KIWI Project

openSUSE - KIWI Image System Cookbook

Project, Design and Implementation
by Marcus Schaefer (ms@suse.de)
*

Author: Marcus Schaefer
Date: June 9, 2009
Version: 3.57

*SUSE LINUX Products GmbH, Maxfeldstrasse 5, D-90409 Nuremberg

Contents

1 Introduction

The openSUSE KIWI Image System provides a complete operating system image solution for Linux supported hardware platforms as well as for virtualisation systems like Xen, VMWare, etc. The KIWI architecture was designed as a two level system. The first stage, based on a valid **software package source**, creates a so called **unpacked image** according to the provided image description. The second stage creates from a required unpacked image an operating system image. The result of the second stage is called a **packed image** or short an image.

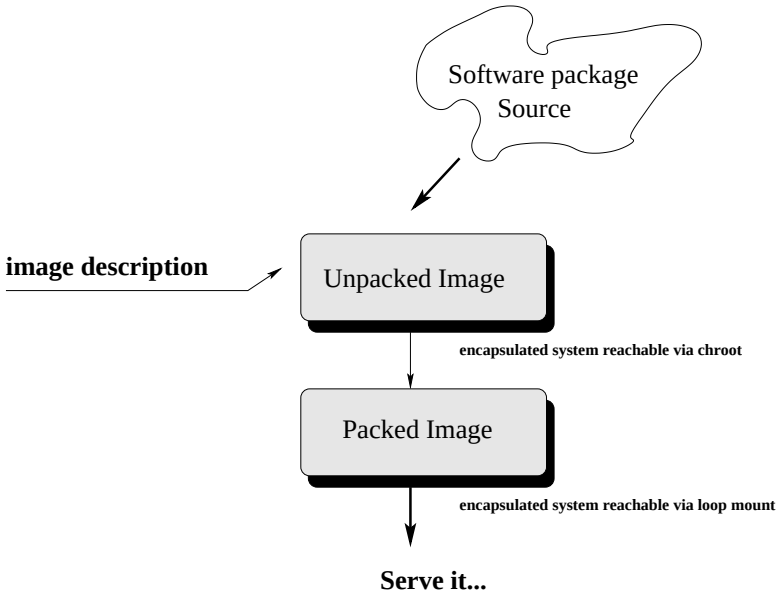

Figure 1.1: Image Serving Architecture

Because this document contains conceptual information about an image system, it is important to understand what an operating system image is all about. A normal installation process is starting from a given installation source and installs single pieces of software until the system is complete. During this process there may be manual user intervention required. However an operating system image represents an already completed *installation* encapsulated as a file and optionally includes the configuration for a specific task. Such an operating system starts working as soon as the image has been brought to a system storage device no matter if this is a volatile or non volatile storage. The process of creating an image takes place without user interaction. This means all requirements of the encapsulated system has to be fulfilled before the image is created. All of this information is stored in the **image description**.

2 Basic workflow

Contents

The creation of an image with KIWI is always divided into two basic steps. These are the **prepare** and the **create** step. the create step requires the prepare step to be exited successfully. Within this first prepare step kiwi builds of a new root tree or, in kiwi-speak, a new unpacked image. The building of a new root tree consists of the creation of the directory specified to hold it and the installation of the selected packages on it. The installation of software packages is driven by a packagemanager. KIWI supports the smart and zypper package managers. The prepare step executes the following major stages:

- **Root directory creation**
  To prevent accidental deletion of an existing root tree, kiwi will stop with an error message if this folder already exists, unless the option –force-new-root is used in which case the existing root will be deleted.
- **Package installation**
  First the selected package manager (smart by default) is instructed to use the repositories specified in the image description file. Then the packages specified in the 'bootstrap' section are installed. These packages are installed externally to the target root system (i.e. not chroot'ed) and establish the initial environment so the rest of the process may run chroot'ed. Essential packages in this section are filesystem and glibc-locale. In practice you only need to specify those two, since the rest of the packages will be pulled because of the dependency system. To save space in your image you could schedule a set of packages for deletion after the package installation phase is over by listing them in the 'delete' section.
- **User defined script config.sh**
  At the end of the preperation stage the optional script named config.sh is called. This script should be used to configure the system which means for example the activation of services. For a detailed description what functions are already available to configure the system please refer to the KIWI::config.sh manual page
- **Managing the new root tree**
  At this point you can make changes on your unpacked image so it fits your

purpose better. Bear in mind that changes at this point will be discarded and not repeated automatically if you rerun the 'prepare' phase unless you include them in your original config.xml file and/or config.sh script. Please also note that the image description has been copied into the new root below the directory **<new-root>/image**. Any subsequent create step will read the image description information from the new root tree and not from the original image description location. According to this if you need to change the image description data after the prepare call has finished you need to change it inside the new root tree as well as in your original description directory to prevent loosing the change when your root tree will be removed later for some reason.

After the prepare step has finished successfully a subsequent building of an image file or, in kiwi-speak, a new packed image follows. The building of an image requires a successfully prepared new root tree in the first place. Using this tree multiple image types can be created. So to speak it's possible to create a VMware image and a XEN image from the same prepared root tree. The create step executes the following major stages:

- **User defined script images.sh**
  At the beginning of the creation stage the optional script named images.sh is called. This script has no distinctive use case like config.sh but is most often used to remove packages which were pulled in by a dependency but are not really required for the later use of the operating system. For a detailed description what functions are already available to images.sh please refer to the KIWI::images.sh manual page
- **Create the requested image type**
  What image type(s) a kiwi image supports depends on what types has been setup in the main image description file config.xml. At least one type must be setup. The following picture shows what image types are currently supported by kiwi:

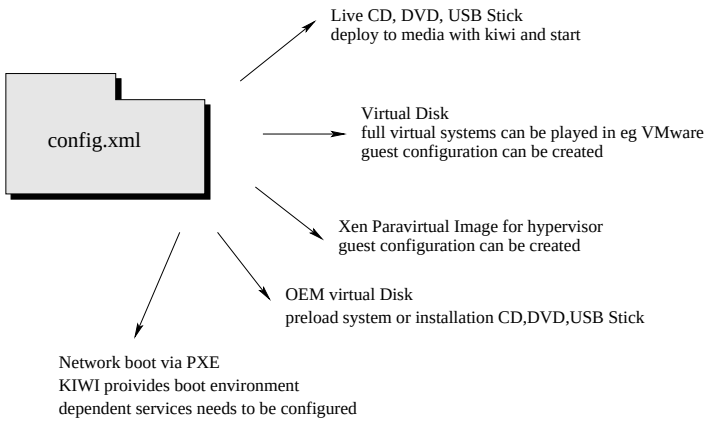

Figure 2.1: Image Types

Detailed information including a step by step guidance how to call kiwi and how to make use of the result image can be found in the image type specific sections

later in this document.

2.1 Boot process

Todays linux systems are using a special boot image to control the boot process. This boot image is called **initrd**. The linux kernel loads this initial ramdisk which is a compressed cpio archive into RAM and calls init or if present the program named linuxrc. The KIWI image system also takes care for the creation of this boot image. Each image type has it's own special boot code and shares the common parts in a set of module functions. The image descriptions for the boot images are provided by KIWI and thus the user has in almost all cases no need to take care for the boot image.

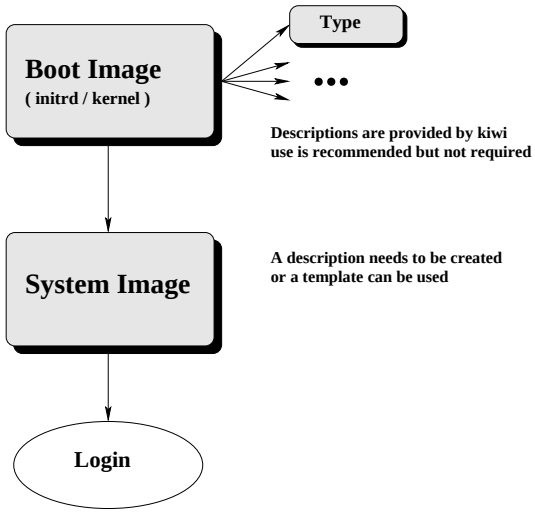

Figure 2.2: Boot process

Furthermore KIWI automatically creates this boot image along with the requested image type. It does that by calling itself in a prepare and create call. There is no difference in terms of the description of such a boot image compared to the system image description. The system image description is the one the user creates and this image represents the later operating system whereas the boot image only lives temporarly in RAM as long as the system image will be activated. The boot image descriptions are stores in /usr/share/kiwi/image/*boot and can be build in the same way as the system image. The boot image without a corresponding system image doesn't make sense though.

2.2 Boot parameters

When booting an image created by kiwi using one of the provided boot images there are some useful kernel parameters mainly meant for debugging purposes. Please note the following parameters are only useful if the kiwi initrd is used. In

case of any other initrd code written by yourself or simply because kiwi replaced itself with the distribution specfic mkinitrd tool the parameters might not have any effect.

- **kiwidebug=1**
  If the boot process encounters a fatal error the system normally reacts with a reboot after 120 secconds. This so called exception can be influenced by the kiwidebug parameter. If set to 1 the system will stop and provide the user with a shell prompt instead of a reboot. Within that shell some basic standard commands are available which could help to find the cause of the problem
- **kiwistderr=/dev/...**
  While the system boots kiwi writes messages to tty1 and tty3. The tty1 messages are highlevel information whereas the tty3 messages represents the shell debug output and any error messages from the commands called. With the kiwistderr parameter one can combine both message sets and specify where to write them to. It's very common to set /dev/console as possible alternative to the default logging behaviour

2.3 Common and Distribution specific code

KIWI has been developed to be usable for any Linux distribution. By design of a Linux distribution there are differences between each of them. With KIWI we provide on one hand the code which is common to all Linux distributions according to standards and on the other hand there is also code where we have to distinguish between the distribution type.

In case of such specific tasks which are almost all in the area of booting, KIWI provides a set of functions which all have to come with a distribution prefix. As this project uses SUSE Linux as base distribution all required distribution specific tasks has been implemented for SUSE and could be missing for other distributions. The existing implementation for SUSE turns out to be adapted to other distributions very easily though.

A look into the code therefore will show you functions which are prefixed by "suse" as well as scripts whose names starts with "suse-". At any time you see such a script or function you can be assured that this is something distribution specific and needs to be adapted if you plan to use KIWI with another distribution than SUSE. For example the boot workflow is controlled by a program called linuxrc which is in KIWI a script represented by suse-linuxrc. Another example would be the function called suseStripKernel which is able to remove everything but a specified list of kernel drivers from the SUSE kernel.

The prefixed implementation allows us to integrate all the distribution specific tasks into one project but this of course requires the help and knowledge of the people who are familar with their preferred linux distribution.

3 KIWI image description

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 alternatively ***.kiwi**. A good start for such a description can be found in the examples provided in /usr/share/doc/-packages/kiwi/examples.

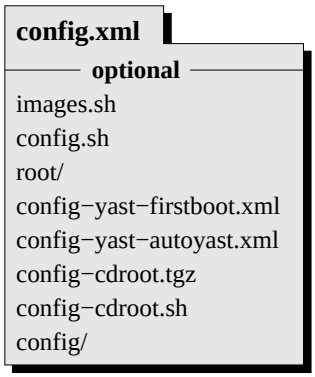

Figure 3.1: Image description directory

The following additional information is optional for the process of building an image but most often mandatory for the functionality of the later 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-firstboot.xml**
  Configuration file for the control of the yast2 firstboot service. Similar to the autoyast approach yast also provides a boot time service called firstboot. Unfortunately there is no GUI available to setup the firstboot but a good documentation in /usr/share/doc/packages/yast2-firstboot. Once you have created such a firstboot file in your image description directory KIWI will process on the file and setup your image as follows:

  1. KIWI enables the firstboot service
  2. While booting the image YaST is started in firstboot mode
  3. The firstboot service handles the instructions listed in the config-yast-firstboot.xml
  4. If the process finished successfully the environment is cleaned and firstboot won't be called at next reboot.

- **config-yast-autoyast.xml**
  Configuration file which has been created by autoyast. To be able to create such an autoyast profile you should first call:

  ```
  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 readme 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

3.1 config.xml

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 called the XML structure is validated by a RelaxNG based schema. For details on attributes and values please refer to the schema documentation file at /usr/share/doc/packages/ kiwi/kiwi.rng.html.

```
<image schemaversion="3.5" name="iname"
        displayname="text"
        inherit="path" 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 file names created by KIWI. Because we don't want spaces in file names the name attribute must not have any spaces in its name.

- The optional attribute displayname allows setup of the boot menu title for isolinux and grub. So you can have *suse-SLED-foo* as the image name but something like *my cool Image* as the boot display name.
- The optional attribute inherit allows to inherit the packages information from another image description.
- The optional attribute kiwirevision allows to specify a kiwi SVN revision number which is known to build a working image from this description. If the kiwi SVN revision is less than the specified value the process will exit. The currently used SVN revision can be queried by calling kiwi −−version
- The optional attribute id allows to set 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")**.

```
<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.

```
<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/drivers tag does not have a profiles attribute, it is assumed to be present for all profiles.

```
<preferences profiles="name">
    <version>1.1.2</version>
    <packagemanager>smart</packagemanager>
    <type .../>
</preferences>
```

The mandatory preferences section contains information about the supported image type(s), the used packagemanager, 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 use the **smart** packagemanager but it is also possible to use the

SUSE packagemanager called **zypper**.

Normally one preferences section is enough but it's possible to share data between different namespaces, so called profiles. According to this it's possible to have for example two preferences sections whereas one contains specific oem options and the other doesn't. This allows to add specific type based information while building the image.

At least one type must be set to be able to build an image from this description. Multiple type lines are allowed whereas you can specify with the boolean attribute named **primary** which should be the primary image if no type is requested on creation. The following list describes the possible types and their attributes:

- **usb**
  Use this type to create a USB stick system along with the attributes **filesystem** and **boot="usbboot/suse-*"** In addition to that type you can specify which bootloader to use. The optional attribute **bootloader** is used to setup either grub or syslinux as bootloader types. Setting the bootloader also works for vmx images but not for oem images. In case of an oem image only the distribution supported bootloader (grub) is allowed.
- **vmx**
  Use this type to create a virtual disk system along with the attributes **filesystem**, **boot="vmxboot/suse-*"** and optionally **format**. The format attribute spcifies one of the qemu supported virtualization formats, for example vmdk or qcow2. The optional attribute **vga** can be specified to configure the kernel framebuffer mode. Detailed information about the possible values can be found in /usr/src/linux/Documentation/fb/vesafb.txt. The vga attribute also works for the image types usb and oem.
- **oem**
  Use this type to create a preload virtual disk system along with the attributes **filesystem**, **boot="oemboot/suse-*"** and optionally **format**. If the format attribute is set to "iso" or "usb" KIWI will additionally create an installation media suitable for a CD/DVD or an USB stick. This installation media takes over the task of deploying the preload system onto the storage devices which it detects at boot time.
- **pxe**
  Use this type to create a network boot image along with the attributes **filesystem** and **boot="netboot/suse-*"** Additionally the attribute **compressed** specifies whether the image file should be compressed or not. This is not the filesystem compression just the image file compression most often used to transfer the compressed version over to the boot server.
- **iso**
  Use this type to create a live system on CD or DVD along with the attributes **boot="isoboot/suse-*"** and optionally **flags**. If no flags are specified the filesystem will not be compressed and no union filesystem is used. Allowed flags are:
  - **unified**: Compress filesystem with squashfs and mount the system read-

write with an aufs based overlay mount
  - **compressed**: Compress filesystem with squashfs and use a link list to mount the system read-write. An additional split section controls the read-write information
  - **dmsquash**: Creates an ext3 image file and puts that into a squashfs filesystem. On boot the root tree is mounted via a device mapper snapshot device to allow full write access over the complete tree.
  - **clic**: Creates a fuse based compressed read-only filesystem which allows write operations into a cow file

  If no flags or the flags ”compressed” are set an additional **split** section is recommended.

- **xen**
  Use this type to create a Xen enabled para-virtual gues image along with the attributes **filesystem** and **boot=”xenboot/suse-*”**
- **split**
  Use this type if you want to use one of the types **usb,vmx,oem or pxe** but as a split image. The split image support allows to create the image as split files wheras one part represents the read-write data and the other part represents the read-only data. Different filesystems can be assigned to each portion. According to this use this type together with the attributes **fsreadwrite**, **fsreadonly** and
  boot=”usb|vmx|oem|netboot/suse-*”
- **cpio**
  Use this type if your image is a boot image (intrd). Along with this type the optional attributes **bootprofile=”default”** and/or **bootkernel=”std”** exists. A boot image should group the different kernels it supports in profiles. It’s mandatory to have one profile named **std** which is used if no other bootkernel is specified. Profiles which leaves only a subset of drivers or packages should also be grouped in profiles. It’s mandatory to have one profile named **default** which is used if no other bootprofile is specified. Within the system image one can select the group by also specifying a bootprofile and/or bootkernel attribute. These information is passed automatically to kiwi when it builds the boot image.

All of the mentioned types can specify the **boot** attribute which tells kiwi to call itself to build the requested boot image (initrd). It is possible to tell kiwi to check for an already built boot image which is a so called **prebuilt boot image**. To activate searching for an appropriate prebuilt boot image the type section also provides the attribute **checkprebuilt=”true|false”**. If specified kiwi will search for a prebuilt boot image in a directory named */usr/share/kiwi/image/*boot/*-prebuilt*. Example: If the boot attribute was set to isoboot/suse-10.3 and checkprebuilt is set to true kiwi will search the prebuilt boot image in /usr/share/kiwi/image/isoboot/suse-10.3-prebuilt. The directory kiwi searches for the prebuilt boot images can also be specified at the commandline with the −−**prebuiltbootimage** parameter.

Within the preferences section there are the following optional attributes:

- **size**
  Specifies the size of the image with a numerical value in Megabytes or Gigabytes. Use the ”unit” attribute to assign the unit **M** for Megabytes or **G** for Gigabytes. KIWI extends the image size automatically if the specified value is too small. If the actual size is more than 100MB larger than the specified size, KIWI aborts with an error message. KIWI does not automatically reduce the image size if the specified value is too large, because the extra space might be needed to, for example, run custom scripts. If no size is specified, KIWI uses the required size plus approximately 30% free space. The optional ”additive” attribute can be set to tell kiwi to use the required size for the image plus the given size as additional free space. The ”additive” attribute is a bool attribute and can be set to either true or false.
- **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 locale to use, which defines the contents of the RC_LANG system environment variable in /etc/sysconfig/language
- **boot-theme**
  Specifies the name of the gfxboot and bootsplash 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
- **defaultbaseroot**
  Used if the option –base-root is not specified when calling KIWI. It’s possible to prepare and create an image using a predefined non empty root directory as base information. This could speedup the build process a lot if the base root path already contains most of the image data.

```
<users group="users" id="number">
    <user pwd="..." home="dir"
      name="user" id="number"/>
    ...
</users> ...
```

The optional users element specifies the users to be added to the image. The group attribute specifies the group the users belong to. If this group does not exist, it is created. A user element must be specified for each group. The user child element specifies the users belonging to that group, and the name, pwd and home attributes specifies the username, crypted password, and path to the home directory. The password can be created by the kiwi −−createpassword tool.

```
<drivers type="type" 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. The type attribute specifies one of the following driver types:

- **drivers**
  Each file is specified relative to the /lib/modules/<Version>/kernel directory.
- **netdrivers**
  Each file is specified relative to the /lib/modules/<Version>/kernel/drivers directory.
- **scsidrivers**
  Each file is specified relative to the /lib/modules/<Version>/kernel/drivers
- **usbdrivers**
  Each file is specified relative to the /lib/modules/<Version>/kernel/drivers directory.

According to the driver type the specified files are searched in the corresponding directory. The information about the drivernames 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.

```
<repository type="type"
      status="replaceable"
      alias="name"
      priority="number">
            <source path="URL"/>
</repository>
...
```

The mandatory repository section specifies the source URL and type used by the package manager. The type attribute specifies the repository type which must be supported by the package manager. At the moment KIWI supports the package managers smart and zypper whereas smart has support for more repository types compared to zypper. Therefore the possible values for the type attribute has beend copied from smart. The following table shows the possible repo types:

type	smart	zypper
apt-deb	yes	no
apt-rpm	yes	no
deb-dir	yes	no
mirrors	yes	no
red-carpet	yes	yes
rpm-dir	yes	yes
rpm-md	yes	yes
slack-site	yes	no
up2date-mirrors	yes	no
urpmi	yes	no
yast2	yes	yes

Within the repository section there are the following optional attributes:

- **status=”replaceable”**
  This attribute makes only sense for boot image descriptions. It indicates that the repository is allowed to become replaced by the repositories defined in the system image descriptions. Because kiwi automatically builds the boot image if required it should create that image from the same repositories which are used to build the system image to make sure both fit together. Therefore it is required to allow the repository to become overwritten which is indicated by the status attribute.
- **alias=”name”**
  Specifies an alternative name used to identify the source channel. 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
- **priority=”number”**

Specifies 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. At the moment this only works for the smart package manager.

The source child element contains the path attribute, which specifies the location (URL) of the repository. The path specification can be any of the following, and can include the %arch macro which is expanded to the architecture of the image building host.

- **this://<path>**
  A relative path name, which is relative to the image description directory being referenced.

- **iso://<path/to/isofile**
  A path to a local .iso file which is then loopback mounted and used as a local path based repository. Alternatively one can do the loop mount himself and point a standard local path to the mounted directory

- **http://<url>**
  A http protocol based network location

- **https://<url>**
  A https protocol based network location

- **ftp://<url>**
  A ftp protocol based network location

- **opensuse://<Project-Name>**
  A special http based network location which is created from the given openSUSE buildservice project name. The result is pointing to an rpm-md repository on the openSUSE buildservice. For example:
  path=”opensuse://openSUSE:10.3/standard”

- **file:///local/path**
  A local path which should be an absolute path description. The file:// prefix is optional and could also be omitted.

- **obs://$dir1/$dir2**
  A special buildservice path whereas $dir1 and $dir2 represents the buildservice project location. If this type is used as part of a boot attribute kiwi evaluates it to this://images/$dir1/$dir2 and if used as part of a repository source path attribute it evaluates to this://repos/$dir1/$dir2

Multiple repository sections are allowed and combined by the used package manager. By default the package manager will always use the latest packages available.

```
<packages type="type" profiles="name"
          patternType="type"
          patternPackageType="type"
    <package name="name" arch="arch"/>
    <package name="name" replaces="name"/>
    <package name="name"
          bootinclude="yes" bootdelete="yes"/>
    <package .../>
    <opensusePattern name="name"/>
    <opensusePattern .../>
    <opensuseProduct name="name"/>
    <opensuseProduct .../>
    <ignore name="name"/>
    <ignore .../>
</packages>
```

The mandatory packages element specifies the list of packages and pattern names to be used with the image. There are five different types of package sets or patterns, specified with the type attribute:

- **image**
  Image packages, list of packages used to finish the image installation. All packages which make up the image are listed here
- **bootstrap**
  Bootstrap packages, list of packages used to start creating a new operating system root tree. Basic components which are required to chroot into that system, such as glibc, are listed here.
- **delete**
  Delete packages, list of packages stored for later deletion. The package names are available in the $delete environment variable of the /.profile file created by KIWI. The baseGetPackagesForDeletion() function returns the contents of this environment variable, and can be used to delete the packages while ignoring requirements or dependencies. According to this a config.sh or images.sh script needs to be provided such as the following code snippet shows:

  ```
  rpm -e --nodeps --noscripts \
     $(rpm -q ‘baseGetPackagesForDeletion‘ | grep -v "is not installed")
  ```

- **xen**
  Xen required packages, list of packages used when the image needs support for Xen-based virtualization.
- **vmware**

VMware required packages, list of packages used when the image needs support for VMware- or generic based full virtualization.

Using patterns

Using a pattern name enhances the package list with a number of additional packages belonging to this pattern. Support for patterns is SUSE-specific, and available with openSUSE 10.1 or later. The optional patternType and patternPackageType attributes specify which pattern references or packages should be used in a given pattern. The values of these attributes are only evaluated if the KIWI pattern solver is used. If the new (up to SUSE 11.0) satsolver pattern solver is used these values are ignored because the satsolver can't handle that at the moment. Allowed values for the pattern* attributes are:

- **onlyRequired**
  Incorporates only patterns and packages that are required by the given pattern
- **plusSuggested**
  Incorporates patterns and packages that are required and suggested by the given pattern
- **plusRecommended**
  Incorporates patterns and packages that are required and recommended by the given pattern.

By default, only required patterns and packages are used. The result list of packages is solved into a clean conflict free list of packages by the package manager. This for example means that including a suggested package may include required and recommended packages as well according to the dependencies. If a pattern contains unwanted packages, you can use the ignore element to specify an ignore list, with the name attribute containing the package name. Please note that you can't ignore a package if it is required by a package dependency of another package in your list. The packagemanager will automatically pull in the package even if you have ignored it.

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.

Image type specific packages

If a package is only required for a specific type of image and replaces another package you can use the **replaces** attribute to tell kiwi to install the package by

replacing another one. For example you can specify the kernel package in the type=image section as

```
1 <package name="kernel-default" replaces="kernel-xen"/>
```

and in the type=xen section as

```
1 <package name="kernel-xen" replaces="kernel-default"/>
```

The result is the xen kernel if you request a xen image and the default kernel in any other case.

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. 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

```
<vmwareconfig arch="arch" memory="MB"
        HWversion="number"
        guestOS="suse|sles"
        usb="true|false"/>
    <vmwarenic driver="name"
        interface="number" mode="mode"/>
    <vmwaredisk controller="ide|scsi"
        id="number"/>
    <vmwarecdrom controller="ide|scsi"
        id="number"/>
</vmwareconfig>
```

The optional vmwareconfig section is used if the image description includes a packages section of type **vmware**. In this case kiwi is able to create the guest configuration file required to run the image within VMware. The guest configuration file can also be created by the VMware toolkit itself but with the pre-created guest configuration created by kiwi it is possible to provide an all in one bundle ready to run in VMware. The following general information can be provided to create the VMware (.vmx) configuration file:

- **arch**
  The virtualized architecture. Can be one of ix86 or x86_64 Bydefault ix86 is used.
- **memory**
  The mandatory memory attribute specified how much memory in MB should be allocated for the virtual machine
- **HWversion**
  The VMware hardware version number. By default version 3 is used
- **guestOS**
  The guestOS identifier. By default suse is used on ix86 and suse-64 for x86_64. At the moment only the suse and sles guestOS types are supported
- **usb**
  The bool value **usb** specifies whether the guest machine should provide a virtual USB controller or not.

The following information can be provided to setup the VMware virtual main storage device and CD/DVD drive connection.

- **controller**
  The mandatory controller attribute can be either ide or scsi disk
- **id**
  The mandatory id attribute specifies the disk id. If only one disk is set the id value should be set to 0

The following information can be provided to setup the VMware virtual network interface

- **driver**
  The mandatory driver to use for the virtual network card. Possible values are vlance, e1000 or vmxnet. vmxnet requires the vmware tools to be part of the image
- **interface**
  The mandatory network 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.

```
<xenconfig memory="MB"
    <xendisk device="/dev/..."/>
    <xenbridge name="eth0" mac="addr"/>
</vmwareconfig>
```

The optional xenconfig section is used if the image description includes a packages section of type **xen**. In this case kiwi is able to create the guest configuration file

required to run the image within Xen. According to this it's possible to provide an all in one bundle ready to run in Xen. The following general information can be provided to create the Xen (.xenconfig) configuration file:

- **memory**
  The mandatory memory attribute specified how much memory in MB should be allocated for the para virtual machine

The following information can be provided to setup the Xen para virtual main storage device as part of a **xendisk** section

- **device**
  The mandatory device which should appear in the para virtual instance

The default Xen configuration uses bridging within domain 0 to allow all domains to appear on the network as individual hosts. In order to create the bridge which can be used by the Xen virtual network interface(s) the script "/etc/xen/scripts/network-bridge start" can be called to create a bridge as shown in the following picture:

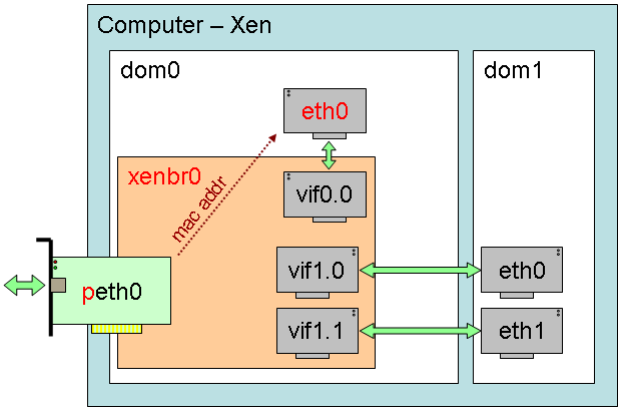

Figure 3.2: Illustration on network-bridge and vif-bridge

Additional information on how to setup networking with Xen can be found here: http://wiki.xensource.com/xenwiki/XenNetworking The following information can be provided to setup the Xen network bridge as part of one ore more **xen-bridge** section(s).

- **name**
  The name of the network interface which is the bridge between the physical device (peth) and the virtual device(s) (vif).

- **mac**
  The optional mac address value for the virtual interface inside the DOM(x).

```
<deploy server="IP" blocksize="4096">
    <timeout>seconds</timeout>
    <commandline>kernel-options</commandline>
    <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="aufs|unionfs"/>
    <configuration source="/KIWI/../file" dest="/../file"
                   arch="..."/>
    <configuration .../>
</deploy>
```

The optional deploy section is only useful if you build the **pxe** image type. For this type an additional network boot infrastructure needs to be set up. To ease the process of setting up such a boot server kiwi provides a package called kiwi-pxeboot. This package sets up the basic pxe boot environment like kiwi expects it. The package will setup a directory structure in /srv/tftpboot. The result files of the kiwi image build process needs to be copied to that location. A detailed explanation of what file needs to be copied and where is provided in the PXE Image chapter later in this document. Among the image files itself it is required to provide an information how KIWI should handle the machine it should be install with the created image. Information like which image should be used or how to partition the machine needs to be provided in a file called config.<MAC-Address> below the directory /srv/tftpboot/KIWI. The reason for the deploy section is to allow KIWI to create that file according to the information provided in the image description.

- The server and blocksize attributes specify the TFTP server which controls the download of image files. KIWI also supports other protocols than tftp but in order to do that the variables **kiwiserver** and **kiwiservertype** must be set as kernel parameter when the client boots.
- The optional timeout section specifies the grub timeout in seconds which is used when the KIWI initrd configures and installs the grub boot loader on the machine after the first deployment to allow standalone boot.
- The optional commandline section specifies the kernel options which should be passed to the kernel by the grub bootloader. the KIWI initrd includes this kernel options when installing the grub for standalone boot

- The optional kernel and initrd sections specifies the KIWI kernel and initrd files on the boot server. In case of a special boot method which is not supported by the distribution standard mkinitrd the KIWI initrd needs to stay on the system and needs to be used for local boot as well. So if your system image makes use of the split type or your deploy section includes any union information the kernel and initrd sections must be provided.
- The partitions section is required if you want to install the system image on a disk or any other permanent storage device. Each partition is specified by one partition subtag which defines the type (see sfdisk -list-type), partition number, size, optional mountpoint, and optional information on if this partition is the system image target partition. With the KIWI netboot boot image, the first partition is always the swap partition, while the second partition is used, by default, for the system image. With the optional target flag, you can specify a partition other than the second partition to install the system image on. If size is set to ”image”, KIWI calculates the required size for this partition in order to have enough space for the later image.
- The optional union section is used if the system image is based on a read-only filesystem such as squashfs and should be mounted read-write by using an overlay filesystem like aufs or by a device. mapper setup with the dmsquash type. In this case, KIWI creates an additional write partition, then combines both partitions with the given overlay filesystem or device map. Currently, there are two such filesystems: unionfs and aufs (aufs is the preferred file system). The partition that holds the read-only system image must be set as the ro attribute value, and the partition that serves as the write partition must be set the rw attribute value.
- The optional configuration section can be used to integrate a network client’s configuration files which are stored remotely on the server. The source attribute specifies the path on the server used by a TFTP client program to download the file, and the dest attribute specifies the target relative to the root (/) of the network client. Each file is specified by one configuration section and can be bound to a specific set of architectures separated by comma.

```
<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>
```

The optional split section is used if your image type is split or iso combined with the attribute compressed. The split section controls which files of your splitted image should be writable and whether they are persistantly writable or only temporarly. In case of an iso image all data specified can only be temporarly writable by design of the live system image type.

The split section distinguishes between directory and files. The information of ”/etc” would make /etc a writable directory however none of the files *within* /etc are affected. They remain symbolic links to the real files in the read-only area. The main advantage to putting just a directory in the read-write area is that any new files created there are stored on the disk instead of tmpfs. If all the files in /etc should also be part of the read-write area and according to this put a complete directory including all its files into the read-write area two lines are required as shown above.

4 Creating Appliances with KIWI

Contents

4.1 History

Traditionally, many computing functions were written as software applications running on top of a general-purpose operating system. The consumer (whether home computer user or the IT department of a company) bought a computer, installed the operating system or configured a pre-installed operating system, and then installed one or more applications on top of the operating system. An e-mail server was just an e-mail application running on top of Linux, Unix, Microsoft Windows, or some other operating system, on a computer that was not designed specifically for that application.

4.2 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. First you should choose an appropriate image description template from the provided kiwi examples and add/adapt repository and/or package names according to the distribution you want to build an image for
2. Allow the image to create an in-place git repository to allow tracking of non binary changes. This is done by adding the following into your config.sh script

```
baseSetupPlainTextGITRepository
```

3. Prepare the preliminary version of your new appliance by calling **kiwi –prepare ...**. Refer to chapter 9 (USB image - Live-Stick System) for details.
4. 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. According to this make sure you have a usb type in your config.xml

```
<type filesystem="ext3"
      boot="usbboot/suse-...">usb</type>
```

5. Create the preliminary live stick image of your new appliance by calling **kiwi –create ...**. 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. Use the **kiwi –bootstick ...** call to deploy the image on the stick. Refer to chapter 9 (USB image - Live-Stick System) for details.
6. Plug in the stick on your test machine and boot it
7. 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.
8. 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

9. 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.

5 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:

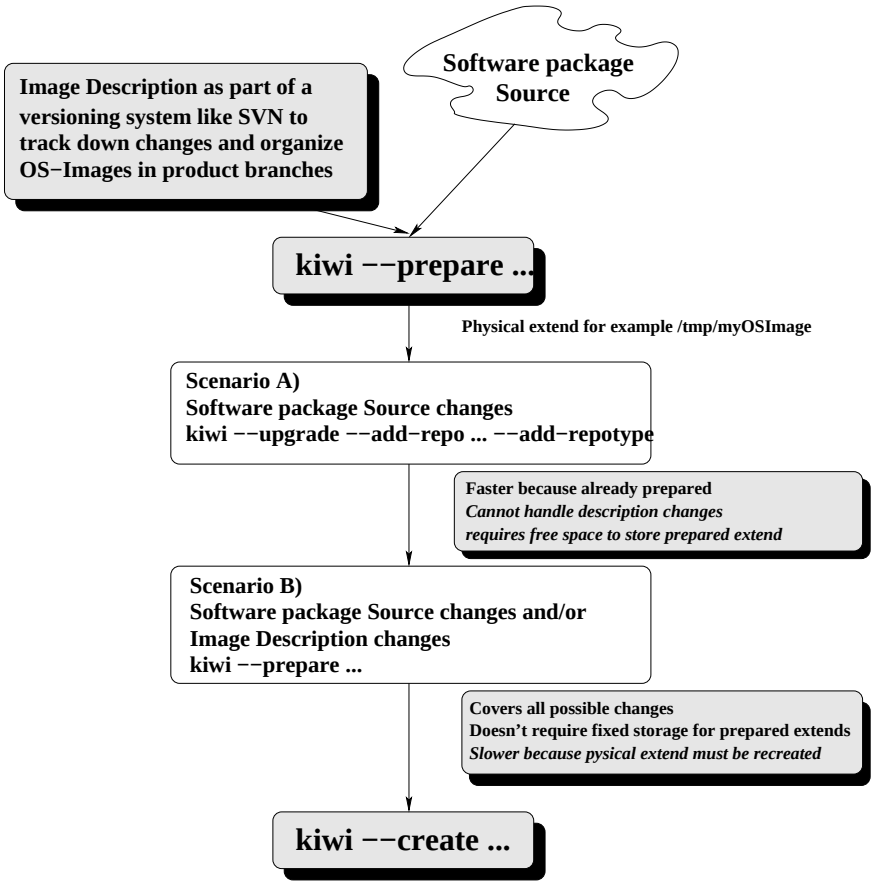

Figure 5.1: Image maintenance scenarios

The picture above 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 principal it's possible to do all those changes manually within the physical extend 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 **dont't modify physical extends 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 subversion. 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 physical extend could be removed.

6 System to image migration

Contents

KIWI provides an experimental module which allows you to turn your running system into an image description. This migration 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
- You can't rely on the result to be a 100% ready to use copy of your system. This means some manual postprocessing might be necessary

When calling KIWI's migrate mode it will try to find the base version of your operating system and assigns a predefined repository to recreate the data which exists in terms of packages. The code inspect your system and creates 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 your configurations your user data and all other stuff. KIWI collects all this information and would copy it as overlay files as part of the image description. The process will skip all remote mounted filesystems and concentrate only on local filesystems.

6.1 Create a migration report first

When running the migration for the first time I recommend to create a report first:

```
kiwi --migrate mySys --destdir /tmp/migrated \
     --report
```

After that call you should walk through the following check list

- check the contents of the config.sh script. The migration added at least the services your system runs and adds them to the configuration script. Check this service list

- check the report file contents. All data which doesn't belong to a package are listed there. You should make sure whether you need them all or if you could exclude some of them. As a recommendation, you should have as little as possible overlay files.
- check the created config.xml image description file. You should at least make sure if the repository is correct and if you need more repositories for packages which are not part of the base repository for example
- check the kiwi output on the console. Each package which it can't find in the base repository of the distribution is skipped and not added as package in your package list. So for example if you use the nvidia binary driver package from an extra repo you need to add the repo and the package later in your config.xml file

6.2 Migrate my system...

After the check list you will have a first impression of your system. What data is there what's not part of packages what doesn't need to be part of the image description and so on. You can exclude the directories which you don't need according to the report file with the --exclude parameter. Now you can call migrate again and let it copy the overlay files too:

```
rm -rf /tmp/migrated
kiwi --migrate mySys --destdir /tmp/migrated \
     --exclude directory --exclude ... \
     --add-repo URI --add-repotype type ...
```

6.3 Turn my system 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 really make sure whether you need them all or not. Now let's try to create a clone image from the description. The most appropriate image type to do this is the virtual disk image (vmx)

```
kiwi -p /tmp/migrated --root /tmp/mySys
kiwi --create /tmp/mySys -d /tmp/myResult \
     --type vmx
```

If everything worked well you can test the created virtual disk 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.

7 Installation Source

Contents

Before you start to use any of the examples provided in the following chapters your build system has to have a valid installation source for the distribution you are about to create an image for. By default all examples will connect to the network to find the installation source. It depends on your network bandwidth how fast an image creation process is and in almost all cases it is better to prepare a local installation source first.

7.1 Adapt the example's config.xml

If you can make sure you have a local installation source it's important to change the path attribute inside of the <repository> element of the appropriate example to point to your local source directory. A typically default repository element looks like the following:

```
<repository type="yast2">
   <!--<source path="/image/CDs/full-11.0-i386"/>-->
   <source path="opensuse://openSUSE:11.0/standard/"/>
</repository>
```

7.2 Create a local installation source

The following describes how to create a local SUSE installation source which is stored below the path: */images/CDs* If you are using the local path as described in this docuement you only need to flip the given path information inside of the example config.xml file.

1. find your SUSE standard installation CDs or the DVD and make them available to the build system. Most linux systems auto-mount a previosly inserted media automatically. If this is the case you simply can change the directory to the auto mounted path below /media. If your system doesn't mount the device automatically you can do this with the following command:

```
mount -o loop /dev/<drive-device-name> /mnt
```

2. You don't have a DVD but a CD set ? No problem all you need to do is copy the contents of **all** CDs into one directory. It's absolutly important that you first start with the **last** CD and copy the first CD at last. In case of CDs you should have a bundly of 4 CDs. Copy them in the order 4 3 2 1
3. Once you have access to the media copy the contents of the CDs / DVD to your hard drive. You need at least 4GB free space available. The following is intended to create a SUSE 11.0 installation source:

```
mkdir -p /image/CDs/full-11.0-i386/
cp -a /mnt/* /image/CDs/full-11.0-i386/
```

   Remember if you have a CD set start with number 4 first and after that unplugg the CD and insert the next one to repeat the copy command until all CDs are copied into to /image

8 ISO image - Live Systems

Contents

A live system image is an operating System on CD or DVD. In principal 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. All write actions takes place in RAM space and therefore all changes will be lost as soon as the computer shuts down.

8.1 Building the suse-live-iso example

The latest example provided with kiwi is based on openSUSE 11.0 and includes the base + kde patterns.

```
cd /usr/share/doc/packages/kiwi/examples
cd suse-11.0
kiwi --prepare ./suse-live-iso \
     --root /tmp/myiso --add-profile KDE
```

```
kiwi --create /tmp/myiso \
     --type iso -d /tmp/myiso-result
```

8.2 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 virtualisation system to test the image directly. Testing an iso can be done with any full virtual system for example:

```
cd /tmp/myiso-result
qemu -cdrom \
  ./suse-11.0-live-iso.i686-2.5.1.iso -m 256
```

8.3 Flavours

KIWI supports different filesystems and boot methods along with the ISO image type. The provided example by default uses a squashfs compressed root filesystem. By design of this filesystem it is not possible to write data on it. To be able to write on the filesysetem another filesystem called aufs is used. aufs is an overlay filesystem which allows to combine two different filesystems into one. In case of a live system aufs is used to combine the squashfs compressed read only root tree with a tmpfs RAM filesystem. The result is a full writable root tree whereas all written data lives in RAM and is therefore not persistent. squashfs and/or aufs does not exist on all versions of SUSE and therefore the flags attribute in config.xml exists to be able to have the following alternative solutions:

- **flags="unified"**
  Compressed and unified root tree as explained above
- **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="dmsquash"**
  Creates an ext3 image file and puts that into a squashfs filesystem. On boot the root tree is mounted via a device mapper snapshot device to allow full write access over the complete tree. No other overlay filesystem is required.
- **flags="clic"**
  Creates a fuse based clicfs image and allows write operations into a cow file. In case of an ISO the write happens into a ramdisk.
- **flags not set**
  If no flags attribute is set no compressed filesystem and 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.

8.3.1 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 because the PAM authentification 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>
   <temporary>
      <!-- allow read/write access to: -->
      <file name="/var"/>
      <file name="/var/*"/>
   </temporary>
</split>
```

9 USB image - Live-Stick System

Contents

A live USB stick image is a system on USB stick which allows you to boot and run from this device without using any other storage device of the computer. It is urgently required that the BIOS of the system which you plug the stick in supports booting from USB stick. Almost all new BIOS systems support that. The USB stick serves as OS system disk in this case and you can read and write data onto it.

9.1 Building the suse-live-stick example

The latest example provided with kiwi is based on openSUSE 11.0 and makes use of the default plus x11 pattern. The operating system is stored on a standard ext3 filesystem.

```
cd /usr/share/doc/packages/kiwi/examples
cd suse-11.0
kiwi --prepare ./suse-live-stick \
     --root /tmp/mystick
```

There are two possible image types which allows you to drive the stick. Both are added into the config.xml of this example image description. If you already have access to the stick you want to run the image on the first approach should be preferred over the second one.

- The first image type named ”usb” creates all required images for booting the OS but requires you to plug in the stick and let kiwi deploy the data onto this stick.

```
kiwi --create /tmp/mystick --type usb \
     -d /tmp/mystick-result
```

- The second image type named ”oem” allows you to create a virtual disk which represents a virtual disk geometry including all partitions and boot information in one file. You simply can ”dd” this file on the stick.

```
kiwi --create /tmp/mystick --type oem \
     -d /tmp/mystick-result
```

9.2 Using the image

To make use of the created images they need to be deployed on the USB stick. For the first image type (usb) you need kiwi itself to be able to deploy the image on the stick. The reason for this is that the usb image type has created the boot and the system image but there is no disk geometry or partition table available. kiwi creates a new partition table on the stick and imports the created images as follows:

```
kiwi --bootstick \
   /tmp/mystick-result/\
   initrd-usbboot-suse-11.0.i686-2.1.1.splash.gz \
   --bootstick-system \
   /tmp/mystick-result/\
   suse-11.0-live-stick.i686-1.1.2
```

In case of the second image type (oem) you only need a tool which allows you to dump data onto a device. On Linux the most popular tool to do this is the **dd** command. The oem image is represented by the file with the .raw extension. As said this is a virtual disk which already includes partition information. But this partition information does not match the real USB stick geometry which means the kiwi boot image (oemboot) has to adapt the disk geometry on first boot. To deploy the image on the stick call:

```
dd if=/tmp/mystick-result/\
   suse-11.0-live-stick.i686-1.1.2.raw \
   of=/dev/<stick-device> bs=32k
```

Testing of the live stick can be done with a test machine which boots from USB or with a virtualisation system. If you test with a virtualisation system for example qemu you should be aware that the USB stick looks like a normal disk to the system. The kiwi boot process searches for the USB stick to be able to mount the correct storage device but in a virtual environment the disk doesn't appear as a USB stick. So if your virtualisation solution doesn't provide a virtual BIOS which allows booting from USB stick you should test the stick on real hardware

9.3 Flavours

USB sticks weren't designed to serve as storage devices for operating systems. By design of these nice little gadgets their storage capacity is limited to only a few G-bytes. According to this KIWI supports compressed filesystems with USB sticks too:

- **filesystem="squashfs"**
  This will compress the image using the squashfs filesystem. The boot process will automatically use aufs as overlay filesystem to mount the complete tree read-write. For the write part an additional ext2 partition will be created on the stick. The support for this compression layer requires squashfs and aufs to be present in the distribution KIWI has used to build the image

- **filesystem="dmsquash"**
  Creates an ext3 image file and puts that into a squashfs filesystem. On boot the root tree is mounted via a device mapper snapshot device to allow full write access over the complete tree.

- **filesystem="clicfs"**
  Creates a fuse based clicfs image and allows write operations into a cow file.

9.3.1 Split stick

If there is no overlay filesystem available it is also possible to define a split section in config.xml and use the split support to split the image into a compressed read-only and a read-write portion. To create a split stick the types needs to be adapted as follows:

- **type setup for split usb type:**

```
<type fsreadwrite="ext3" fsreadonly="squashfs"
      boot="usbboot/suse-11.0">split</type>
```

- **type setup for split oem type:**

```
<type fsreadwrite="ext3" fsreadonly="squashfs"
      boot="oemboot/suse-11.0">split</type>
```

For both types a split section is required which defines the read-write data. A good starting point is to set /var, /home and /etc as writable data.

```
<split>
   <persistent>
      <!-- allow read/write access to: -->
      <file name="/var"/>
      <file name="/var/*"/>
      <file name="/etc"/>
      <file name="/etc/*"/>
      <file name="/home"/>
      <file name="/home/*"/>
   </persistent>
</split>
```

10 VMX image - Virtual Disks

Contents

A VMX image is a virtual disk image for use in full virtualisation systems like QEMU or VMware. The image represents a file which includes partition data and bootloader information. The size of this virtual disk can be influenced by either the <size> element in your config.xml file or by the parameter −−bootvm-disksize

10.1 Building the suse-vm-guest example

The latest example provided with kiwi is based on openSUSE 11.0 and makes use of the base pattern. The operating system is stored on a standard ext3 filesystem.

```
cd /usr/share/doc/packages/kiwi/examples
cd suse-11.0
kiwi --prepare ./suse-vm-guest \
     --root /tmp/myvm
```

```
kiwi --create /tmp/myvm \
     --type vmx -d /tmp/myvm-result
```

10.2 Using the image

The generated virtual disk image serves as the harddisk of the selected virtualisation system. The setup of the virtual hard disk differes from the variety of the virtualisation systems. A very simply to use system is the QEMU virtualisation software. To run your image in qemu call:

```
cd /tmp/myvm-result
qemu suse-11.0-vm-guest.i686-1.1.2.raw -m 256
```

10.3 Flavours

Because there are many virtualisation systems available there are also many virtual disk formats. The .raw format KIWI always creates has the same structure as you can find on a real hard disk. For virtualisation software it makes sense to have specific formats to increase the I/O performance when reading or writing data onto the disk from within the virtual system. If you want to tell KIWI to create an additional disk format just extend the type information of the config.xml file by a format attribute.

```
<type ... format=''name''>vmx</type>
```

The following table shows a list of supported virtual disk formats

Name	Description
vvfat	Disk format DOS FAT32
vpc	Virtual PC read only disk
bochs	Disk format for Bochs emulator
dmg	Disk format for Mac OS X
cloop	Compressed loop
vmdk	Disk format for VMware
ovf	Open Virtual Format requires VMwares ovftool
qcow2	QEMU virtual disk format
qcow	QEMU virtual disk format
cow	QEMU virtual disk format

10.3.1 VMware support

VMware is a very popular and fast virtualisation platform which is the reason why KIWI has special support for it. VMware requires a so called guest configuration which includes information about what hardware should make up the guest and how much ressources should be provided to the guest. With KIWI you can provide the information required to create a guest configuration as part of the config.xml file. Additionally you can group special packages which you may only need in this virtual environment.

```
<packages type="vmware">
   <!-- packages you need in VMware only -->
</packages>
<vmwareconfig memory="512">
   <vmwaredisk controller="ide" id="0"/>
</vmwareconfig>
```

If this information is present KIWI will create a VMware guest configuration with 512 MB of RAM and an IDE disk controller interface. Additional information to setup the VMware guest machine properties are explained in the **vmwareconfig** section. The written guest configuration file can be easily loaded and changed by the native graphics user provided with VMware. The KIWI VMware guest configuration is stored in the file:

```
/tmp/myvm-result/suse-11.0-vm-guest.i686-1.1.2.vmx
```

Together with the **format="vmdk"** attribute KIWI creates a VMware based image (.vmdk file) and the required VMware guest configuration (.vmx)

You can also create an image for the Xen virtualization framework. To do this, you simply need to specify the 'xen' boot profile in your config.xml. Like VMware, Xen has a configuration file as well. Refer to chapter 13 (Xen image) for details.

11 PXE image - Thin Clients

Contents

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 seperately and needs to be copied at specific locations of a network boot server. 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.

11.1 Setting up the required services

Before you start to build pxe images with kiwi you should have setup the boot server. The boot server requires the services **atftp** and **DHCP** to run

11.1.1 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 and set/modify the following variables:
   - ATFTPD_OPTIONS=”−−daemon −−no-multicast”
   - ATFTPD_DIRECTORY=”/srv/tftpboot”
3. run atftpd by calling the command: **rcatftpd start**

11.1.2 DHCP server

In contrast to the atftp server setup the following DHCP server setup can only serve as an example. Please note that according to 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 you should 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**

11.2 Building the suse-pxe-client example

The example provided with kiwi is based on openSUSE 11.0 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 unified (aufs) based system.

```
cd /usr/share/doc/packages/kiwi/examples
cd suse-11.0
kiwi --prepare ./suse-pxe-client \
     --root /tmp/mypxe
```

```
kiwi --create /tmp/mypxe --type pxe \
     -d /tmp/mypxe-result
```

11.3 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-11.0.i686-2.1.1.splash.gz \
      /srv/tftpboot/boot/initrd
   cp initrd-netboot-suse-11.0.i686-2.1.1.kernel \
      /srv/tftpboot/boot/linux
   ```

3. Copy of the system image and md5 sum

   ```
   cp suse-11.0-pxe-client.i686-1.2.8 \
      /srv/tftpboot/image
   cp suse-11.0-pxe-client.i686-1.2.8.md5 \
      /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 globaly the KIWI generated file can be copied as config.default

   ```
   cp suse-11.0-pxe-client.i686-1.2.8.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 the file /srv/tftpboot/pxelinux.cfg/default should provide the following information:

```
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

11.4 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.

11.4.1 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*.

11.4.2 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 immediately 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-11.0-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,..., src;dest;srvip;bsize
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).

- **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. Additionally all size specifications supported by the sfdisk program are allowed as well. The type number specifies the ID of the partition. Valid ID's are listed via the *sfdisk –list-types* command. 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.
- **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**.
- **RELOAD_IMAGE**
  If set to a non-empty string, forces the configured image to be loaded from the server even if the image on the disk is up-to-date. Used mainly for debugging purposes, this option only makes sense on diskful systems.
- **RELOAD_CONFIG**
  If set to an non-empty string, forces all config files to be loaded from the server. Used mainly for debugging purposes, this option only makes sense on diskful systems.
- **COMBINED_IMAGE**
  If set to an 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 tftpserver 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 and usbboot images there is the possibility to use unionfs or aufs as container filesystem in combination with a compressed system image. The recommended compressed filesystem type for the system image is

squashfs. In case of a usb-stick system the usbboot image will automatically setup the unionfs/aufs filesystem. In case of a PXE network image the netboot image requires a config.<MAC> setup like the following example shows: **UNIONFS_CONFIG=/dev/sda2,/dev/sda3,aufs**. 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 aufs 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:

```
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,aufs
DISK=/dev/sda
```

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 port. 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-Port-Number;/dev/NBD-Device;\
          NBD-Swap-Port-Number;/dev/NBD-Swap-Device
  ```

  The NBD-Device, NBD-Swap-Port-Number, and NBD-Swap-Device variables are optional. If they are not set, the default values are used (/dev/nb0 for the NBD-Device, port number 9210 for the NBD-Swap-Port-Number, and /dev/nb1 for the NBD-Swap-Device). The swap space over the network using a

network block device is only established if the client has less than 48 MB of RAM.

- **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 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/system:/aoetools
  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.

- **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.

11.4.3 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.

11.4.4 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"
- sample config.<MAC>

```
IMAGE=/dev/ram1;suse-11.0-pxe-client.i686;\
      1.2.8;192.168.100.2;4096
```

11.4.5 union image

As used in the suse-pxe-client example it is possible to make use of the aufs or unionfs overlay filesystems 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 make sure your config.xml's filesystem attribute contains either squashfs or dmsquash. Below, find an example:

```
DISK=/dev/sda
PART=5;S;x,62;L;/,x;L;x,
IMAGE=/dev/sda2;suse-11.0-pxe-client.i386;\
      1.2.8;192.168.100.2;4096
UNIONFS_CONFIG=/dev/sda3,/dev/sda2,aufs
KIWI_INITRD=/boot/initrd
```

11.4.6 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" fsreadwrite="ext3" boot="netboot/suse-11.0">split</type>
- add a split section to describe the writable portion, for example:

```
<split>
   <persistent>
      <!-- allow read/write access to: -->
      <file name="/var"/>
      <file name="/var/*"/>
      <file name="/etc"/>
      <file name="/etc/*"/>
      <file name="/home"/>
      <file name="/home/*"/>
   </persistent>
</split>
```

- sample config.<MAC>

```
IMAGE=/dev/sda2;suse-11.0-pxe-client.i686;\
      1.2.8;192.168.100.2;4096,\
      /dev/sda3;suse-11.0-pxe-client-read-write.i686;\
      1.2.8;192.168.100.2;4096
PART=200;S;x,500;L;/,x;L;
DISK=/dev/sda
COMBINED_IMAGE=yes
KIWI_INITRD=/boot/initrd
```

11.4.7 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
```

11.4.8 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;2000;/dev/nbd0
```

11.4.9 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 command **"vbladed 0 1 eth0 blockdevice"** to be called first

12 OEM image - Preload Systems

Contents

An oem image is a virtual disk image representing all partitions and bootloader information like it exists on a real disk. The image format is the same compared to the VMX image type. All flavours explained in the VMX chapter also applies to the OEM type.

The original idea of an oem image is to provide this virtual disk data to OEM vendors which now are able to deploy the system independently onto their storage media. The deployment can happen from any OS including Windows if a tool to dump data on a disk device exists. The oem image type is also used to deploy images on USB sticks because in principal it is the same workflow.

12.1 Building the suse-oem-preload example

The latest example provided with kiwi is based on openSUSE 11.0 and includes the patterns default plus x11. The image type is a split type whereas the read-write filesystem is ext3 and the read-only filesystem is squashfs. The additional format attribute also creates an installable ISO image for deploying the image from CD.

```
cd /usr/share/doc/packages/kiwi/examples
cd suse-11.0
kiwi --prepare ./suse-oem-preload \
     --root /tmp/myoem
```

```
kiwi --create /tmp/myoem --type split \
     -d /tmp/myoem-result
```

12.2 Using the image

Testing the oem virtual disk can be done with a virtualisation software like QEMU or VMware. The virtual disk is represented by the .raw extension whereas the .iso extension represents the installation disk for this oem image. The installation disk should be tested on a bare test system For the .raw test just call:

```
cd /tmp/myoem-result
qemu suse-11.0-oem-preload.i686-1.1.2.raw \
     -m 512
```

or dump the image on a test hard disk and select it as boot device in the BIOS:

```
cd /tmp/myoem-result
dd if=suse-11.0-oem-preload.i686-1.1.2.raw \
   of=/dev/<device> bs=32k
```

Please note if you test an oem image the virtual disk geometry of the image is the same as the disk geometry inside the host system. According to this the oem boot workflow will skip the re-partitioning which is performed if there would be a real disk

12.3 Flavours

An interesting part of an oem image is that it can be turned into an installation image too. This means it is possible to create an installation CD / DVD or USB stick which deploysthe oem based image onto the selected storage device. The installation process is a simply dd of the image onto the selected device so don't expect any user interaction or GUI here to pop up. KIWI supports two types of installation media:

- <type ... format="iso">...</type>
  Creates a .iso file which can be burned in CD or DVD. This represents an installation CD
- <type ... format="usb">...</type>
  Creates a .raw.install file which can be dumped (dd) on a USB stick. This represents an installation Stick

12.3.1 Influencing the oem partitioning

By default the oemboot process will create/modify a swap, /home and / partition. It is possible to influence the behavior by the following oem-* elements which

can be optionally specified within the **preferences** section of your system image XML description. KIWI uses this to create the file **/config.oempartition** as part of the automatically created oemboot boot image. The format of the file is a simple key=value format and created by the KIWIConfig.sh function named baseSetupOEMPartition. Following oem-* elements can be specified:

- **<oem-reboot>true|false</oem-reboot>**
  This allows to reboot the oem system after initial deployment. This value is represented by the variable OEM_REBOOT in config.oempartition
- **<oem-swapsize>number in MB</oem-swapsize>**
  Set the size of the swap partition. This value is represented by variable OEM_SWAPSIZE in config.oempartition
- **<oem-systemsize>number in MB</oem-systemsize>**
  Set the size of the / partition. This value is represented by the variable OEM_SYSTEMSIZE in config.oempartition
- **<oem-home>true|false</oem-home>**
  Specify if a home partition should be create. This value is represented by the variable OEM_WITHOUTHOME in config.oempartition.
- **<oem-swap>true|false</oem-swap>**
  Specify if a spaw partition should be create. This value is represented by the variable OEM_WITHOUTSWAP in config.oempartition.
- **<oem-boot-title>text</oem-boot-title>**
  By default the string **OEM** will be appended to the boot manager menu when KIWI creates the grub configuration during first deployment. The oem-boot-title value allows to set a custom name which is used instead of OEM. This value is represented by the variable OEM_BOOT_TITLE in config.oempartition.
- **<oem-recovery>true|false</oem-recovery>**
  If this element is set to yes KIWI will create a recovery archive from the prepared root tree. The archive will appear as */recovery.tar.bz2* within the initial image file. During first boot of the image a single recovery partition will be created and the recovery archive will be moved into that partition. An additional boot menu entry will be created which allows to restore the original root tree information. The user information on the /home partition or in the /home directory are not affected by that recovery process
- **<oem-kiwi-initrd>true|false</oem-kiwi-initrd>**
  If this element is set to yes the initial oemboot boot image (initrd) will **not** be replaced by the system (mkinitrd) created initrd. This option makes sense if the target storage device for the image is not a fixed disk but for example an USB stick. In that case it might be required to re-detect the storage location on first boot which is done as part of the oemboot boot image

13 XEN image - Paravirtual Systems

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.[1] 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 U" (domU) in Xen terminology.

A xen image is a filesystem based image file which requires the Xen dom0 running or the project called Xenner which emulates the capabilities of the domain 0. The image created with kiwi can only be used together with the xen tools.

13.1 Building the suse-xen-guest example

The latest example provided with kiwi is based on openSUSE 11.0 and includes the base pattern.

```
cd /usr/share/doc/packages/kiwi/examples
cd suse-11.0
kiwi --prepare ./suse-xen-guest \
     --root /tmp/myxen
```

```
kiwi --create /tmp/myxen \
     --type xen -d /tmp/myxen-result
```

13.2 Using the image

In order to run a domain U the Xen tool **xm** needs to be called in conjunction with the KIWI genereated domain U configuration file

```
xm create -c \
   /tmp/myxen-result/\
   suse-11.0-xen-guest.i686-1.1.2.xenconfig
```

13.3 Flavours

With KIWI you can provide the information required to create a guest configuration as part of the config.xml file. Additionally you can group special packages which you may only need in this para virtual environment.

```
<packages type="xen">
   <!-- packages you need in Xen only -->
   <package name="kernel-xen"/>
   <package name="xen"/>
</packages>
<xenconfig memory="512" domain="domU">
   <xendisk device="/dev/sda"/>
</xenconfig>
```

If this information is present KIWI will create a Xen domain U (or domain 0) configuration with 512 MB of RAM and expects the disk at /dev/sda. Additional information to setup the Xen guest machine properties are explained in the **xenconfig** section. The KIWI Xen domain U configuration is stored in the file:

```
/tmp/myxen-result/\
    suse-11.0-xen-guest.i686-1.1.2.xenconfig
```

14 EC2 image - Amazon Elastic Compute Cloud

Contents

The Amazon Elastic Compute Cloud (Amazon EC2) web service provides you with the ability to execute arbitrary applications in our computing environment. To use Amazon EC2 you simply:

1. Create an Amazon Machine Image (AMI) containing all your software, including your operating system and associated configuration settings, applications, libraries, etc. Such an AMI can be created by the kiwi ec2 image type. In order to do that kiwi makes use of the tools provided by Amazon. Your build system should have these tools installed. Due to license issues we are not allowed to distribute the tools which means you need to download, install and setup them from here: http://docs.amazonwebservices.com/AmazonEC2/gsg/2
2. Upload this AMI to the Amazon S3 (Amazon Simple Storage Service) service. This gives us reliable, secure access to your AMI.
3. Register your AMI with Amazon EC2. This allows us to verify that your AMI has been uploaded correctly and to allocate a unique identifier for it.
4. Use this AMI ID and the Amazon EC2 web service APIs to run, monitor, and terminate as many instances of this AMI as required. Currently, Amazon provides command line tools and Java libraries but you may also directly access the SOAP-based API.

Please note while instances are running, you are billed for the computing and network resources that they consume. You should start creating an ec2 with kiwi after you can make sure your system is prepared for ec2 which means if you call the command **ec2-describe-images -a** you will get a valid output.

14.1 Building the suse-ec2-guest example

The latest example provided with kiwi is based on openSUSE 11.0 and includes the base pattern plus the vim editor.

Before you run kiwi you need to include some of your ec2 account information into the image description config.xml file. The box below shows the values you need to adapt:

```
<type primary="true"
  ec2accountnr="12345678911"
  ec2privatekeyfile="Path to EC2 private key file"
  ec2certfile="Path to EC2 public certificate file"
>ec2</type>
```

After that call kiwi as follows:

```
cd /usr/share/doc/packages/kiwi/examples
cd suse-11.0
kiwi --prepare ./suse-ec2-guest \
     --root /tmp/myec2
```

```
kiwi --create /tmp/myec2 \
     --type ec2 -d /tmp/myec2-result
```

14.2 Using the image

The generated image needs to be transfered over to Amazon which is done by the ec2-upload-bundle tool. You can do this by calling:

```
ec2-upload-bundle -b myImages \
 -a <AWS Key ID> -s <AWS secret Key ID> \
 -m /tmp/myec2/\
    suse-11.0-ec2-guest.i686-1.1.2.ami.manifest.xml
```

After this is done the image needs to be registered in order to receive a so called AMI id which starts with **ami-** followed by a random key sequence. To register call:

```
ec2-register myImages/\
    suse-11.0-ec2-guest.i686-1.1.2.ami.manifest.xml
```

The result is the AMI id which you need to run an instance from your image. The command ec2-describe-images allows you to review your registered images. Since you will be running an instance of a public AMI, you will need to use a public/private keypair to ensure that only you will have access. One half of this keypair will be embedded into your instance, allowing you to login securely without a password using the other half of the keypair. Every keypair you generate requires a name. Be sure to choose a name that is easy to remember, perhaps one that describes the image's content. For our example we'll use the name gsg-keypair.

```
ec2-add-keypair gsg-keypair
```

The private key returned needs to be saved in a local file so that you can use it later. Using your favorite text editor, create a file named **id_rsa-gsg-keypair** and paste everything between (and including) the —*BEGIN RSA PRIVATE KEY*— and —*END RSA PRIVATE KEY*— lines into it. To review your keypairs call:

```
ec2-describe-keypairs
```

We are almost done now but to be able to run an instances you also need to specifiy which kernel and boot image (initrd) should be used to run the instance. Kernels are registered as aki-... images and initrd's are registered as ari-... images at Amazon. You will need to select a aki/ari image that matches your ami. The following table shows which Distributions are supported at the moment:

Distro	**ARI id**	**AKI id**	**Arch**
SUSE 11.0	ari-49db3f20	aki-4adb3f23	ix86
SUSE 11.0	ari-48db3f21	aki-4ddb3f24	x86_64

For this example we need the id's provided for openSUSE 11.0. According to this call the following command to fire up your new ec2 instance:

```
ec2-run-instances ami-... \
  --kernel aki-4adb3f23 --ramdisk ari-49db3f20 \
  -k gsg-keypair
```

To check the state of your instance(s) call the following command:

```
ec2-describe-instances
```

If you see your instance at the status: **running** you can login into it. If you can't make sure you have allowed port 22 to be available

```
ec2-authorize default -p 22
```

Congratulations ! You made it and can now use Amazons storage and computing power.

15 KIWI testsuite

Contents

The KIWI test suite is useful to perform basic quality checks on the image root directory. The test cases are stored in subdirectories below /usr/share/kiwi/tests. To run the testsuite call kiwi as follows:

```
kiwi --testsuite <image-root> \
    [ --test name --test name ... ]
```

If not test names are set the default tests rpm and ldd run. The name of a test corresponds with the name of the directory the test is implemented in.

15.1 testsuite packages

If a test requires special software to be installed but this software is not an essential part of the image itself it can be specified as testsuite packages in the system image config.xml as follows:

```
<packages type="testsuite">
    <package name="..."/>
</packages>
```

The testsuite packages are installed when calling kiwi with the testsuite option and are removed after the tests has finished.

15.2 Creating a test

The test itself is defined by a xml description ”test-case.xml” and its template definition file /usr/share/kiwi/modules/KIWISchemaTest.rnc The following example shows the basic structure of the rpm test:

```
<test_case
  name="rpm"
  summary="check rpm database and verify all rpms"
  description="check if rpm db is present, run rpm‘s build-in Verify method"

  <requirements>
     <req type="directory">/var/lib/rpm</req>
     <req type="file">/var/lib/rpm/__db.000</req>
     <req type="file">/var/lib/rpm/Packages</req>
  </requirements>

  <test type="binary"  place="extern">
     <file>rpm.sh</file>
     <params>CHROOT</params>
  </test>
</test_case>
```

There are basically two sections called ”equirements” and ”test”. In requirements you define what files/directories or packages has to be present in your image to run the test. For example if you need to check the rpm database, the database has to be present within the image. All requirements are checked, and if any of them fail the test won’t be executed and an error message is printed. There are three types of requirements:

- **file**
  Existence of a file
- **directory**
  Existence of a directory
- **rpm-package**
  Existence of a package

The test section defines the test script. It could be a binary, shell script or any other kind of executable. Scripts are expected to be in the same directory as where the xml definition for the test resides. There are two types of scripts, extern and intern.

- external scripts are executed outside of the image and are preferred. Their first parameter should be CHROOT. This parameter is changed to the real path of the image chroot directory.
- internal scripts are executed inside image using the ”chroot” command. Files are copied into the image and deleted after execution.

A test script always has to return 0 in case of a test to pass, or 1 if any error occur. All messages printed to standard and error output are stored and printed out of the test has failed.

Index