openSUSE-KIWI Image System

Cookbook

Marcus Schäfer

openSUSE-KIWI Image System: Cookbook

by Marcus Schäfer
Thomas Schraitle <toms@suse.de>

KIWI Version 4.75

License

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or (at your option) version 1.3; with the Invariant Section being this copyright notice and license. A copy of the license version 1.2 is included in the appendix entitled “GNU Free Documentation License”.

SUSE®, openSUSE®, the openSUSE® logo, Novell®, the Novell® logo, the N® logo, are registered trademarks of Novell, Inc. in the United States and other countries. Linux® is a registered trademark of Linux Torvalds. All other third party trademarks are the property of their respective owners.

All information found in this book has been compiled with utmost attention to detail. However, this does not guarantee complete accuracy. Neither Novell, Inc., SUSE Linux Products GmbH, the authors, nor the translators shall be held liable for possible errors or the consequences thereof.

Table of Contents

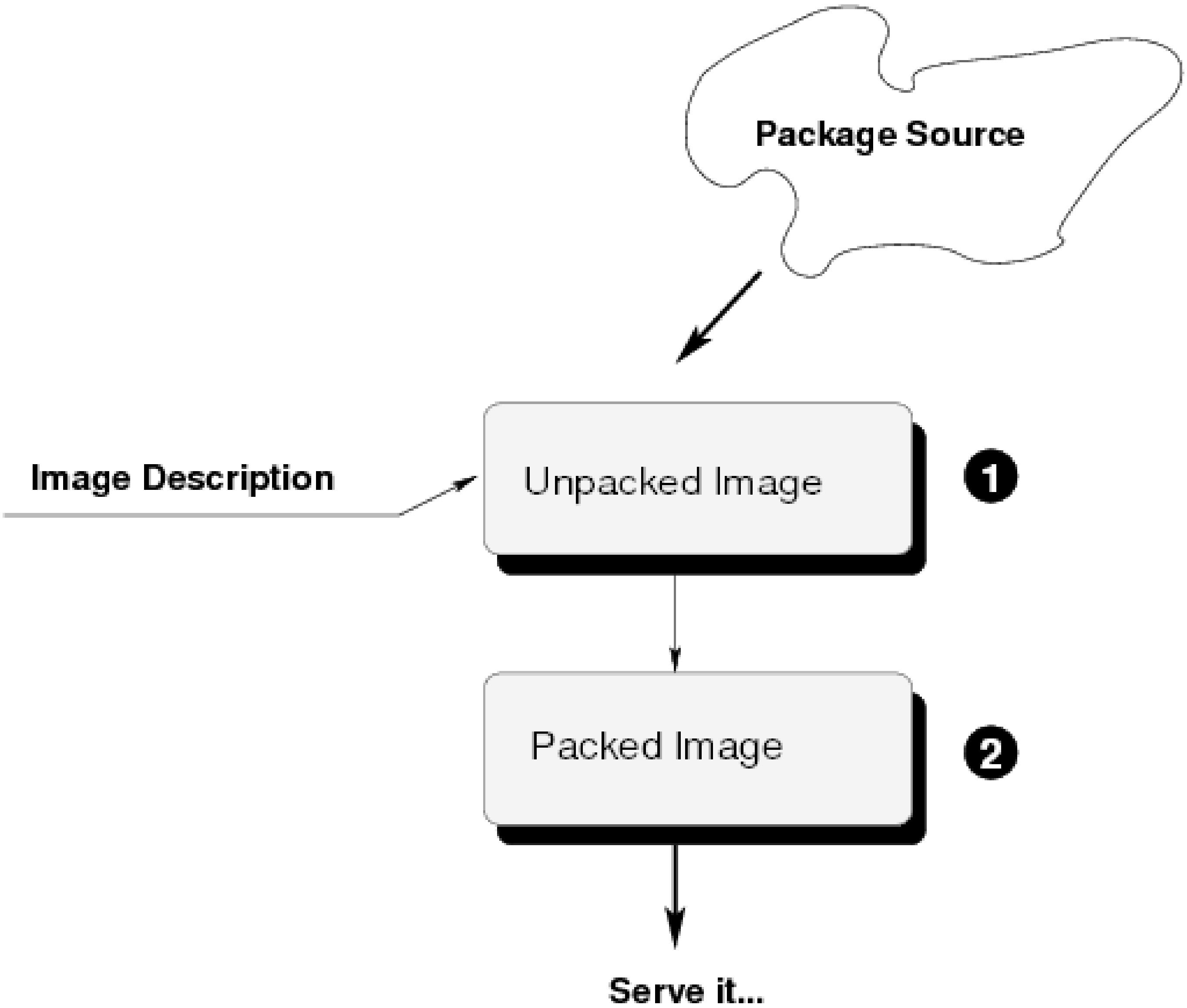
1 Basic Workflow

Table of Contents

1.1. Introduction

The openSUSE KIWI Image System provides a complete operating system image solution for Linux supported hardware platforms as well as for virtualization 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



❶ Encapsulated system reachable via chroot
❷ Encapsulated system reachable via kernel filesystem/extension drivers

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 manual user intervention may be 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 have to be fulfilled before the image is created. All of this information is stored in the *image description*.

1.2. Boot Process

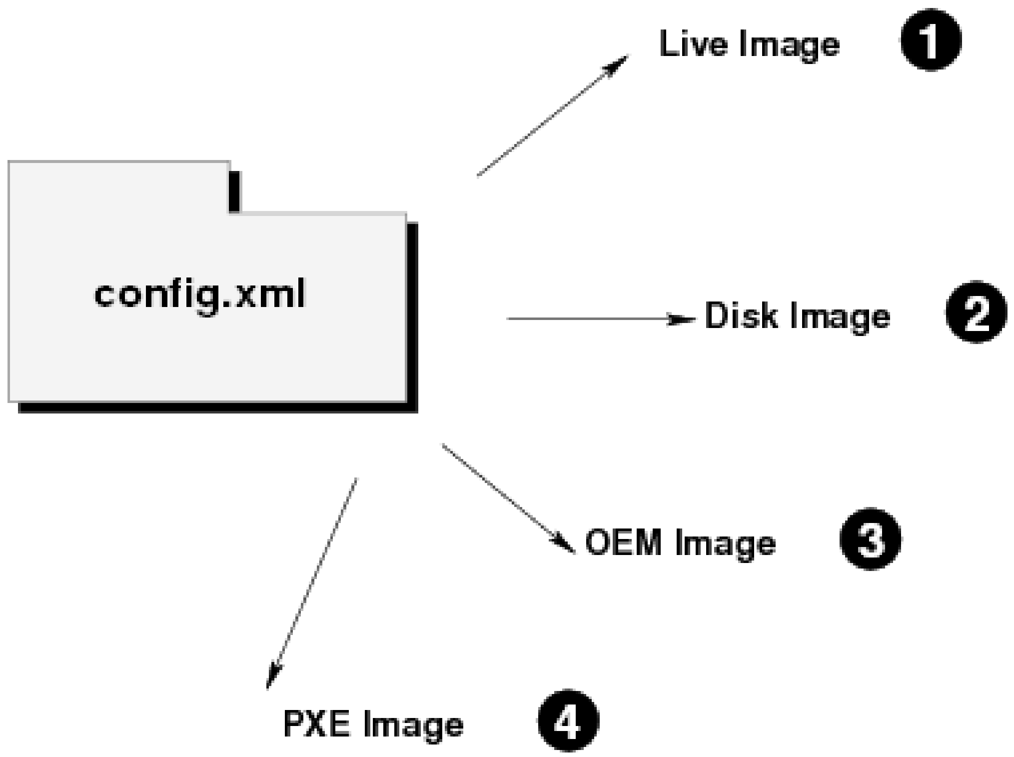
The creation of an image with KIWI is always divided into two basic steps: the *prepare* and the *create* step. The create step requires the prepare step to be completed successfully. Within this first prepare step, KIWI builds a new root tree or, in KIWI speak, a new unpacked image. The building of 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 package manager. KIWI supports the smart package managers. The prepare step executes the following major stages:

- **Creating Root Directory.** 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.
- **Installing Packages.** 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
- **Executing User Defined config.sh Script.** At the end of the preparation 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, refer to the kiwi::config.sh(1) man page.
- **Managing 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 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 follows or, in KIWI speak, a new packed image. 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:

- **Executing 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`. However, it is most often used to remove packages which were pulled in by a dependency, but are not really required for later use of the operating system. For a detailed description what functions are already available to `images.sh`, refer to the kiwi::images.sh(1) man page.
- **Creating Requested Image Type.**
  What image type(s) a KIWI image supports, depends on what types have 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 1.2. Image Types

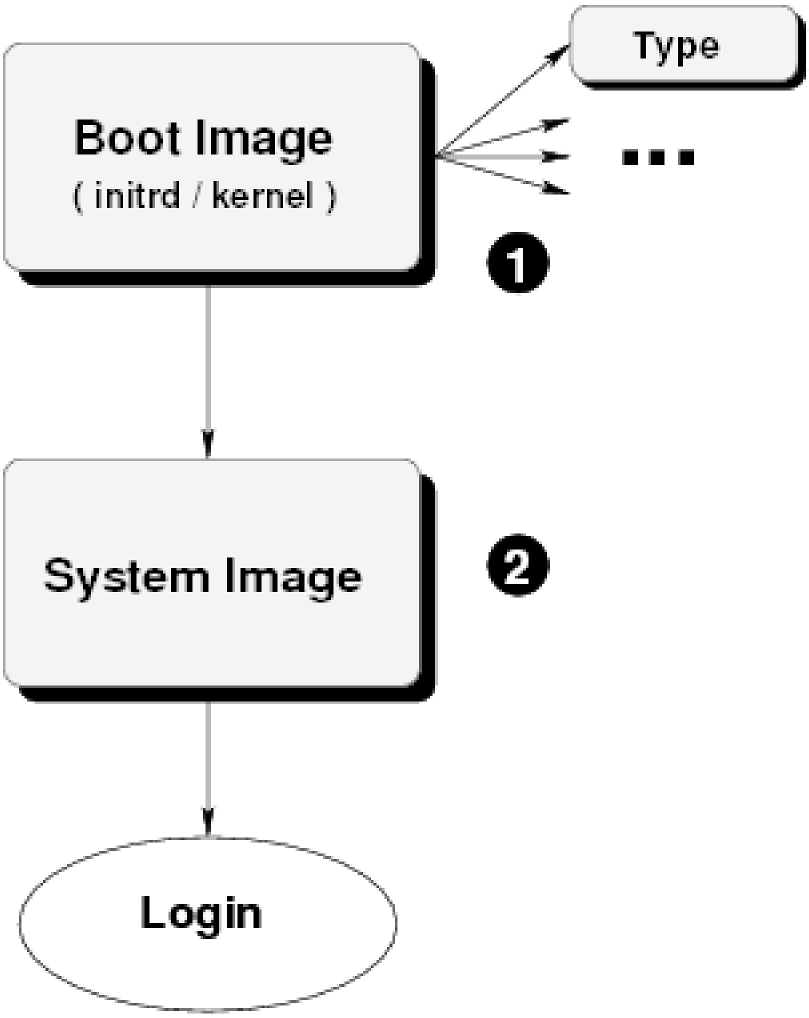


1. Live Image on CD, DVD or USB stick
2. Full virtual system which can be played in VMware, Xen, Amazon Cloud, etc. Guest configuration can be created.
3. Preload system with install media on CD or USB stick
4. Network boot image. Kiwi also provides the bootp environment via the package kiwi-pxeboot

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.

Today's Linux systems are using a special boot image to control the boot process. This boot image is called initrd. The Linux kernel loads this compressed cpio initial ramdisk into RAM and calls init or (if present) the program linuxrc. The KIWI image system also handles the creation of this boot image. Each image type has its 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 1.3. Boot Process



❶ Descriptions are provided by KIWI, use is recommended but not required
❷ A description needs to be created or a template can be used

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 it represents the later operating system, whereas the boot image only lives temporarily in RAM as long as the system image will be activated. The boot image descriptions are stored in `/usr/share/kiwi/image/*boot` and can be built in the same way as the system image. A boot image without a corresponding system image doesn't make sense, though.

1.3. 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. 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 specific 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 seconds. 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. This shell contains some basic standard commands 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 high-level 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 behavior.

1.4. Common and Distribution Specific Code

KIWI has been developed to be usable for any Linux distribution. However, each Linux distribution is different. On one hand, KIWI provides common code

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 the case of specific tasks mostly related to booting, KIWI provides a set of functions which all contain a distribution prefix. As this project uses SUSE Linux as base distribution, all required distribution specific tasks have been implemented for SUSE. Other distributions could be missing. The existing implementation for SUSE can 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-”. If you see such a prefix, script, or function, you can be assured that this is something distribution specific. If you plan to use KIWI with another distribution than SUSE, you need to adapt it. For example, the boot workflow is controlled by a program called **linuxrc** which in KIWI is a script represented by **suse-linuxrc**. Another example is the function `suseStripKernel`, which is able to remove everything but a specified list of kernel drivers from the SUSE kernel.

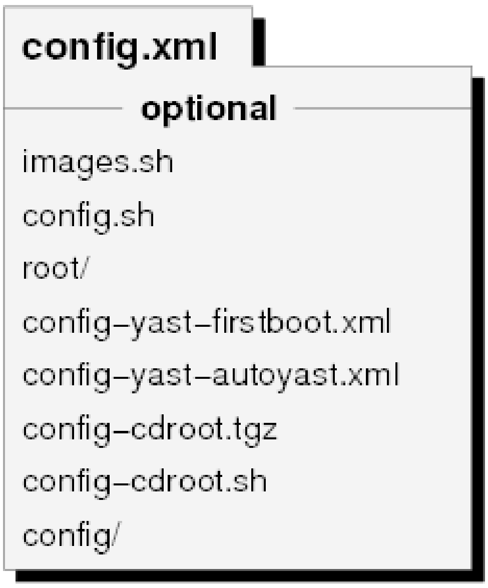
The prefixed implementation allows us to integrate all the distribution specific tasks into one project but this of course requires the help and knowledge of the people who are familiar with their preferred Linux distribution.

2 KIWI Image Description

Table of Contents

In order to be able to create an image with KIWI, a so called image description must be created. The image description is represented by a directory which has to contain at least one file named `config.xml` or `*.kiwi`. A good start for such a description can be found in the examples provided in `/usr/share/doc/packages/kiwi/examples`.

Figure 2.1. Image Description Directory



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

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

`config.sh`
: Optional configuration script while creating the unpacked image. This script is called at the end of the installation, but *before* the package scripts have run. It is designed to configure

the image system, such as the activation or deactivation of certain services (`insserv`). The call is not made until after the switch to the image has been made with chroot.

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

`config-yast-firstboot.xml`
: Configuration file for the control of the YaST 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 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 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 `fileconfig-yast-firstboot.xml`.
4. If the process finished successfully, the environment is cleaned and firstboot will not 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, run:

```
yast2 autoyast
```

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

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

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

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

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

2.1. The config.xml File

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

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

2.1.1. `image` Element

```
<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 filenames created by KIWI. Because we don't want spaces in filenames the `name` attribute must not have any spaces in its name.

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

`displayname`
: allows setup of the boot menu title for 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.

`inherit`
: inherits the packages information from another `image` description

`kiwirevision`
: specifies 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`.

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

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

2.1.2. `description` Element

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

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

2.1.3. `profiles` Element

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

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

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

2.1.4. `preferences` Element

```
<preferences profiles="name">
  <version>1.1.2</version>
  <packagemanager>smart</packagemanager>
  <type image="name" ...>
    <ec2config|systemdisk|oemconfig|pxedeploy|size|split|machine>
  </type>
</preferences>
```

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

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

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

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

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

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

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

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

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

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

  clic
  : Creates a fuse based compressed read-only filesystem which allows write operations into a cow file.

  compressed
  : Compressed filesystem with squashfs mounted with an aufs based overlay mount to allow read-write access.

  unified
  : Compressed filesystem with squashfs mounted with an aufs based overlay mount to allow read-write access.

  If the flags attribute is not used the filesystem will not be compressed and no union filesystem is used. In this case it is recommended to specify a split section as a child of this type element. The specification of a split block is also recommended when flags="compressed" is used.

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

is kiwiVG. Further configuration of the image is performed using the appropriate `*config` child block.

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

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

`image="usb"`
: Use the usb value to create a USB stick image. Set the `filesystem` attribute to the desired supported filesystem for the image and use the `boot="usbboot/suse-*"` attribute to select the USB boot image for the system. For a USB image you may also select GRUB or Syslinux as a bootloader by setting the optional `bootloader` attribute to grub or syslinux, respectively. The boot timeout value my be set by using the optional `boottimeout` attribute. The unit for the timeout value is seconds if GRUB is used as the boot loader and 1/10 seconds if syslinux is used as the boot loader. The USB image may also be created with LVM support. The same rules as indicated for the OEM image apply.

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

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:

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

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

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

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

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

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

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

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

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

ec2config
: The optional ec2config block is used to specify information relevant only to AWS EC2 images. The following information can be provided:

```
<ec2config>
  <ec2accountnr> Your AWS account number </ec2accountnr>
  <ec2certfile> Path to the AWS cert-*.pem file </ec2certfile>
```

```
  <ec2privatekeyfile> Path to the AWS pk-*.pem file </ec2privatekeyfile>
</ec2config>
```

systemdisk

Using the optional systemdisk section it possible to create a LVM (Logical Volume Management) based storage layout. By default, the volume group is named kiwiVG. It is possible to change the name of the group by setting the lvmgroup attribute to the desired name. Individual volumes within the volume group are specified using the volume element.

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

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

With the optional freespace attribute it is possible to add space to the volume. If the freespace attribute is not set the created volume will be 80 % to 90 % full. Using the optional size attribute the absolute size of the given volume is specified. The size attribute takes precedence over the freespace attribute. Should the specified size be too small the value will be ignored and a volume with approximately 80 % to 90 % fill will be created.

oemconfig

By default, the oemboot process will create or modify a swap, and / partition. It is possible to influence the behavior by the oem-* elements explained below. KIWI uses this information 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.

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

<oem-align-partition>>true|false</oem-align-partition>

Kiwi attempts to align the start sector of the disk partition on a 4K boundary.

<oem-boot-title>text</oem-boot-title>

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

<oem-bootwait>true|false</oem-bootwait>

Specify if the system should wait for user interaction priot to continuing the boot process after the oem image has been dumped to the designated storage device (default value is false). This value is represented by the OEM_BOOTWAIT variable in config.oempartition.

<oem-inplace-recovery>true|false</oem-inplace-recovery>

Specify if the recovery archive is stored as part of the image or if the archive is to be created at the time the image is deployed to the target storage device. OEM_RECOVERY_INPLACE variable in config.oempartition.

<oem-kiwi-initrd>true|false</oem-kiwi-initrd>

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

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

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

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

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

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

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

`<oem-shutdown-interactive>`true|false`</oem-shutdown-interactive>`
: Specify if the system is to be powered down after the oem image has been deployed to the designated storage device (default value is false). Prior to shutdown a message is posted and must be acknowledged by the user in order for the system to power off. This value is represented by the `OEM_SHUTDOWN_INTERACTIVE` variable in `config.oempartition`.

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

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

<oem-systemsize>number in MB</oem-systemsize>
: Set the size of the root partition. This value is represented by the variable OEM_SYSTEMSIZE in config.oempartition.

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

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

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

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

- The server attribute is used to specify the IP address of the PXE server. The blocksize attributes specifies the blocksize for the image download. Other protocols are supported by KIWI but require the *kiwiserver* and *kiwiservertype* kernel parameters to be set when the client boots.
- The value of the optional timeout element specifies the grub timeout in seconds to be used when the KIWI initrd configures and installs the grub boot loader on the client machine after the first deployment to allow standalone boot.
- Passing kernel parameters is possible with the use of the optional kernelcmdline attribute in the type section. The value of this attribute is a string specifying the settings

to be passed to the kernel by the GRUB bootloader. The KIWI initrd includes these kernel options when installing grub for standalone boot

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

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

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

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

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

`size`

Use the size element to specify the image size in Megabytes or Gigabytes. The unit attribute specifies whether the given value will be interpreted as Megabytes (`unit="M"`) or Gigabytes (`unit="G"`). The optional boolean attribute additive specifies whether or not the given size should be added to the size of the generated image or not.

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

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

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

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

split

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

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

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

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

machine

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.1.5. users Element

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

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

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

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

2.1.6. drivers Element

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

2.1.7. `repository` Element

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

The mandatory `repository` element 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 have been copied from smart. The following table shows the possible repo types:

Table 2.1. Supported Types for zypper and smart

Type	smart Support	zypper Support
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 repository priority assigned to all packages available in this repository. For smart the following applies: If the exact same package is available in more than one channel, the repository with the *highest* priority number is used. The value 0 means “no priority is set”. For zypper the following applies: If the exact same package is available in more than one channel, the repository with the *lowest* priority number is used. The value 99 means “no priority is set”.

`username="name"`
: Specifies the repository user name. It depends on the source type whether this information is used or not

`password="string"`
: Specifies the repository password. It depends on the source type whether this information is used or not

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

  When using multiple .iso files from the same product, such SLES all .iso files need to be located in the same directory, but only the first .iso file needs to be added as a repository to the configuration. The first .iso file contains sufficient information for the package management tool to find packages in the other .iso files as long as they are located in the same directory. Attempting to use a second or third .iso file in a series as a standalone repository will result in an error.

`plain://`*`URI`*
: A plain resource string. Everything following 'plain://' will be forwarded to the package manager without further modifications. This can be used if kiwi does not support a special URI but the package manager does.

`smb://`*`Samba share pathname`*
: A path to a samba share using the cifs protocol. kiwi creates a mount point and mount the share including username and password if specified. In order to allow the mount inside the new root tree, make sure you have installed the package which provides the cifs mount utility as part of the type='bootstrap' package section. Currently this package is named

cifs-utils. Because the mount utility has to be part of the system and the boot image you also need to make sure that the package is bootincluded like the following example shows:

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

`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://PROJECTNAME`
: 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`

2.1.8. packages Element

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

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

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

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

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

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

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

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

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

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

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

usb
: Image packages, a list of additional packages to be installed when building a USB image.

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

2.1.8.1. Using Patterns

Using a pattern name allows you to considerably shorten the list of specified packages in the config.xml file. A named pattern, specified with the opensusePattern element is a representation of a predefined list of packages. Specifying a pattern will install all packages listed in the named pattern to be installed in the image. Support for patterns is SUSE-specific, and available with openSUSE 10.1 or later. The optional patternType attribute on the packages element allows you to control the installation of dependent packages in the image. You may assigne one of the following values to the patternType attribute:

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

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

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

2.1.8.2. Architecture Restrictions

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

2.1.8.3. 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

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

and in the type="xen" section as

```
<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.

2.1.8.4. 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

2.1.8.5. Data not Available as Packages to Become Included

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

3 Creating Appliances with KIWI

Table of Contents

3.1. 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.

3.2. The KIWI Model

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

1. Choose an appropriate image description template from the provided KIWI examples. Add or adapt repositories, package names or both, according to the distribution you want to build an image for.

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

   ```
   baseSetupPlainTextGITRepository
   ```

3. Prepare the preliminary version of your new appliance by calling **kiwi** `--prepare`… and refer to Chapter 8, *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 8, *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.

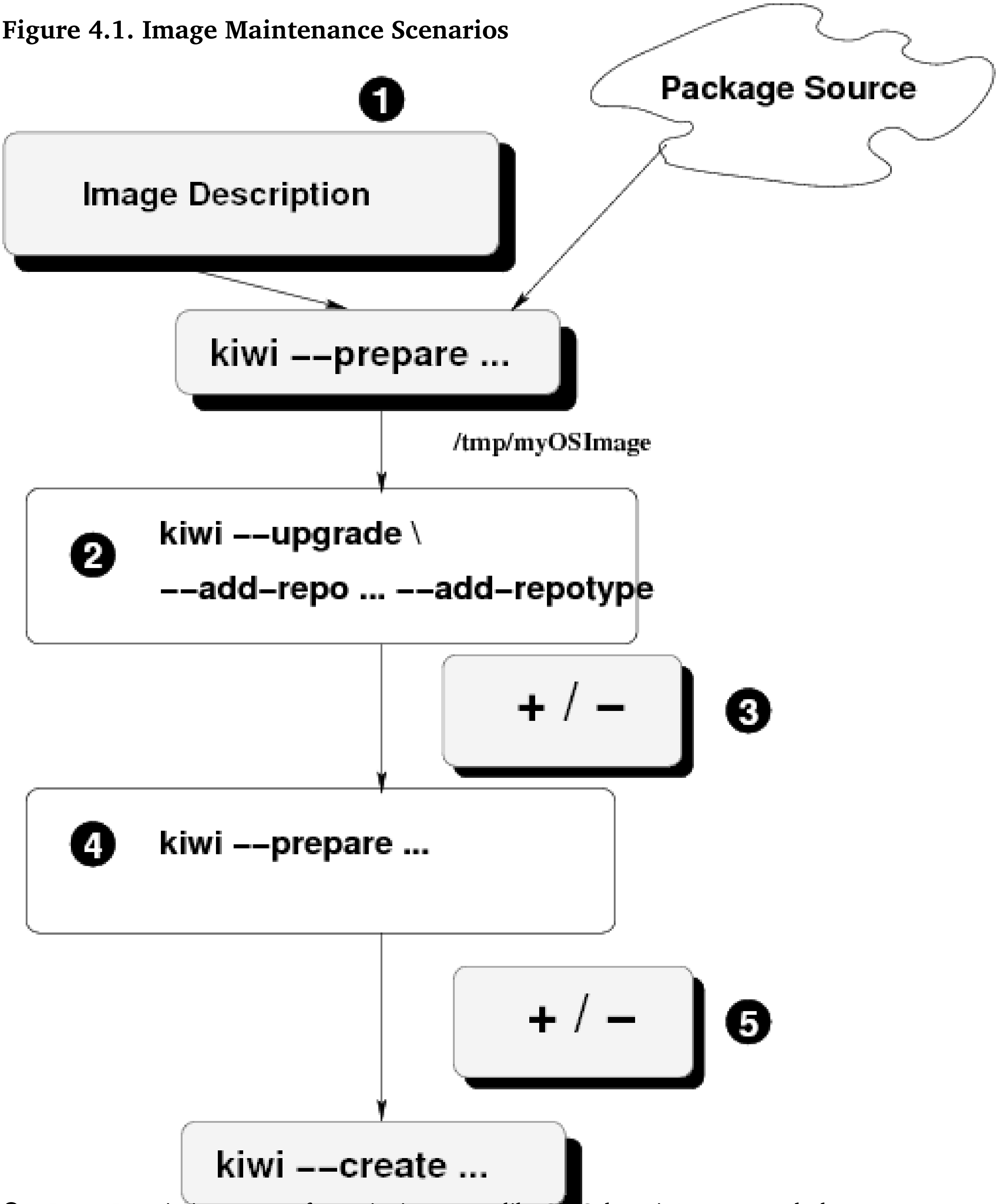
9. All your valuable work is now stored in one image description and can be re-used in all KIWI supported image types.

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.

4 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 4.1. Image Maintenance Scenarios



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

The picture in Figure 4.1 shows two possible scenarios which requires an image to become updated. The first reason for updating an image are changes to the software, for example a new kernel should be used. If this change doesn't require additional software or changes in the configuration the update can be done by KIWI itself using its `--upgrade` option. In combination with `--upgrade` KIWI allows to add an additional repository which may be needed if the updated software is not part of the original repository. An important thing to know is that this additional repository is *not* stored into the original `config.xml` file of the image description.

Another reason for updating an image beside software updates are configuration changes or enhancements, for example an image should have replaced its browser with another better browser or a new service like apache should be enabled. In principle it's possible to do all those changes manually within the 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:

> Don'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.

5 System to Image Migration

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

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

5.1. Create a Clean Repository Set First

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

```
kiwi --migrate mySystem
```

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

Continue calling the following command until your list is clean You should continue the migration if you have a clean list of solved packages without any package skipped except you know that this package can't be provided or is not worth to become part of the migration.

```
kiwi --migrate mySystem --nofiles [--skip package ... ]
```

5.2. Watch the Overlay and Unpackaged Files

Files which has been modified but belong to a package will be automatically copied into the overlay directory below `/tmp/mySystem/root`. You should check that no modified file is a binary because such a binary would be replaced by a new install of the package anyway. Software developers tend to compile software from source and copy/install them into their system. Doing this could cause binary files previously installed by a package to be reported as modified. You should remove such files from your overlay tree.

The migration also copy the entire `/etc` directory into the overlay root directory because it stores all important configuration files. Beside the important files there are most probably a bunch of file which doesn't belong to any package exists only for historical reasons. kiwi creates a list of files and directories to support you best in sorting out what is required and what can be ignored. Nevertheless this is the most time consuming part of your migration review. Simply click on the *all unpackaged files* link to take a look at the complete list. Those files you want to have in your image needs to be copied over to the `/tmp/mySystem/root` directory

5.3. Checklist

After that you should walk through the following check list

- Change author and contact in `config.xml`
- Set appropriate name for your image in `config.xml`.
- Add/modify default type (oem) set in `config.xml` if needed
- Make sure your X11 configuration is appropriate according to the new target. A failsafe version was created in `/tmp/mysys/root/etc/X11/xorg.conf.install` -> fbdev based
- Make sure **yast2** is installed to be able to reconfigure the system. If **yast2** is not installed these tasks needs to be done else. Otherwise yast's second stage is started on first boot of the migrated image
- If you want to access any remote filesystem it's a good idea to let AutoYaST add them on first boot of the system
- Check your network setup in `/etc/sysconfig/network`. Is this setup still possible in the cloned environment? Make sure you check for the MAC address of the card first.

5.4. 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 make sure to maintain only required overlay files. Now let's try to create a clone image from the description. By default an OEM image which is a virtual disk which is able to run on real hardware too is created.

On success you will also find a ISO file which is an installable version of the OEM image. If you burn the ISO on a DVD you can use that DVD to install your cloned image on another computer.

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

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

6 Installation Source

Table of 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.

6.1. 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>
```

6.2. Create a Local Installation Source

The following procedure 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 document 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 previously 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. If you do not have a DVD but a CD set, copy the contents of *all* CDs into one directory. It's absolutely important that you first start with the *last* CD and copy the first CD at last. In case of CDs you should have a bundle of 4 CDs. Copy them in the order 4 3 2 1.

3. Copy the contents of the CDs/DVD to your hard drive once you have access to the media. 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, eject the CD and insert the next one to repeat the copy command until all CDs are copied into to /image

7 ISO Image—Live Systems

Table of Contents

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

7.1. Building the suse-live-iso Example

The latest example provided with KIWI is based on openSUSE 11.2 and includes the base and KDE patterns.

```
cd /usr/share/doc/packages/kiwi/examples cd suse-11.2
kiwi --prepare ./suse-live-iso --root /tmp/myiso
```

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

7.2. Using the Image

There are two ways to use the generated ISO image:

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

- Use a virtualization system to test the image directly. Testing an iso can be done with any full virtual system for example:

  ```
  cd /tmp/myiso-result
  qemu -cdrom ./suse-11.2-live-iso.i686-2.5.1.iso -m 256
  ```

7.3. 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 filesystem 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="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, 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.

7.3.1. Split mode

If no overlay filesystem is in use but the image filesystem is based on a compressed filesystem KIWI allows to setup which files and directories should be writable in a so called split section. In order to allow to login into the system, at least the `/var` directory should be writable. This is because the PAM authentication requires to be able to report any login attempt to `/var/log/messages` which therefore needs to be writable. The following split section can be used if the flag compressed is used:

```
<split>
  <temporary>
   <file name="/var"/>
   <file name="/var/*"/>
   <file name="/boot"/>
   <file name="/boot/*"/>
   <file name="/etc"/>
   <file name="/etc/*"/>
   <file name="/home"/>
   <file name="/home/*"/>
  </temporary>
</split>
```

8 USB Image—Live-Stick System

Table of 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.

8.1. Building the suse-live-stick Example

The next example provided with KIWI is based on openSUSE 11.2 and uses the default plus x11 pattern. The operating system is stored on a standard ext3 filesystem:

```
cd /usr/share/doc/packages/kiwi/examples
cd suse-11.2
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, use the first approach. In this case it is 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
  ```

8.2. Using the Image

To use the created images deployed them 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-11.2.i686-2.1.1.splash.gz \
    --bootstick-system \
    /tmp/mystick-result/ \
    suse-11.2-live-stick.i686-1.1.2
```

In case of the second image type (oem) dump the raw 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. However, this partition information does not match the real USB stick geometry. This means, the KIWI boot image (oemboot) has to adapt the disk geometry on first boot. To deploy the image on the stick, run:

```
dd if=/tmp/mystick-result/ \
      suse-11.2-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 (booting from USB) or with a virtualization system. If you test with a virtualization system, for example qemu, 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. However, in a virtual environment the disk doesn't appear as a USB stick. If your virtualization solution doesn't provide a virtual BIOS which allows booting from USB stick, test the stick on real hardware.

8.3. 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 gigabytes. Therefore, KIWI supports compressed filesystems on USB sticks too:

`filesystem="squashfs"`
: Compresses 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="clicfs"`
: Creates a fuse based clicfs image and allows write operations into a cow file.

8.3.1. Split Stick

If there is no overlay filesystem available, it is also possible to define a split section in `config.xml`. 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 image="split" fsreadwrite="ext3"
     fsreadonly="squashfs" boot="usbboot/suse-11.2"/>
  ```

- Type setup for split oem type:

```
<type image="split" fsreadwrite="ext3"
   fsreadonly="squashfs" boot="oemboot/suse-11.2"/>
```

For both types, a split section inside the type 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>
```

If no split section is added the default split section from `/usr/share/kiwi/modules/`, the file `KIWISplit.txt` is used.

8.3.2. LVM Support

KIWI supports LVM, the Logical Volume Manager. In this mode, the disk partition table includes one lvm partition and one standard ext2 boot partition. KIWI creates the kiwiVG volume group and adds logical volumes as they are needed and configured according to the image type and filesystem. After booting, the user has full control over the volume group and is free to change, resize, or increase the group and the volumes inside. Support for LVM has been added for all image types which are disk based. This includes vmx, oem and usb. In order to use LVM for the usb type just add the `--lvm` option as part of **kiwi** `--bootstick` deployment or add the attribute `lvm="true"` as part of the `type` section in your `config.xml` file.

The optional `systemdisk` section can be used to set one or more top level directories into a separate volume. See Chapter 2, *KIWI Image Description* for a detailed explanation.

9 VMX Image—Virtual Disks

Table of Contents

A VMX image is a virtual disk image for use in full virtualization systems like Qemu or VMware. The image is a file containing the system represented by the configured packages in config.xml as well as partition data and bootloader information. The size of this virtual disk can be specified by using the size element in the `config.xml` file or by adding the `--bootvm-disksize` command line argument.

9.1. Building the suse-vm-guest Example

The vm-guest example provided with KIWI is based on recent openSUSE releases, one example configuration per release. The example uses base pattern and the virtual disk is formatted using the distribution default filesystem.

```
cd /usr/share/doc/packages/kiwi/examples
cd suse-11.2
kiwi --prepare ./suse-vm-guest --root /tmp/myvm
```

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

9.2. Using the Image

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

```
cd /tmp/myvm-result
qemu suse-11.2-vm-guest.i686-1.1.2.raw -m 256
```

9.3. Flavours

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

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

The following table lists the supported virtual disk formats:

Table 9.1. Supported Virtual Disk Formats

Name	Description
vmdk	Disk format for VMware
ovf	Open Virtual Format requires VMware's ovftool
qcow2	QEMU virtual disk format

9.3.1. VMware support

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

Within the `config.xml` file it is possible to specify the VMware configuration settings. In addition it is possible to include selected packages in the created image that are specific to the VM image generation. The following `config.xml` snippet provides general guidance on the elements in `config.xml`.

```
<packages type="vmx">
   <!-- packages you need in VMware only -->
</packages>
<type......>
  <machine memory="512">
    <vmdisk controller="ide" id="0"/>
  </machine>
</type>
```

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

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

```
/tmp/myvm-result/suse-11.2-vm-guest.i686-1.1.2.vmx
```

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

In addition it is possible to create an image for the Xen virtualization framework. By adding the `bootprofile` and `bootkernel` attributes to the `<type>` start tag with values of `xen` and `xenboot`, respectively. Please refer to the Chapter 12, *Xen Image—Paravirtual Systems* for additional details.

9.3.2. LVM Support

KIWI also provides support for LVM (Logical Volume Management). In this mode the disk partition table will include one lvm partition and one standard `ext2` boot partition. KIWI creates the kiwiVG volume group and adds logical volumes as they are needed and configured according to the image type and filesystem. After boot of the system the user has full control over the volume group and is free to change/resize/increase the group and the volumes inside.

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

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

With the optional systemdisk section you can set one or more top level directories into a separate volume. See Chapter 2, *KIWI Image Description* for a detailed explanation.

10 PXE Image—Thin Clients

Table of 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 separately 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.

10.1. Setting Up the Required Services

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

10.1.1. Atftp Server

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

1. Install the packages atftp and kiwi-pxeboot.
2. Edit the file `/etc/sysconfig/atftpd`. Set or modify the following variables:
   - `ATFTPD_OPTIONS="--daemon --no-multicast"`
   - `ATFTPD_DIRECTORY="/srv/tftpboot"`
3. Run `atftpd` by calling the command:

   ```
   rcatftpd start
   ```

10.1.2. DHCP Server

In contrast to the atftp server setup the following DHCP server setup can only serve as an example. Depending on your network structure, the IP addresses, ranges and domain settings needs to be adapted in order to allow the DHCP server to work within your network. If you already have a DHCP server running in your network, make sure that the filename and next-server information is provided by your server. The following steps describe how to setup a new DHCP server instance:

1. Install the package dhcp-server.

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

   ```
   option domain-name "example.org";
   option domain-name-servers 192.168.100.2;
   option broadcast-address 192.168.100.255;
   option routers 192.168.100.2;
   option subnet-mask 255.255.255.0;
   default-lease-time 600;
   max-lease-time 7200;
   ddns-update-style none; ddns-updates off;
   log-facility local7;

   subnet 192.168.100.0 netmask 255.255.255.0 {
      filename "pxelinux.0";
      next-server 192.168.100.2;
      range dynamic-bootp 192.168.100.5 192.168.100.20;
   }
   ```

3. Edit the file `/etc/sysconfig/dhcpd` and setup the network interface the server should listen on:

   ```
   DHCPD_INTERFACE="eth0"
   ```

4. Run the `dhcp` server by calling:

   ```
   rcdhcpd start
   ```

10.2. Building the suse-pxe-client Example

The example provided with KIWI is based on openSUSE 11.2 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-11.2
kiwi --prepare ./suse-pxe-client --root /tmp/mypxe
```

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

10.3. 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-11.2.i686-2.1.1.splash.gz \
  /srv/tftpboot/boot/initrd
cp initrd-netboot-suse-11.2.i686-2.1.1.kernel \
  /srv/tftpboot/boot/linux
```

3. Copy of the system image and md5 sum:

```
cp suse-11.2-pxe-client.i686-1.2.8 /srv/tftpboot/image
cp suse-11.2-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 globally, copy the KIWI generated file as config.default:

```
cp suse-11.2-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, insert the following information into the file `/srv/tftpboot/pxelinux.cfg/default`:

```
DEFAULT KIWI-Boot

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

LABEL Local-Boot
    localboot 0
```

6. Connect the client to the network and boot.

10.4. 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.

10.4.1. 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`.

10.4.2. 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-11.2-pxe-client.i686;1.2.8;192.168.100.2;4096
```

The following format is used:

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

`IMAGE`

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

srvip

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

bsize

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

compressed

Specifies if the image file on the TFTP server is compressed and handles it accordingly. To specify a compressed image download only the keyword "compressed" needs to be added. If compressed is not specified the standard download workflow is used. **Note:** The download will fail if you specify "compressed" and the image isn't compressed. It will also fail if you don't specify "compressed" but the image is compressed. The name of the compressed image has to contain the suffix .gz and needs to be compressed with the **gzip** tool. Using a compressed image will automatically *deactivate* the multicast download option of atftp.

`CONF`

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

Table 10.1. Configuration files synchronization possibilities

cfg file in CONF,VENDOR_CONF	cfg file in InstalledConfig-Files backup	action
hash_a	hash_a	nothing, keep
hash_a	hash_b	download from server
none	hash	download from server
hash	none	download from server
none	none	nothing, keep
present	not present	download from server (regardless hash)
not present	present	delete on client (regardles hash)

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

`PART`

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

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

`RAID`

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

```
                DISK=/dev/sda
                IMAGE=/dev/md1;LimeJeOS-openSUSE-11.3.i686;1.11.3;192.168.100.2;4096
                PART=5;S;x,2000;83;/
                RAID=1;/dev/sda;/dev/sdb
```

`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, this forces the configured image to be loaded from the server even if the image on the disk is up-to-date. The primary purpose of this setting is to aid debugging. The option is sensible only for disk based systems.

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

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

`KIWI_INITRD`
: Specifies the KIWI initrd to be used for local boot of the system. The variables value must be set to the name of the initrd file which is used via PXE network boot. If the standard tftp setup suggested with the kiwi-pxeboot package is used all initrd files resides in the `boot/` directory below the tftp server path `/var/lib/tftpboot`. Because the tftp server do a chroot into the tftp server path you need to specify the initrd file as the following example shows:

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

`UNIONFS_CONFIG`
: For netboot 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;\
  NBD-Write-Port-Number;/dev/NBD-Write-Device
```

The NBD-Device, NBD-Swap-Port-Number, NBD-Swap-Device, NBD-Write-Port-Number and NBD-Write-Device variables are optional. If the nbd root device is not set, the default values (`/dev/nb0`, port 2000) applies and if the nbd swap device is not set the default values (`/dev/nb1`, port 9210) applies. The swap space over the network using a network block device is only established if the client has less than 48 MB of RAM. The optional NBD-Write-Port-Number and NBD-Write-Device specifies a write COW location for the root filesystem. aufs is used as overlay filesystem in this case.

`AOEROOT`
: Mount the system image root filesystem remotely via AoE (ATA over Ethernet). This means there is a server which exports a block device representing the root directory of the system image via the AoE subsystem. The block device could be a partition of a real or a virtual disk. In order to use the AoE subsystem I recommend to install the aoetools and vblade packages from here first: http://download.opensuse.org/repositories/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. In case of a compressed read-only image with `aufs` or `clicfs`, the AOEROOT variable can also contain a device for the write actions:

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

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

`NFSROOT`
: Mount the system image root filesystem remotely via NFS (Network File System). This means there is a server which exports the root filesystem of the network client in such a way that the client can mount it read/write. In order to do that, the boot image must know the server IP address and the path name where the root directory exists on this server. The information must be provided as in the following example:

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

Optionally you can set a `UNIONFS_CONFIG` variable which defines an aufs based overlay NFS directory or device like:

```
UNIONFS_CONFIG=/tmp/kiwi-11.1-cow,nfs,aufs # write to NFS directory
UNIONFS_CONFIG=/dev/ram1,nfs,aufs # write to RAM
```

This way you can keep the original root tree clean from any modifications

`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.

10.4.3. 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.

10.4.4. RAM Only Image

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

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

  ```
  IMAGE=/dev/ram1;suse-11.2-pxe-client.i686;\
        1.2.8;192.168.100.2;4096
  ```

10.4.5. 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 `clicfs`. Below, find an example:

```
DISK=/dev/sda
PART=5;S;x,62;L;/,x;L;x,
IMAGE=/dev/sda2;suse-11.2-pxe-client.i386;\
      1.2.8;192.168.100.2;4096
UNIONFS_CONFIG=/dev/sda3,/dev/sda2,aufs
KIWI_INITRD=/boot/initrd
```

10.4.6. Split Image

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

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

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

```
   image="split" fsreadwrite="ext3" boot="netboot/suse-11.2"/>
```

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

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

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

```
IMAGE=/dev/sda2;suse-11.2-pxe-client.i686;\
      1.2.8;192.168.100.2;4096,\
      /dev/sda3;suse-11.2-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
```

10.4.7. 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
```

10.4.8. 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
```

10.4.9. Root Tree Over AoE

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

- Use the **vbladed** command to bind a block device to an ethernet interface. The block device can be a disk partition or a loop device (losetup) but not a directory like with NBD.

- Sample `config.MAC`:

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

  This would require the following command to be called first:

  ```
  vbladed 0 1 eth0 blockdevice
  ```

10.5. Hardware Grouping

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

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

10.5.1. The Group Configuration File

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

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

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

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

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

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

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

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

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

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

- KIWI_GROUP

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

  - test1
  - test2
  - test3

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

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

- <GROUP_NAME>_KIWI_MAC_LIST

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

  - test1_KIWI_MAC_LIST
  - test2_KIWI_MAC_LIST
  - test3_KIWI_MAC_LIST

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

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

10.5.2. The Group Details File

In addition to the config.group file, each defined group will require a config.<GROUP_NAME> file. This file is exactly like a standard KIWI config.<MAC> file, but is assigned to a group of hosts rather than a single unit. If we continue with the example we used in the previous section, we would need the following files:

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

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

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

```
DISK=/dev/sda
PART=5;S;x,x;L;/
IMAGE=/dev/sda2;suse-11.2-pxe-client.i686;1.2.8;192.168.100.2;4096
CONF=CONFIGURATIONS/xorg.conf.test1;/etc/X11/xorg.conf;192.168.100.2;4096,CONFIGURATIONS/syslog
```

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

```
DISK=/dev/sda
PART=5;S;x,x;L;/
IMAGE=/dev/sda2;suse-11.2-pxe-client.i686;1.2.8;192.168.100.2;4096
CONF=CONFIGURATIONS/xorg.conf.test2;/etc/X11/xorg.conf;192.168.100.2;4096,CONFIGURATIONS/syslog
```

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

10.5.3. Using Hardware Mapping to Provide Overrides

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

10.5.3.1. The Hardware Mapping Elements

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

```
HARDWARE_MAP="vendor_name_model"

vendor_name_model_HARDWARE_MAP="00:00:00:11:11:11"
```

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

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

The definitions for the above parameters are as follows:

- HARDWARE_MAP

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

- <HARDWARE_MAP_NAME>_HARDWARE_MAP

  This element behaves exactly like the <GROUP_NAME>_KIWI_MAC_LIST element defined above, in that it lists all MAC addreses that need to be linked to a hardware map. Any host defined within the list will receive configuration files that have been specifically defined in a hardware_config.<hardware_map> file, in addition to any files defined within a CONF element.

10.5.3.2. The Hardware Mapping Details File

Once the hardware map has been defined, the last step is to ensure configuration specific elements are linked to the host(s) in question. This is done by creating a new hardware_config.<hardware_map> file. The contents of the file is quite simple, and contains only one element called VENDOR_CONF, as the following example shows:

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

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

- The host is assigned to the current hardware map

- The file is defined within the CONF and VENDOR_CONF elements

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

10.5.3.3. A Complete Example

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

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

```
[root@test01 KIWI]# cd /srv/tftpboot/KIWI
[root@test01 KIWI]# ls -la
total 24
drwxr-xr-x  2 root root 4096 Nov  9 04:44 .
drwxr-x---  9 root root 4096 Nov  9 04:38 ..
-rw-r--r--  1 root root  497 Nov  9 04:43 config.example1
-rw-r--r--  1 root root  228 Nov  9 04:41 config.group
-rw-r--r--  1 root root  133 Nov  9 04:44 hardware_config.maxterm_3500
```

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

```
[root@test01 KIWI]# cat config.group
KIWI_GROUP="example1"

example1_KIWI_MAC_LIST="00:00:00:00:00:01 00:00:00:00:00:02 00:00:00:00:00:03 00:00:00:00:00:04
                        00:00:00:00:00:07 00:00:00:00:00:08 00:00:00:00:00:09 00:00:00:00:00:0A"
```

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

```
[root@test01 KIWI]# cat config.example1
KIWI_INITRD=/boot/initrd
KIWI_KERNEL=/boot/linux
DISK=/dev/sda
PART=5;S;x,769;L;/,x;L;x,
IMAGE=/dev/sda2;exmaple-kiosk-opensuse-11.3-pxe-client.i686;0.0.1;192.168.1.2;4096
UNIONFS_CONFIG=/dev/sda3,/dev/sda2,clicfs
CONF=prefs.js;/home/kioskuser/.mozilla/firefox/07xvl1ty.default/prefs.js;192.168.1.2;4096,xorg.c
RELOAD_IMAGE=yes
RELOAD_CONFIG=yes

HARDWARE_MAP="maxterm_3500"
maxterm_3500_HARDWARE_MAP="00:00:00:00:00:02 00:00:00:00:00:03 00:00:00:00:00:04"
```

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

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

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

```
[root@test01 KIWI]# cat hardware_config.maxterm_3500
VENDOR_CONF=xorg.conf.maxterm_3500;/etc/X11/xorg.conf;192.168.1.2;4096,someconfig.cfg;/etc/sysc
```

When the VENDOR_CONF defintition is used, we are telling KIWI that all files defined within this element, are specific to the hardware map they are linked to. As a result, any files listed here will be transferred to a host if, and only if, the host has been linked to the hardware map via the maxterm_3500_HARDWARE_MAP element. In our example the only systems that will receive the xorg.conf.maxterm_3500 file will be the three maxterms we linked to the hardware map itself.

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

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

11 OEM Image—Preload Systems

Table of Contents

An OEM image is a virtual disk image representing all partitions and bootloader information in the same fashion it exists on a physical disk. The image format matches the format of the VMX image type. All flavors discussed previously for the VMX image type apply to the OEM image type.

The basic idea behind an oem image is to provide the virtual disk data for OEM vendors to support easy deployment of the system to physical storage media. The deployment can be performed from any OS including Windows as long as a tool to dump data onto a disk device exists and is used. The oem image type may also be used to deploy an image on a USB stick. A USB stick is simply a removable physical storage device.

11.1. Building the suse-oem-preload Example

The OEM example provided with kiwi is based on recent openSUSE releases, one example configuration per release, and includes the default and x11 patterns. The image type is a split type utilizing the distributions default filesystem format for the read-write partition and the `squashfs` filesystem for the read-only partition. Using the additional `installiso` attribute creates an installable ISO image. When booting from the ISO image the OEM disk image will be deployed to the storage media on the booting machine (after confirmation by the user).

The commands provided below use the openSUSE 11.2 based example built for the x86 architecture.

```
cd /usr/share/doc/packages/kiwi/examples
cd suse-11.2
kiwi --prepare ./suse-oem-preload --root /tmp/myoem
```

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

11.2. Using the Image

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

```
cd /tmp/myoem-result
qemu suse-11.2-oem-preload.i686-1.1.2.raw -m 512
```

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

```
cd /tmp/myoem-result
dd if=suse-11.2-oem-preload.i686-1.1.2.raw of=/dev/device bs=32k
```

Note, when testing an oem image using the virtual disk image, i.e. the `.raw` file, the geometry of the disk image is not changed and therefore retains the disk geometry of the host system. This implies that the re-partitioning performed for a physical disk install during the oem boot workflow will be skipped.

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

```
cd /tmp/myoem-result
qemu-img create /tmp/mydisk 20G
qemu -hda /tmp/mydisk -cdrom suse-11.2-oem-preload.i686-1.1.2.iso -boot d
```

11.3. Flavours

As indicated above the use of the `installiso` and `installstick` attributes for the oem image supports the creation of an installation image. The installation image can be created in two formats, one suitable for CD/DVD media and a second suitable for a USB stick. The self installing image deploys the oem image onto the selected storage device. The installation process is a simple image dump using the **dd** command. During this process the target system remains in terminal mode. The following configuration snippets show the use of the `installiso` and `installstick` attributes to create the ISO or USB installation image format respectively.

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

  Creates a `.iso` file which can be burned onto a CD or a DVD. This represents an installation CD/DVD

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

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

11.3.1. Specializing the OEM install process

It is possible to specialize the OEM install process by providing shell scripts with the following names in the `root/kiwi-hooks` directory in the configuration tree as overlay files.

- `preHWdetect.sh` This script is executed prior to the hardware scan on the target machine.
- `preImageDump.sh` This script is executed immediately prior to the OEM image dump onto the target storage device.

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

11.3.2. Influencing the OEM Partitioning

By default the oemboot process will create/modify a swap, /home and / partition. It is possible to influence the behavior with the oem-* elements. See Chapter 2, *KIWI Image Description* for details.

11.3.3. LVM Support

KIWI also provides support for LVM (Logical Volume Management). In this mode the disk partition table will include one lvm partition and one standard ext2 boot partition. KIWI creates the kiwiVG volume group, unless the lvmgroup attribute has been set, and adds logical volumes to the group based on the configuration given by the systemdisk block for this type. The filesystem for the volume group is determined by the filesystem attribute of the type element. After booting the system the user has full control over the volume group and is free to change (resize/increase) the group and the volumes inside. Support for LVM has been added for all disk based image types. This includes the vmx, oem and usb image types. In order to use LVM for the oem type just add the --lvm command line option when executing the create step or add the attribute lvm="true" to of the type element in your config.xml file.

```
kiwi --create /tmp/myoem --type oem -d /tmp/myoem-result --lvm
```

With the optional systemdisk section you can specify to have one or more top level directories in a separate volume. See Chapter 2, *KIWI Image Description* for a detailed explanation.

11.3.4. Partition Based Installation

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

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

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

- The bootloader will be setup to boot the installed system. There is no multiboot setup. The user is expected to implement the setup of a multiboot bootloader.

- The oem options for system, swap and home doesn't have any effect if the installation happens in predefined partitions.

- There is no support for remote (PXE) OEM installation because kiwi has to loop mount the disk image in order to access the partitions which can't be done remotely.

- The raw disk image is stored uncompressed on the install media. This is because KIWI needs to loop mount the disk image which it can't do if the file is only available as compressed version. This means the install media in this mode will be approximately double the size of a standard install media.

12 Xen Image—Paravirtual Systems

Table of Contents

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

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

A Xen image is a virtual disk like a vmx but with the xen kernel installed. In order to run it a Xen dom0 server needs to run. Xen images in KIWI makes use of the PVGrub method supported by current Xen versions. Xen extracts the kernel and initrd from the virtual disk as well as the grub configuration and displays the menu which allows emulation of the Grub console

12.1. Building the suse-xen-guest Example

The latest example provided with KIWI is based on openSUSE 11.3 and includes the base pattern.

```
cd /usr/share/doc/packages/kiwi/examples cd suse-11.2
kiwi --prepare ./suse-xen-guest --root /tmp/myxen
```

```
kiwi --create /tmp/myxen --type vmx -d /tmp/myxen-result
```

12.2. Using the Image

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

```
xm create -c /tmp/myxen-result/  suse-11.2-xen-guest.i686-1.1.2.xenconfig
```

12.3. 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 with a profile.

```
<packages type="image" profiles="xenFlavour">
  <package name="kernel-xen" replaces="kernel-ec2"/>
</packages>
<type ....>
  <machine memory="512" domain="domU">
    <vmdisk ... device="/dev/xvda"/>
  </machine>
</type>
```

If this information is present KIWI will create a Xen domain U configuration with 512 MB of RAM and expects the disk at `/dev/xvda`. Additional information to setup the Xen guest machine properties are explained in the machine section. The KIWI Xen domain U configuration is stored in the file `/tmp/myxen-result/suse-11.2-xen-guest.i686-1.1.2.xenconfig`.

13 EC2 Image—Amazon Elastic Compute Cloud

Table of Contents

The Amazon Elastic Compute Cloud™ (Amazon EC2) web service provides you with the ability to execute arbitrary applications in the Amazon 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 format. 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://aws.amazon.com/documentation/ec2/ software, including the operating system and associated configuration settings, applications, libraries, etc.

2. Upload this AMI to the Amazon S3 (Amazon Simple Storage Service) service.

3. Register your AMI with Amazon EC2.

4. Use the AMI ID created during registration to run your EC2 guest.

KIWI creates an AMI using tools provided by Amazon. Due to licensing issues it is not possible to distribute the Amazon tools with your Linux distribution or KIWI. Amazon provides two sets of tools, AMI tools and API tools. The tools can be downloaded from the Amazon web site at http://aws.amazon.com/developertools/368 and http://aws.amazon.com/developertools/351 respectively. The documentation for the command line tools may be accessed on the Amazon web-site as well at http://docs.amazonwebservices.com/AWSEC2/latest/CommandLineReference. While KIWI only needs to interact with the AMI tools to create the AMI you will also need to install the API tools to interact with EC2. You must at least install the AMI tools to create an EC2 image with KIWI.

When working with the Amazon tools it is useful to set the EC2_HOME, EC2_PRIVATE_KEY, and EC2_CERT environment variables. Setting EC2_PRIVATE_KEY, and EC2_CERT allows you to forego specification of the `--private-key` and `--cert` with every command. The EC@_HOME environment variable is used by the tools to find required libraries.

- EC2_HOME

Location of the bin and lib directories installed by the Amazon tools. A good location for the tools on your system is `/usr/local`.

- EC2_PRIVATE_KEY

  Path to your private key file (including the filename). For example `/home/USERNAME/AWS/keys/pk-....pem`

- EC2_CERT

  Path to your certificate file (including the filename). For example `/home/USERNAME/AWS/keys/cert-....pem`

Please note that your account will be billed by Amazon at the published rate for any computing resources you consume in EC2. This includes but is not limited to, running instances, storing data (your image) on S3 or EBS, and network traffic.

13.1. Building the suse-ec2-guest Example

The example provided with KIWI uses openSUSE as the base distribution and includes the base pattern plus the vim editor.

Before building the EC2 example assure that the EC2 command line tools are installed and work as expected. Executing the **ec2-describe-images** `-a` command should return a list of all available instances in EC2. Further, you need to edit the `config.xml` file of the example and enter your EC2 account information. Replace the values for `ec2accountnr`, `ec2privatekeyfile`, and `ec2certfile`. Note that the account number is displayed with dsahes ('-") on the Amazon web site, but the entry in `config.xml` is expected to be numeric only. The private key file is the private file you downloaded when you created the certificate. It is recommended to create a copy of the example `/usr/share/doc/packages/kiwi/examples/suse-`*`Version`*`/suse-ec2-guest`.

You are now ready to build the EC2 example. Lets assume you copied the example configuration directory to /tmp prior to modifying the `config.xml` file.

```
kiwi --prepare /tmp/suse-ec2-guest --root /tmp/myec2
```

```
kiwi --create /tmp/myec2 -d /tmp/myec2-result -y
```

13.2. Using the Image

The generated image needs to be transfered to Amazon. The upload of the AMI is performed using the **ec2-upload-bundle** command line tool. This tool is part of the API tools provided by Amazon. Upload the AMI as follows, replacing *`AWS_Key_ID`* and *`AWS_secret_Key_ID`* with your Amazon key information. Also you may want to choose a different name for your bucket than `myImages`. If the bucket does not exist in S3 it will be created.

```
ec2-upload-bundle -b myImages \
```

```
     -a AWS_Key_ID -s AWS_secret_Key_ID -m /tmp/myec2/ \
   suse-11.3-ec2-guest.i686-1.1.2.ec2/ \
   suse-11.3-ec2-guest.i686-1.1.2.ami.manifest.xml
```

After the upload process is complete register your image with the EC2 infrastructure using the **ec2-register** command as shown below. The result of the registration process is an AMI ID returned on the command line in the form “ami-” followed by a random key sequence.

```
ec2-register myImages/ suse-11.3-ec2-guest.i686-1.1.2.ami.manifest.xml
```

Once the image is registered via the **ec2-register** command you can use the AWS management console in your browser to launch an instance of your image. It is alos possible to launch an instance from the command line. This process is demonstrated below. The **ec2-describe-images** command will provide you with an overview of all your registered images.

Prior to launching your instance from the command line you will need to have a keypair. In the Web UI the launch wizard will guide you through this process. The command line interface if the **ec2-add-keypair** command. The public/private keypair is used to allow you to have access to your instance. If you do not already have a keypair for interacting with EC2 images you can generate a keypair as shown below. The *gsg-keypair* name is used in this example, but you can choose any name you like. As you will need to use the key quite often you probably want a name that is easy to remember and not too terribly long to type.

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

Save the private key returned by the command in a local file. 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 existing keypairs use the **ec2-describe-keypairs** command.

The Amazon EC2 infrastructure uses PVGrub (Para-Virtual Grub) to boot instances of an image. Basically Amazon provides a minimal Linux kernel that has parts of Grub built in and is specialized to EC2. PVGrub at some point looks for and reads the `/boot/grub/menu.lst` file in your image and then boots the specified kernel. Each availability zone in EC2 has it's own independent copy of this boot mechanism. Every boot kernel has an ID starting with the TLA (Three Letter Acronym) *aki*. The Amazon Kernel Image IDs table below provides guidelines for the selection of the AKI ID.

Table 13.1. Amazon Kernel Image IDs

Zone	AKI	Arch	Name
US-East	aki-407d9529	x86	ec2-public-images/pv-grub-hd0-V1.01-i386.gz.manifest.xml
US-East	aki-427d952b	x86-64	ec2-public-images/pv-grub-hd0-V1.01-x86_64.gz.manifest.xml
US-West	aki-99af01dc	x86	ec2-public-images-us-west-1/pv-grub-hd0-V1.01-x86_64.gz.manifest.xml
US-West	aki-99af01dc	x86-64	ec2-public-images-us-west-1/pv-grub-hd0-V1.01-x86_64.gz.manifest.xml
EU-West	aki-4deec439	x86	ec2-public-images-eu/pv-grub-hd0-V1.01-x86_64.gz.manifest.xml
EU-West	aki-4feec43b	x86-64	ec2-public-images-eu/pv-grub-hd0-V1.01-x86_64.gz.manifest.xml

Zone	AKI	Arch	Name
AP-SouthEast	aki-13d5aa41	x86	`ec2-public-images-ap-southeast-1/pv-grub-hd0-V1.01-x86_64.gz.manifest.xml`
AP-SouthEast	aki-11d5aa43	x86-64	`ec2-public-images-ap-southeast-1/pv-grub-hd0-V1.01-x86_64.gz.manifest.xml`

The information in the table above was extracted from the Amazon documentation found at: http://ec2-downloads.s3.amazonaws.com/user_specified_kernels.pdf. Kiwi images created for EC2 do not contain a partition table. Therefore, PVGrub images designated by *h0* in the manifest name need to be used with the images created by Kiwi.

Kiwi can embed the appropriate PVGrub Kernel ID in the EC2 manifest file if you specify the optional `ec2region` as part of the `ec2config` block in your `config.xml`. If you specify multiple regions Kiwi generates AMIs for each region specified. Supported values for the `ec2region` element are *AP-Singapore*, *EU-West*, *US-East*, and *US-West*.

With the appropriate boot kernel selected you can start your image. In this example we use the x86 Kernel in the US-East availability zone.

```
ec2-run-instances ami-... \
   --kernel aki-407d9529 \
   -k gsg-keypair
```

To check the state of your instance(s) use the command **ec2-describe-instances**.

Once the instance state is shown as *running* you can login into it using ssh as follows:

```
ssh -i PATH_TO_PRIVATE_KEY \
      root@YOUR_INSTANCE_ID
```

If you are unable to log in, it is most likely that the security setting for the instance is blocking the ssh access. To remedy this problem open port 22 using the **ec2-authorize** command as shown below.

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

A few more remarks about EC2 are in order. As you have seen at the end of our example it is important that you include ssh in your image and that you start the ssh daemon in your image. Starting the sshd process is accomplished by adding *suseInsertService sshd* in `config.sh`.

Earlier in the example you generated a keypair that was then used to allow you to log into your running instance. The key does need to get injected into the instance when you launch it. This key injection happens through an init script `/etc/init.d/amazon`. This script is provided by the `suse-ami-tools` package. Therefore, you need to include this package in the `packages` section of your `config.xml` file. You also need to make sure the `amazon` init script gets executed during boot by using *suseInsertService amazon* in your `config.sh` file.

The key injection mechanism needs to access the network. Therefore, you must configure the network of your instance. The network interface of the guest is always eth0 and it needs to be configured to use DHCP. See the overlay file tree of the example for configuration details.

Last but not least some information about the storage options. Amazon offers S3 http://aws.amazon.com/s3/ backed images (this is what we have worked with in this example) and EBS (Elastic Block Store) http://aws.amazon.com/ebs/ backed images. At the time of this writing it is not possible to use an image format created by Kiwi with the Amazon tools directly to create an EBS backed image. It is possible to create an EBS volume from a KIWI raw image, but the process is cumbersome and requires a running Linux instance in EC2. It is anticipated that at some point in the future the Amazon team will address this problem and that the **ec2-import-instance** command in conjunction with the **ec2-upload-disk-image** command will be able to create EBS volumes from Linux based disk images. At this point you will be able to use the raw image created by KIWI to create an EBS backed image.

A KIWI Man Pages

Table of Contents

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

kiwi

kiwi — Creating Operating System Images

Synopsis

`kiwi` { -l | --list }

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

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

Basics

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

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

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

Image Preparation and Creation

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

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

Image Upgrade

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

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

System to Image Migration

The migration module allows you to migrate your currently running system into an image description. The module will check for files not managed by a package manager and also inspects your system for package pattern and file consistency according to the currently active repositories. The system requires the zypper backend in order to work properly.

The migration process creates a cache file so that subsequent calls of the migration runs much faster. Please have in mind that if your system has changed (files created/deleted, etc.) the cache file might not be worth to become reused. In this case you should remove the cache first and start from scratch. The option `--nofiles` will prevent the system from searching for unpackaged and packaged but modified files The option `--notemplate` will prevent the creation of the image description files which are needed if you want to use KIWI to create a clone image from the result of the migration. With the options `--exclude` and `--skip` you can tell the system to ignore specific directories and/or packages. This makes sense if you know before that some data is not worth to become migrated or can be restored easily later inside the cloned image like software repositories.

The migration process will always place it's result into the `/tmp/$OptionValueOf-m` directory. The reason for this is because `/tmp` is always excluded from the migration operation and therefore we can safely place new files there without influencing the migration itself. You should have at least 50 MB free space for the cache file and the image description all the rest are just hard links.

As one result a HTML based report file is created which contains important information about the system. You are free to ignore that information but with the risk that the migrated image does not represent the same system which is running at the moment. The less issues left in the report the better is the result. In most cases a manual fine tuning is required. This includes the repository selection and the unmanaged files along with the configuration details of your currently running operating system. You should understand the module as a helper to migrate running servers into images. The implementation is still under construction so expect better migration results in future releases :)

`kiwi` { -m | --migrate } *name* [--exclude *`directory`*...] [--skip *package*...] [--nofiles] [--notemplate]

Image Postprocessing Modes

The KIWI post-processing modes are used for special image deployment tasks, like installing the image on a USB stick. So to say they are the third step after preparation and creation. KIWI calls the postprocessing modules automatically according to the specified output image type and attributes but it's also possible to call them manually.

`kiwi` --bootstick *`initrd`* [--bootstick-system *`systemImage`*] [--bootstick-device *`device`*]

`kiwi` --bootvm *`initrd`* --bootvm-system *`systemImage`* [--bootvm-disksize *`size`*] kiwi --bootcd *initrd*

kiwi --installcd *initrd* --installcd-system *vmx-system-image*

kiwi --installstick *initrd* --installstick-system *vmx-system-image*

Image format conversion

The KIWI format conversion is useful to perform the creation of another image output format like vmdk for VMware or ovf the open virtual machine format. Along with the conversion KIWI also creates the virtual machine configuration according to the format if there is a machine section specified in the XML description

kiwi --convert *systemImage* [--format *vmdk|ovf|qcow2*]

Helper Tools

The helper tools provide optional functions like creating a crypted password string for the users section of the `config.xml` file as well as signing the image description with an md5sum hash and adding splash data to the boot image used by the bootloader.

kiwi --createpassword

kiwi --createhash *image-path*

kiwi { -i | --info } *ImagePath* {--select *repo-patterns|patterns|types|sources|size|profiles|packages* }

kiwi --setup-splash *initrd*

The following list describes the helper tools more detailed

`[--createpassword]`
: Create a crypted password hash and prints it on the console. The user can use the string as value for the pwd attribute in the XML users section

`[--createhash` *image-path* `]`
: Sign your image description with a md5sum. The result is written to a file named `.checksum.md` and is checked if KIWI creates an image from this description.

`[ -i | --info` *image-path* `--select` *selection* `]`
: List general information about the image description. So far you can get information about the available patterns in the configured repositories with *repo-patterns*, a list of used patterns for this image with *patterns*, a list of supported image types with *types*, a list of source URLs with *sources*, an estimation about the install size and the size of the packages marked as to be deleted with *size*, a list of profiles with *profiles*, and a list of solved packages to become installed with *packages*.

`[--setup-splash` *initrd* `]`
: Create splash screen from the data inside the initrd and re-create the initrd with the splash screen attached to the initrd cpio archive. This enables the kernel to load the splash screen at boot time. If splashy is used only a link to the original initrd will be created

Global Options

[--add-profile *profile-name*]
: Use the specified profile. A profile is a part of the XML image description and therefore can enhance each section with additional information. For example adding packages.

[--set-repo *URL*]
: Set/Overwrite repo URL for the first listed repo. The change is temporary and will not be written to the XML file.

[--set-repotype *type*]
: Set/Overwrite repo type for the first listed repo. The supported repo types depends on the packagemanager. Commonly supported are rpm-md, rpm-dir and yast2. The change is temporary and will not be written to the XML file.

[--set-repoalias *name*]
: Set/Overwrite alias name for the first listed repo. Alias names are optional free form text. If not set the source attribute value is used and builds the alias name by replacing each "/" with a "_". An alias name should be set if the source argument doesn't really explain what this repository contains. The change is temporary and will not be written to the XML file.

[--set-repoprio *number*]
: Set/Overwrite priority for the first listed repo. Works with the smart packagemanager only. The Channel priority assigned to all packages available in this channel (0 if not set). If the exact same package is available in more than one channel, the highest priority is used.

[--add-repo *URL*, --add-repotype *type* --add-repoalias *name* --add-repoprio *number*]
: Add the given repository and type for this run of an image prepare or upgrade process. Multiple `--add-repo`/`--add-repotype` options are possible. The change will not be written to the `config.xml` file

[--ignore-repos]
: Ignore all repositories specified so far, in XML or elsewhere. This option should be used in conjunction with subsequent calls to `--add-repo` to specify repositories at the commandline that override previous specifications.

[--logfile *Filename* | terminal]
: Write to the log file *Filename* instead of the terminal.

[--gzip-cmd *cmd*]
: Specify an alternate command to run when compressing boot and system images. Command must accept **gzip** options.

[--log-port *PortNumber*]
: Set the log server port. By default port 9000 is used. If multiple KIWI processes runs on one system it's recommended to set the logging port per process.

[--package-manager *smart|zypper*]
: Set the package manager to use for this image. If set it will temporarily overwrite the value set in the xml description.

[-A | --target-arch *i586|x86_64|armv5tel|ppc*]
: Set a special target-architecture. This overrides the used architecture for the image-packages in `zypp.conf`. When used with smart this option doesn't have any effect.

[--debug]
: Prints a stack trace in case of internal errors

[--verbose *1|2|3*]
: Controls the verbosity level for the instsource module

Image Preparation Options

[-r | --root *RootPath*]
: Set up the physical extend, chroot system below the given root-path path. If no `--root` option is given, KIWI will search for the attribute defaultroot in `config.xml`. If no root directory is known, a **mktemp** directory will be created and used as root directory.

[--force-new-root]
: Force creation of new root directory. If the directory already exists, it is deleted.

Image Upgrade/Preparation Options

[--cache *directory*]
: When specifying a cache directory, KIWI will create a cache each for patterns and packages and re-use them, if possible, for subsequent root tree preparations of this and/or other images

[--add-package *package*]
: Add the given package name to the list of image packages multiple `--add-package` options are possible. The change will not be written to the XML description.

[--add-pattern *name*]
: Add the given pattern name to the list of image packages multiple `--add-pattern` options are possible. The change will not be written to the xml description. Patterns can be handled by SUSE based repositories only.

[--del-package *package*]
: Removes the given package by adding it the list of packages to become removed. The change will not be written to the xml description.

Image Creation Options

[-d | --destdir *DestinationPath*]
: Specify destination directory to store the image file(s) If not specified, KIWI will try to find the attribute *defaultdestination* which can be specified in the *preferences* section of the `config.xml` file. If it exists its value is used as destination directory. If no destination information can be found, an error occurs.

[-t | --type *Imagetype*]
: Specify the output image type to use for this image. Each type is described in a *type* section of the preferences section. At least one type has to be specified in the `config.xml` description. By default, the types specifying the *primary* attribute will be used. If there is no primary attribute set, the first type section of the preferences section is the primary type. The types are only evaluated when KIWI runs the `--create` step. With the option `--type` one can distinguish between the types stored in `config.xml`

[-s | --strip]
: Strip shared objects and executables - only makes sense in combination with `--create`

[--prebuiltbootimage *Directory*]
: Search in *Directory* for pre-built boot images.

[--isocheck]
: in case of an iso image the checkmedia program generates a md5sum into the ISO header. If the --isocheck option is specified a new boot menu entry will be generated which allows to check this media

[--lvm]
: Use the logical volume manager to control the disk. The partition table will include one lvm partition and one standard ext2 boot partition. Use of this option makes sense for the create step only and also only for the image types: vmx, oem, and usb

[--fs-blocksize *number*]
: When calling KIWI in creation mode this option will set the block size in bytes. For ISO images with the old style ramdisk setup a blocksize of 4096 bytes is required

[--fs-journalsize *number*]
: When calling KIWI in creation mode this option will set the journal size in mega bytes for ext[23] based filesystems and in blocks if the reiser filesystem is used

[--fs-inodesize *number*]
: When calling KIWI in creation mode this option will set the inode size in bytes. This option has no effect if the reiser filesystem is used

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

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

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

[--partitioner *parted|fdasd*]
: Select the tool to create partition tables. Supported are parted and fdasd (s390). By default parted is used

[--check-kernel]
: Activates check for matching kernels between boot and system image. The kernel check also tries to fix the boot image if no matching kernel was found.

For More Information

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

http://en.opensuse.org/Portal:KIWI
: KIWI wiki

config.xml
: The configuration XML file that contains every aspect for the image creation.

file:///usr/share/doc/packages/kiwi/kiwi.pdf
: The system design document which describes some details about the building process.

file:///usr/share/doc/packages/kiwi/schema/kiwi.xsd.html
: The KIWI RELAX NG XML Schema documentation.

file:///usr/share/doc/packages/kiwi/schema/test.xsd.html
: The KIWI RELAX NG XML Schema documentation.

kiwi::config.sh

KIWI::config.sh — Configuration File for KIWI image description

Description

The KIWI image description allows to have an optional `config.sh` script in place. This script should be designed to take over control of adding the image operating system configuration. Configuration in that sense means stuff like activating services, creating configuration files, prepare an environment for a firstboot workflow, etc. What you shouldn't do in `config.sh` is breaking your systems integrity by for example removing packages or pieces of software. Something like that can be done in `images.sh`. The `config.sh` script is called *after* the user and groups have been set up. If there are SUSE Linux related YaST XML information, these are validated before `config.sh` is called too. If you exit config.sh with an exit code != 0 kiwi will exit with an error too.

Example A.1. Template for config.sh

```
#======================================
# Functions...
#--------------------------------------
test -f /.kconfig && . /.kconfig
test -f /.profile && . /.profile

#======================================
# Greeting...
#--------------------------------------
echo "Configure image: [$kiwi_iname]..."

#======================================
# Call configuration code/functions
#--------------------------------------
...

#======================================
# Exit safely
#--------------------------------------
exit
```

Common functions

The `.kconfig` file allows to make use of a common set of functions. Functions specific to SUSE Linux specific begin with the name *suse*. Functions applicable to all linux systems starts with the name *base*. The following list describes the functions available inside the `config.sh` script.

[baseCleanMount]
: Umount the system filesystems `/proc`, `/dev/pts`, and `/sys`.

[baseDisableCtrlAltDel]
: Disable the **Ctrl-Alt-Del** key sequence setting in `/etc/inittab`

[baseGetPackagesForDeletion]
: Return the name(s) of packages which will be deleted

[baseGetProfilesUsed]
: Return the name(s) of profiles used to build this image

[baseSetRunlevel {value}]
Set the default run level

[baseSetupBoot]
Set up the linuxrc as init

[baseSetupBusyBox {-f}]
activates busybox if installed for all links from the `busybox/busybox.links` file—you can choose custom apps to be forced into busybox with the `-f` option as first parameter, for example:

```
baseSetupBusyBox -f /bin/zcat /bin/vi
```

[baseSetupInPlaceGITRepository]
Create an in place git repository of the root directory. This process may take some time and you may expect problems with binary data handling

[baseSetupInPlaceSVNRepository {path_list}]
Create an in place subversion repository for the specified directories. A standard call could look like this baseSetupInPlaceSVNRepository `/etc`, `/srv`, and `/var/log`

[baseSetupPlainTextGITRepository]
Create an in place git repository of the root directory containing all plain/text files.

[baseSetupUserPermissions]
Search all home directories of all users listed in `/etc/passwd` and change the ownership of all files to belong to the correct user and group.

[baseStripAndKeep {list of info-files to keep}]
helper function for strip* functions read stdin lines of files to check for removing params: files which should be keep

[baseStripDocs {list of docu names to keep}]
remove all documentation, except one given as parameter

[baseStripInfos {list of info-files to keep}]
remove all info files, except one given as parameter

[baseStripLocales {list of locales}]
remove all locales, except one given as parameter

[baseStripMans {list of manpages to keep}]
remove all manual pages, except one given as parameter example: baseStripMans more less

[baseStripRPM]
remove rpms defined in `config.xml` under image=delete section

[baseStripTools {list of toolpath} {list of tools}]
helper function for suseStripInitrd function params: toolpath, tools

[baseStripUnusedLibs]
remove libraries which are not directly linked against applications in the bin directories

[baseUpdateSysConfig {filename} {variable} {value}]
update sysconfig variable contents

[Debug {message}]
Helper function to print a message if the variable `DEBUG` is set to 1

[Echo {echo commandline}]
Helper function to print a message to the controlling terminal

[Rm {list of files}]
Helper function to delete files and announce it to log

[Rpm {rpm commandline}]
Helper function to the RPM function and announce it to log

[suseActivateDefaultServices]
Call all postin scriptlets which among other things activates all required default services using suseInsertService

[suseActivateServices]
Check all services in `/etc/init.d/` and activate them by calling suseInsertService

[suseCloneRunlevel {runlevel}]
Clone the given runlevel to work in the same way as the default runlevel 3.

[suseConfig]
Setup keytable language and timezone if specified in `config.xml` and call SuSEconfig afterwards

[suseInsertService {servicename}]
Recursively insert a service. If there is a service required for this service it will be inserted first. The suse insserv program is used here

[suseRemoveService {servicename}]
Remove a service and its dependent services using the suse insserv program

[suseService {servicename} {on|off}]
Activate/Deactivate a service by using the chkconfig program The function requires the service name and the value on or off as parameters

[suseServiceDefaultOn]
Activates the following services to be on by default using the chkconfig program: boot.rootfsck boot.cleanup boot.localfs boot.localnet boot.clock policykitd dbus consolekit haldaemon network atd syslog cron kbd

[suseSetupProductInformation]
This function will use zypper to search for the installed product and install all product specific packages. This function only makes sense if zypper is used as packagemanager

[suseStripPackager {-a}]
Remove smart o zypper packages and db files Also remove rpm package and db if `-a` given

Profile environment variables

The .profile environment file contains a specific set of variables which are listed below. Some of the functions above makes use of the variables.

[$kiwi_compressed]
The value of the compressed attribute set in the type element in `config.xml`

[$kiwi_delete]
: A list of all packages which are part of the packages section with `type="delete"` in `config.xml`

[$kiwi_drivers]
: A comma separated list of the driver entries as listed in the drivers section of the `config.xml`. Similar variables exists for the `usbdrivers` and `scsidrivers` sections

[$kiwi_iname]
: The name of the image as listed in `config.xml`

[$kiwi_iversion]
: The image version string major.minor.release

[$kiwi_keytable]
: The contents of the keytable setup as done in `config.xml`

[$kiwi_language]
: The contents of the locale setup as done in `config.xml`

[$kiwi_profiles]
: A list of profiles used to build this image

[$kiwi_size]
: The predefined size value for this image. This is not the computed size but only the optional size value of the preferences section in `config.xml`

[$kiwi_timezone]
: The contents of the timezone setup as done in `config.xml`

[$kiwi_type]
: The basic image type. Can be a simply filesystem image type of ext2, ext3, reiserfs, squashfs, cpio, or one of the following complex image types: iso, split, usb, vmx, oem, xen, or pxe.

kiwi::images.sh

KIWI::images.sh — Configuration File for KIWI image description

Description

The KIWI image description allows to have an optional `images.sh` script in place. This script is called at the beginning of the KIWI create step. It is allowed to remove software there to shrink down the size of the image. Most often `images.sh` is used for boot images because they needs to be small. As `images.sh` is called in the create step you should be aware to design the script in a way that it can be called multiple times without shooting itself into its knee. As KIWI allows to create different image types from one previously prepared tree one needs to take into account that `images.sh` can be called more than one time. If you exit `images.sh` with an exit code != 0 KIWI will exit with an error too.

Example A.2. Template for images.sh

```
#======================================
# Functions...
#--------------------------------------
test -f /.kconfig && . /.kconfig
test -f /.profile && . /.profile

#======================================
# Greeting...
#--------------------------------------
echo "Configure image: [$kiwi_iname]..."

#======================================
# Call configuration code/functions
#--------------------------------------
...

#======================================
# Exit safely
#--------------------------------------
exit
```

Common functions

The `.kconfig` file allows to make use of a common set of functions. Functions specific to SUSE Linux specific begin with the name *suse*. Functions applicable to all linux systems starts with the name *base*. The following list describes the functions available inside the `images.sh` script.

[baseCleanMount]
: Umount the system filesystems `/proc`, `/dev/pts`, and `/sys`.

[baseGetProfilesUsed]
: Return the name(s) of profiles used to build this image.

[baseGetPackagesForDeletion]
: Return the list of packages setup in the packages `type="delete"` section of the `config.xml` used to build this image.

[baseSetupOEMPartition]
: Writes the file `/config.oempartition` depending on the following `config.xml` parameters: oem-reboot, oem-swapsize, oem-systemsize, oem-swap,oem-boot-title,oem-re-

covery, oem-kiwi-initrd. kiwi takes the information from `config.xml` and creates the config.oempartition file as part of the automatically created boot image (initrd). The information must be available as part of the boot image because it controls the OEM repartition workflow on first boot of an OEM image. Detailed information about the meaning of each option can be found in the OEM chapter of the KIWI cookbook.

[suseGFXBoot {theme} {loadertype}]
: This function requires the gfxboot and at least one bootsplash-theme-* package to be installed in order to work correctly. The function creates from this package data a graphics boot screen for the isolinux and grub boot loaders. Additionally it creates the bootsplash files for the resolutions 800x600, 1024x768, and 1280x1024

[suseStripKernel]
: This function removes all kernel drivers which are not listed in the *drivers sections of the `config.xml` file.

[suseStripInitrd]
: This function removes a whole bunch of tools binaries and libraries which are not required in order to boot a suse system with KIWI.

[Rm {list of files}]
: Helper function to delete files and announce it to log.

[Rpm {rpm commandline}]
: Helper function to the rpm function and announce it to log.

[Echo {echo commandline}]
: Helper function to print a message to the controlling terminal.

[Debug {message}]
: Helper function to print a message if the variable `DEBUG` is set to 1.

Profile environment variables

The .profile environment file contains a specific set of variables which are listed below. Some of the functions above makes use of the variables.

[$kiwi_iname]
: The name of the image as listed in config.xml

[$kiwi_iversion]
: The image version string major.minor.release

[$kiwi_keytable]
: The contents of the keytable setup as done in `config.xml`

[$kiwi_language]
: The contents of the locale setup as done in `config.xml`

[$kiwi_timezone]
: The contents of the timezone setup as done in `config.xml`

[$kiwi_delete]
: A list of all packages which are part of the packages section with `type="delete"` in `config.xml`

[$kiwi_profiles]
A list of profiles used to build this image

[$kiwi_drivers]
A comma separated list of the driver entries as listed in the drivers section of the `config.xml`. Similar variables exists for the usbdrivers and scsidrivers sections

[$kiwi_size]
The predefined size value for this image. This is not the computed size but only the optional size value of the preferences section in `config.xml`

[$kiwi_compressed]
The value of the compressed attribute set in the type element in config.xml

[$kiwi_type]
The basic image type. Can be a simply filesystem image type of ext2, ext3, reiserfs, squashfs, and cpio or one of the following complex image types: iso split usb vmx oem xen pxe

kiwi::kiwirc

KIWI::kiwirc — Resource file for the Kiwi imaging system

Description

The KIWI imaging toolchain supports the use of an optional resource file named `.kiwirc` located in the users home directory.

The file is sourced by a Perl process and thus Perl compatible syntax for the supported variable settings is required.

Example A.3. Template for .kiwi.rc

```
$BasePath='/usr/share/kiwi';
$Gzip='bzip2';
$LogServerPort='4455';
$System='/usr/share/kiwi/image';
```

Supported Resource Settings

KIWI recognizes the BasePath, Gzip, LogServerPort, LuksCipher, and System settings in the `.kiwirc` file.

[BasePath]
: Path to the location of the KIWI image system components, such as modules, tests, image descriptions etc.

  The default value is `/usr/share/kiwi`

[Gzip]
: Specify the compression utility to be used for various compression tasks during image generation.

  The default value is **gzip** `-9`

[LogServerPort]
: Specify a port number for log message queuing.

  The default value is off

[LuksCipher]
: Specify the cipher for the encrypted Luks filesystem.

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

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

Index