100.2;4096'
```

As we can see, while group 1 and 2 share the syslog.conf configuration file, they have different `xorg.conf` files defined, therefore two distinct groups with one or more hosts assigned to each group can now be configured by managing a smaller number of files.

13.5.3. Using Hardware Mapping to Provide Overrides

The only issue with running mixed hardware configurations pertains primarily to hardware differences. For instance, it may be possible to create a single, `xorg.conf` file that is able to work with all of the hardware, but there is a chance it might not be possible to do so. With this in mind, KIWI provides a mechanism to provide "default" configurations that works with the most common hardware configuration, while providing hardware specific overrides to allow for any differences and yet have all hardware linked to the same group.

13.5.3.1. The Hardware Mapping Elements

To make use of the hardware linking mechanism, two additional parameters needs to be added to the group details file, the one named `config.<group_name>`. These two elements "link" hardware specific configurations to the appropriate systems. A general example would look like this:

```
HARDWARE_MAP="vendor_name_model"
vendor_name_model_HARDWARE_MAP="00:00:00:11:11:11"
```

These parameters are not required, and the same functionality can be applied by using multiple groups to do the same thing, but that might not be desirable to some administrators.

This feature allows for a slightly more complex group to be defined, but the end result is a single group, that can contain multiple sub-groups ensuring flexibility in using a mixed set of hardware.

The definitions for the above parameters are as follows:

- 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_NAME>_HARDWARE_MAP

  This element behaves exactly like the <GROUP_NAME>_KIWI_MAC_LIST element defined above, in that it lists all MAC addreses 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.

13.5.3.2. The Hardware Mapping Details File

Once 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 contents of the file is quite simple, and contains only one element called VENDOR_CONF, as the following example shows:

```
VENDOR_CONF='CONFIGURATIONS/xorg.conf.hardware_name_model;/etc/X11/xorg.conf;192.168.100.2;4096
```

The format of the VENDOR_CONF values is exactly the same as the CONF variable used in the standard host and group configurations. In addition, files defined within this list will overwrite any files defined in the group configuration, if and only if, all of the following cases apply:

- 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 overwritting will take place as it is considered a new file.

13.5.3.3. A Complete Example

The following is an example of a group that is using hardware from multiple vendors. For the purposes of this example, lets assume the group will have 10 defined hosts, seven are imaginative HP thinstations, while the remaining three are older Maxterm thinstations. We will also assume that the differences we are trying to address are specific to the video card and X.Org drivers used as a result.

With this in mind, we will need the following KIWI specific files:

```
cd /srv/tftpboot/KIWI
ls
   config.example1
   config.group
   hardware_config.maxterm_3500
```

As we can see, there is a KIWI group file, the group configuration or details file, and a new file that we have not seen before called hardware_config.maxterm_3500. We will first look at the contents of the config,group file:

```
cat config.group

KIWI_GROUP="example1"
example1_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"
```

Within the file, there is a group called "example1", with ten hosts defined, in this case with imaginary sequential MAC addresses. Next, we look at the config.example1 group details/configuration file:

```
cat config.example1

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,clicfs
CONF='prefs.js;/home/kioskuser/.mozilla/firefox/07xvl1ty.default/prefs.js;192.168.1.2;4096,xorg.conf;/
RELOAD_IMAGE=yes
RELOAD_CONFIG=yes
HARDWARE_MAP='maxterm_3500'
maxterm_3500_HARDWARE_MAP='00:00:00:00:00:02 00:00:00:00:00:03 00:00:00:00:00:04'
```

Here, most of the standard KIWI configuration elements are in place, with a few extras. There are three areas we want to focus our attention on, the CONF, HARDWARE_MAP and maxterm_3500_HARDWARE_MAP variables, as they are the most critical elements to our example.

The first parameter to look at is the CONF parameter, which indicates a `prefs.js` (for Mozilla Firefox), and a xorg.conf (for X Windows) files will be copied to the host during boot up. These files should be considered defaults for the group, and all hosts defined in this group will use these files. As such, when the systems boot, both of these files will be copied over to their local file systems when the CONF element is processed.

Lastly, we have a hardware mapping group called "maxterm_3500", with three of the groups hosts defined as part of of a sub-group, or hardware map. The content of this file is as follows:

```
cat hardware_config.maxterm_3500

VENDOR_CONF='xorg.conf.maxterm_3500;/etc/X11/xorg.conf;192.168.1.2;4096,
            someconfig.cfg;/etc/sysconfig/someconfig.cfg;192.168.1.2;4096'
```

When the VENDOR_CONF defintition 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, any files listed here will be transferred to a host if, and only if, the host has been linked to the hardware map via the maxterm_3500_HARDWARE_MAP element. In our example the only systems that

will receive the xorg.conf.maxterm_3500 file will be the three maxterms we linked to the hardware map itself.

In our VENDOR_CONF element, we are indicating two files that should be transferred, in addition to any file transferred during the processing of the CONF element. A "specific" `xorg.conf` file, as well as someconfig.cfg. In the case of the xorg.conf.maxterm_3500 file, when it is transferred to the host, it will overwrite the `xorg.conf` file that was previously transferred via the CONF element. However, with the `someconfig.cfg` file, because it was not previously defined in the CONF element, it will simply get transferred over, and is a perfect example of how one could enable functionality that is not otherwise configured.

As a result of this example, we have seven terminals that are using a prefs.js and generic xorg.conf file for their system configuration, and three terminals that are using prefs.js, a new version of the `xorg.conf` file as well as a file called somconfig.cfg. For the purposes of our example, the contents of the `prefs.js`, `xorg.conf`, xorg.conf.maxterm_3500 and `someconfig.cfg` are arbitrary, and don't need to be explained here.

14 OEM Image / Preload Systems

Table of Contents

An OEM image is a virtual disk image representing all partitions and bootloader information in the same fashion it exists on a physical disk. All flavors discussed previously for the VMX image type apply also to the OEM image type. The image format matches the format of the VMX image type, however an OEM image can do more. It is able to 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.

14.1. Building an OEM System with Installation DVD

This example is based on SLES version 12. The image creation process actually creates two images. The OEM disk image and an Installation ISO image which contains the OEM disk image. As a user I can decide to take the OEM disk image and dump it with some tool on the disk of the target system or boot the target system from the ISO image and run through an image deployment process which could also be configured to run without user interaction.

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

14.2. Using the Image

The virtual disk image created by KIWI with the commands shown above can be tested using virtualization software such as QEMU, VMware, or VirtualBox. The virtual disk is represented by the file with the `.raw` extension, whereas the file with the `.iso` extension represents the installation disk for this oem image. The ISO image is bootable (`filename.iso`) and can be burned to optical media. It is recommended to test the image on a bare test system. The following command shows how to use QEMU to test the OEM disk image (`filename.raw`).

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

or using the **dd** command you can dump the image onto a test hard disk or USB stick and upon reboot select the appropriate device as the boot device in the BIOS:

```
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, i.e. 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.

You can test the installation procedure in a virtual environment using the `.iso` file. In this case the re-partitioning code in the boot image will be executed. The following commands show this procedure using QEMU.

```
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
```

14.3. Flavours

As indicated above the use of the `installiso` and `installstick` attributes for the oem image supports the creation of an installation image. The installation image can be created in two formats, one suitable for CD/DVD media and a second suitable for a USB stick. The self installing image 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 following configuration snippets show the use of the `installiso` and `installstick` attributes to create the ISO or USB installation image format respectively.

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

  Creates a `.iso` file which can be burned onto a CD or a DVD. This represents an installation CD/DVD. You might have noticed the *hybrid* attribute. This turns the ISO into a hybrid media which let look the ISO like an ISO and a disk. Therefore it's possible to dump the ISO on for example an USB Stick. If your bios is not confused about the hybrid setup kiwi did with the ISO, the stick will boot like a disk. If it does not work use the installstick media as explained below.

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

  Creates a `.raw.install` file which can be dumped (**dd**) onto a USB stick. This represents an installation Stick

14.3.1. Specializing the OEM install process

It is possible to specialize the OEM install process by providing shell scripts with the following names. For more information how to pack the scripts and make them work in the boot code, see the chapter Section 3.4, “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 once the image checksum has been successfully verified.

14.3.2. Influencing the OEM Partitioning

By default the oemboot process will create a swap and an optional recovery partition, and will expand all other partitions according to the setup in the kiwi configuration and the underlaying disk geometry. It is possible to influence the behavior with the `oem-*` elements. See Chapter 5, *KIWI Image Description* for details.

14.3.3. Partition Based Installation

The default installation method of an OEM is dumping the entire virtual disk on the selected target disk and repartition the disk to the real geometry. This works but will also wipe everything which was on the disk before. KIWI also supports the installation into already existing partitions. This means the user can setup a disk with free partitions for the KIWI OEM installation process. This way already existing data will not be touched. In order to activate the partition based install mode the following OEM option has to be set in `config.xml`:

```
<oem-partition-install>true</oem-partition-install>
```

Compared to the disk based install the following differences should be mentioned:

- The bootloader will be setup to boot the installed system. There is no multiboot setup. The user is expected to implement the setup of a multiboot bootloader.
- The oem options for system, swap and home doesn't have any effect if the installation happens in predefined partitions. In this mode kiwi will not create additional partitions.
- There is no support for remote (PXE) OEM installation because KIWI has to loop mount the disk image and need to address specific regions inside of the image. These block operations are not implemented for remote access

14.3.4. Network Based Installation

Instead of manually dumping the OEM image on the target device or creating a KIWI install CD, USB stick, there is a third method of deploying the OEM image on the target device. It's possible to let the image be downloaded from a PXE boot server over the network. This requires a PXE network boot server to be setup properly in the first place For details how to do this refer to the chapter: Chapter 13, *PXE Image—Thin Clients*. If your pxe server is running the following steps are required to setup the install process over the network

- Make sure you have created an install PXE tarball along with your oem image:

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

- unpack the created <image-name>.tgz file and copy the initrd and kernel images over to your PXE server

  ```
  tar -xf <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
  ```

- Next copy the system image and md5 sum over to 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/
```

- At last set the kernel commandline parameters to the append line in your PXE configuration (for example: `pxelinux.cfg/default`). The required parameters are stored in the file `<image-file>.append` from the KIWI generated install tarball

  Optionally the image can be stored on a FTP,HTTP server specified via the **kiwiserver** and **kiwiservertype** append information. In this case make sure you copied the system image and md5 file to the correct location on the ftp, http, etc. server. KIWI searches the image at one place only which is below the `image/` directory on the root path of the specified server. initrd and linux kernel are loaded by PXE thus they require a tftp server to be present.

15 Xen Para- and Full virtual Images

Table of Contents

Xen is a free software virtual machine monitor. It allows several guest operating systems to be executed on the same computer hardware at the same time.

A Xen system is structured with the Xen hypervisor as the lowest and most privileged layer. Above this layer are one or more guest operating systems, which the hypervisor schedules across the physical CPUs. The first guest operating system, called in Xen terminology “domain 0” (dom0), is booted automatically when the hypervisor boots and given special management privileges and direct access to the physical hardware. The system administrator logs into dom0 in order to start any further guest operating systems, called “domain 0” (domU) in Xen terminology.

A Xen image is a virtual disk like a vmx but with the xen kernel installed for dom0 or para virtual guest images. For full virtual guest images any kernel e.g kernel-default can be used together with a xen kernel modules packages which must be available for the used kernel

In order to run it a Xen dom0 server needs to run. A xen guest is booting via a surrounded boot infrastructure. For paravirtual images pyGrub or pvGrub can be used, while for HVM an extra hvmloader is used. Xen extracts information to boot from the given image and boots the guest. Depending on what type of guest is booted also the bootloader configuration could be read. Thus this put some constraints on the configuration which are addressed by kiwi

15.1. Building a Dom0

This example is based on SLES version 12. The example provides a xenFlavour profile which builds a dom0 image for the oem image type as follows

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

15.2. Using the Dom0 Image

Basically the dom0 represents the most privileged layer with access to the hardware. It is possible to run such an image inside of a full virtual system like Qemu but this is not recom-

mended. First and foremost because the performance suffers from doing so and secondly because kernel-xen is not officially supported to work within a stack of hypervisors. For testing however it's ok to just run this oem image in Qemu as follows:

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

Once installed *mydom0* is a Xen dom0 from which other xen guests can be started

15.3. Building a Para Virtual Xen Guest

This example is based on SLES version 12. The example makes again use of the xenFlavour profile but builds a simple vmx image. The result is a disk image with kernel-xen prepared for paravirtual boot via grub2. In order 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
```

15.4. Building a Full Virtual Xen Guest

This example is based on SLES version 12. In contrast to the paravirtual guest image this example builds a simple vmx image including the standard kernel plus some xen required kernel modules. In order 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
```

15.5. Using the Guest Images

In order to run a domain U the Xen tool **xm** needs to be called in conjunction with a domain U configuration file like the following example:

```
xm create -c config-file
```

For paravirtual guest images KIWI supports the creation of the configuration file according to information given in the KIWI `config.xml`

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

If this information exists kiwi creates a file with the extension `.xenconfig`

Please note that not all possible configuration options are supported with KIWI xen config file creator. For hvm images there is currently no support to create the configuration from kiwi. However tools like *virt-manager* support you with setting up the machine configuration.

More details how to configure the guest image with Xen is provided on http://www.xenproject.org/help/documentation.html

16 KIWI RAID Support

KIWI supports three types of RAID systems:

Real RAID controllers with its own firmware
: KIWI only has 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's because an image consists initially out of one disk and not more. The user should add devices or change the RAID mode manually after deployment. This is an easy task if the system comes up prepared for all this. In order to use linux software raid in kiwi images you only have to set:

<type ... mdraid="mirroring">

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

17 KIWI Custom Partitions

Table of Contents

KIWI supports custom parttions only via LVM, the logical volume manager for the Linux kernel, or filesystems with volume support like btrfs or zfs.

17.1. Custom Partitioning via LVM

KIWI supports LVM, the logical volume manager for the Linux kernel that manages disk drives and similar mass-storage devices. KIWI supports custom parttion only via LVM or filesystems with volume support like btrfs or zfs.

To define a 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 the name specifies a path which will be created by kiwi if it does not exist inside the root directory. However if the name contains the kiwi internal path field separator '_', it's required to specify the path in an additional mountpoint attribute. KIWI also makes sure that the volumes are created in the correct order of the filesystem hierarchy. The special attribute *@root* can be used 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. The value is used as MB by default but you can add "M" and/or "G" as postfix.

`freespace`
: An optional attribute. Free space to be added to this volume. The value is used as MB by default but you can add "M" and/or "G" as postfix.

`mountpoint`
: An optional attribute. The mountpoint specifies a path which has to exist inside the root directory.

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

```
<systemdisk name="vgroup-name">
    <volume name="tmp"/>
</systemdisk>
```

To do the same but with 200 MB of size, use:

```
<systemdisk name="vgroup-name">
    <volume name="tmp" size="200M"/>
</systemdisk>
```

To create the logical volume named 'foo' >with 200 MB of free space mounted as `/tmp`, use:

```
<systemdisk name="vgroup-name">
    <volume name="tmp" freespace="200M" mountpoint="tmp"/>
</systemdisk>
```

There are always the volumes LVRoot and LVSwap for the rootfs and the swap space. In order to influence LVRoot one can use "@root" as a name:

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

17.2. Custom Partitioning via Btrfs

If Btrfs is used as a filesystem, the subvolume management is configured via the same `systemdisk` element as in the case of LVM. Also the same rules as explained for lvm volumes applies to btrfs subvolumes with the following exceptions:

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

18 KIWI Encryption Support

KIWI supports LUKS encrypted images. To setup cryptographic volume along with the given filesystem using the LUKS extension, add the parameter luks in the type element in your config.xml. The value of the parameter represents the password string used to be able to mount that filesystem while booting:

<type ... luks="password">

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 is able to 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 filesystem 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 is able to 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 analyse 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 backend 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 analyse 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 a crypted 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 bootloader.

`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 packagemanager. 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 packagemanager 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 commandline 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 buildservice. 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 circumstations.

[--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 has 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 blocksize 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 filesystems and in blocks if the reiser filesystem 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 filesystem is used

[--fs-inoderatio *number*]
Set the bytes/inode ratio. This option has no effect if the reiser filesystem is used

[--fs-max-mount-count *number*]
When calling kiwi in creation mode this option will set the number of mounts after which the filesystem will be checked. Set to 0 to disable checks. This option applies only to ext[234] filesystems.

[--fs-check-interval *number*]
When calling kiwi in creation mode this option will set the maximal time between two filesystem checks. Set to 0 to disable time-dependent checks. This option applies only to ext[234] filesystems.

[--fat-storage *size in MB*]
if the syslinux bootlaoder 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 bootloader is installed. This allows to modify the bootloader configuration file written by kiwi. The scripts working directory is the one which represents the image structure including the bootloader configuration files. Please have in mind that according to the image type, architecture and bootloader type the files/directory structure and also the name of the bootloader configuration files might be different.

[--edit-bootinstall *script*]
Specifies the location of a custom script which is called right after the bootloader 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 device instead of the loop device.

For More Information

More information about KIWI, its files can be found at:

http://en.opensuse.org/Portal: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}]
Deactive 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 packagemanager

[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 makes use of 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 filesystem 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` exitswith 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 filesystems `/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 in order 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 in order 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 makes use of 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 filesystem 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 toolchain 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 filesystem.

[System]
: Specify the location of the KIWI system image description.

  The default value is the value of BasePath concatenated with /image.

Index

Symbols

A

B

C

D

E

F

H

I

K

M

N

O

P

R

S

U

V

X