Chapter 5. Image Building with KIWI

Marcus Schäfer
Frank Sundermeyer

Basic structure only
$Revision: 212 $
$Date: 2007-02-22 17:21:45 +0100 (Thu, 22 Feb 2007) $

TODO: Abstract

5.1. Introduction

The OpenSuSE KIWI Image System provides a complete operating system image solution for Linux supported hardware platforms as well as for virtualisation systems like Xen, VMware and others. The KIWI architecture is designed as a two stage system. The first stage, based on a valid *software package source*, creates a so called *physical extend* as provided by an image description. The second stage creates an operating system image from the physical extend . The result of that second stage is called a *logical extend* or operating system image.

Concept

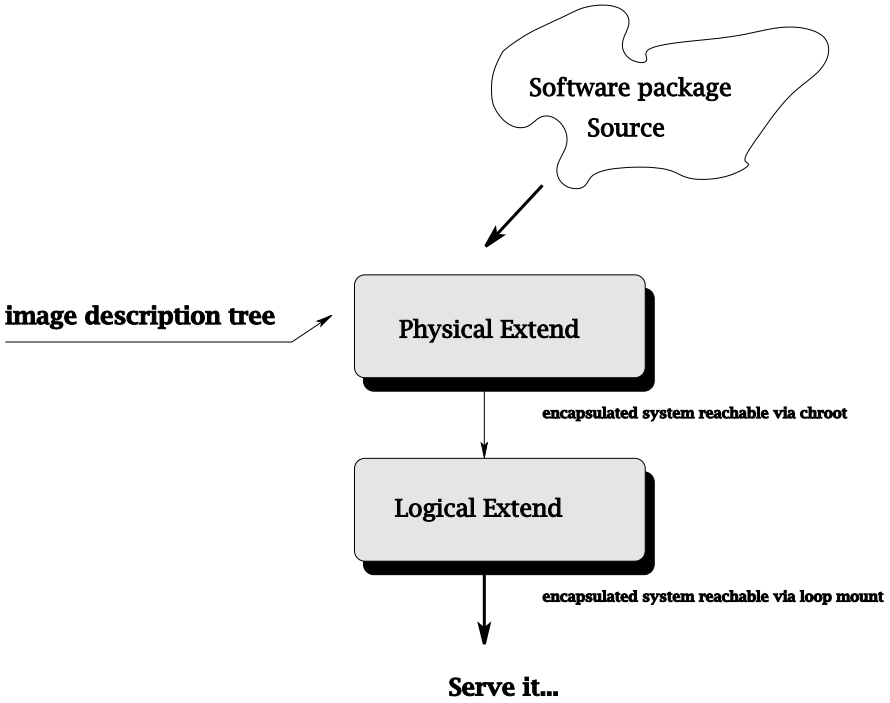
The first stage created by KIWI, the physical extend, requires at least one valid software package source, a so-called repository, in order to access the software to build a system. A repository consists of software packages, organized in a package tree that also includes some meta data. Software repositories can exist in different formats, therefore KIWI uses a Package-Manager to access them. With kiwi you have the choice for either *smart* or *zypper* to be used as package manager. Smart handles a wide range and also the most important repository formats. More information on smart is available at `http://labix.org/smart`.

The second stage—creating an operating system image—takes place without user interaction. Therefore all the necessary information needs to be created prior to the image building process. An image description tree stores all these information needed to create an image (see Section 5.1, “Introduction” for details).

Operating System Images

A regular installation process is starting from an installation image and installs single pieces of software until the system is complete. Normally such an installation process is interactive - the user is able to alter installation settings and configuration options. Contrary to that, 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 installed to a system storage device (no matter if this is a volatile or non volatile storage device). An operating system image is deployed “as is”—no user interaction is possible.

Figure 5.1. Image Serving Architecture



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

Supported Image Types

The logical extend is the final result of an image creation process and represents an operating system as part of a specific filesystem which could also be covered by the structures of real or virtual hardware layers. There are different types of images whereas kiwi supports the following:

The Live-System image [iso]
: The iso type is used to create live systems. A live system is an operating system on a CD or DVD. When the system boot all data is read from the CD/DVD. The system provides write support but all data is stored in RAM. So as soon as the system shut down the data will be lost. The generated iso file needs to be burned on the media

The virtual hard disk image [vmx]
: The vmx type is used to create a virtual disk. The disk provides the partition information the boot manager and all other data which are found on a real disk as well. Such an image can be used as disk for full-virtual systems like Qemu or VMware. Kiwi also creates the VMware configuration file if requested. The generated image requires a virtualisation software to be installed. In a full virtualized all components are virtualized. This includes the storage devices as well as the processor and all other parts of the system. To activate a virtual disk system system the user only needs to call the correct “player” application which in case of QEMU is `qemu` and in case of VMware `vmplayer` is used.

The para-virtual Xen image [xen]
: The xen type is used to create an operating system image based on a given filesystem including a special xen boot image, the xen kernel and the xen configuration file. If the current system is a xen hypervisor one

can create new para virtual machines with the data provided by kiwi. The generated image requires a Xen hypervisor running on the computer.

The USB-Stick image [usb]
: The usb type is used to create an operating system image based on a given filesystem including a special usb boot image. kiwi is able to install the generated image onto an USB stick. If your system BIOS is able to boot from a USB stick you can use this stick as complete operating system. Compared to a live system an USB stick provides permanent storage of your private data as well. The generated image can be installed by a subsequent kiwi call

The network image [pxe]
: the pxe type is used to create an operating system image based on a given filesystem including a special network boot image. The image itself will be stored on a server from where the boot image can download and activate it. The generated image requires a DHCP/TFTP network infrastructure running on a server

The network split image [split]
: the split type is used to create an operating system image based on two given filesystems. The image will be divided into two portions whereas the first portion represents the data which requires read/write access and the second portion represents the read-only data. According to this the second porition can be part of a compressed read-only filesystem. This image type doesn't automatically create its boot image and works only with netboot and xennetboot boot images. The generated image requires a DHCP/TFTP network infrastructure running on a server

The image base type is referenced in the main image configuration file config.xml and has several mandatory/optional parameters. The parameters influences the operating system environment like the used filesystem but the result is still an image made for the purpose described by the base type.

5.2. Creating Operating System Images

When starting to create an image with kiwi it's required to create a so called *system image description*. Such a description is a directory containing at least one file named *config.xml*. The config.xml describes what packages/patterns should make your image from which source repositories the data should be obtained which image type(s) can be generated from this description and many more (see Section 5.3, “The KIWI image description”).

In order to be able to start with an example image one can use an existing system image description as template. An image description is normally provided as architecture independent package. For your reference download the *kiwi-desc-livesystem* from here: `http://software.opensuse.org/openSUSE:/Tools`. This package contains descriptions for openSUSE live systems which can serve as basis for any other image type as well.

The process of creating an image always start with a *prepare* command which requires at least the path to your system image description and an optional destination directory as arguments:

```
kiwi --prepare /path/to/description --root /tmp/myroot
```

The result of this first step is an operating system which has its root directory below the given destination directory, /tmp/myroot. After this step the new root directory serves as data source for creating the requested operating system image. This step is called the *create* step and requires at least the path to your previosly created new root environment plus a destination directory for the image files as arguments:

```
kiwi --create /tmp/myroot --destdir /tmp/myimages
```

The destination directory /tmp/myimages must exist and kiwi will store all file for the requested image type below it. kiwi allows to specify *defaultroot* and *defaultdestination* attributes as part of the config.xml. If these are present and no --root and/or --destdir options are specified the information from this default attributes are used.

kiwi does not only create your system image it also provides a subsystem for the image deployment. Each system image somehow needs to be activated. For example the system image for a network client needs to be downloaded from a server, it must be installed, maybe a disk needs to be partitioned, etc. All the steps before a system image

is activated are done in the so called *boot image*. kiwi provides boot images for all of its supported image types and furthermore kiwi creates them automatically while the *create* step is called. People who are used to Linux may know the name *initrd*, the kiwi boot images are special initrd images concerning the image type they should activate. The boot images are implemented as normal kiwi image descriptions and are stored in a subdirectory below /usr/share/kiwi/image. The subdirectories there use the naming scheme *type*boot. Concerning this the boot images for network images are stored in the directory netboot/ and the ones for the virtual disk images are stored in vmxboot/ and so on. The boot images shouldn't require any change they are a service to the customer, automatically created to provide a maximum of comfort to the customer. For detailed information of the boot images see the section called "`config`".

5.3. The KIWI image description

The creation of operating system images from a physical extend is based on image description trees. An image description tree contains a set of files in a certain directory structure that are required to generate an image using `kiwi`. An image description tree is organized as follows:

```
|
|- config.xml
|- config [optional]
|   |- package configuration scripts
|
|- cdboot [optional]
|   |- isolinux.cfg
|   |- isolinux.msg
|   |- isolinux.sh -> ../../suse-isolinux
|
|- config.sh [optional]
|- images.sh [optional]
|- config-yast.xml [optional]
|- config-cdroot.tgz [optional]
|- root [optional]
    |- root tree files/directories
```

`config`

Optional Subdirectory that contains shell scripts that are executed after all packages have been installed, for example to remove 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` file.

`cdboot`

An optional directory needed when creating a bootable CD. It contains files required by the isolinux boot loader. This includes the `isolinux.cfg`,`isolinux.msg` configuration files and the `isolinux.sh` build script. It creates an ISO image from a prebuild CD tree based upon the configuration from `isolinux.cfg`.

`config.sh`

Optional configuration script while creating the physical extend. This script is executed at the end of the installation when having switched to the operating sstem image with `chroot`, but *before* the package scripts paced in the config directory (see the section called "`config`") have run. It is used to configure the image system by, for example, activating or deactivating services.

`images.sh`

Optional configuration script executed at the beginning of the image creation process when creating the logical extend. It cleans the image system from programs and files only needed while the physical extend exists.

config-yast.xml

Optional AutoYaST configuration file. To generate such a file, check *Clone This System for AutoYaST* during the installation of openSUSE. This creates a ready-to-use profile as `/root/autoinst.xml` that can be used to create clones of this particular installation. To create an autoinstallation file from scratch or to edit an existing one, use the YaST module *Autoinstallation*.

In order to use `/root/autoinst.xml` move it to the images description tree and rename it to `config-yast.xml`. kiwi will process the file and setup your image as follows:

- When booting the image YaST is automatically started in AutoYaST mode
- The system is configured by YaST by applying the rules from `config-yast.xml`
- If the process finishes successfully the environment is cleaned and AutoYaST will not be started with next reboot.

config-cdroot.tgz

Optional compressed tar archive which is used for live systems only. The data in the archive will be uncompressed and stored in the CD/DVD root directory. The archive could be used to integrate a license or readme information on the CD or DVD

root

Subdirectory that contains special files, directories, and scripts for adapting the image environment *after* the installation of all packages. The entire directory is copied into the root of the image tree using `cp -a`. The data in root directory allows you to customize your image with data that doesn't exist in the form of a package

config.xml

The main configuration file, defining image type, base name, repositories, profiles, options, and the package/pattern list.

Note

All values entered within the `image,preferences` and `drivers` elements of the `config.xml` file are additionally stored in a file called `.profile`. This file is created before the execution of an image script like `config.sh`, `images.sh`, or a “package script”. This makes the parameters of the config file are available as variables that can be sourced via script. Such a script should follow this template (replace `name` by the image name):

```
#!/bin/sh
test -f /.kconfig && . /.kconfig test -f /.profile
test -f /.profile && . /.profile test -f /.profile
echo "Configure image: [$name]..."
...
exit 0
```

Such a script is called within the image environment, which means it is not possible to damage the host system with it script even if you are using absolute paths

```
<image name="Name" inherit="optional path" schemeversion="1.4">❶
```

```
  <description type="boot|system">❷
    <author>Author</author>
    <contact>Contact</contact>
    <specification>Specification</specification>
  </description>
  <preferences>❸
    <type primary="true" boot="..." filesystem="..." flags="...">Type</type>
    <version>Version</version>
    <size unit="Unit">Size</size>
    <packagemanager>Name</packagemanager>
    <rpm-check-signatures>True|False</rpm-check-signatures>
    <rpm-force>True|False</rpm-force>
    <keytable>Name</keytable>
    <timezone>Name</timezone>
    <locale>Name</locale>
    <defaultdestination>Path</defaultdestination>
    <defaultroot>Path</defaultroot>
    <compressed>Yes|No</compressed>
  </preferences>
  <profiles>❹
    <profile name="Name" description="Description"/>
  </profiles>
  <users group="Groupname">❺
    <user name="User" pwd="Password" home="Homedirectory"/>
  </users>
  <drivers type="Type" profiles="Name">❻
    <file name="Filename"/>
  </drivers>
  <repository type="Type">❼
    <source path="Url"/>
  </repository>
  </deploy server="IP address" blocksize="Size">❽
    <partitions device="Devicename">
      <partition type="Type" number="Number" size="Size"/>
    </partitions>
    <union rw="RW-Device" ro="RO-Device" type="aufs|unionfs"/>
    <configuration source="Source" dest="Destination"/>
  </deploy>
  <packages type="Type" profiles="Name">❾
    <package name="Packagename" arch="Arch"/>
    <opensusePattern name="Patternname"/>
    <ignore name="Packagename"/>
  </packages>
</image>
```

❶ The `image` element contains the attribute `name` which spcifies the base name of the image. It is automatically expanded with the version number and the current date. The version number is extracted from the directory in which the description files for this image are located. When using the optional attribute `inherit` with a path to another kiwi description, this description will be prepended to the current one. The mandatory attribute `schemeversion` must be set and allows version *1.4* at the moment.

❷ `description` element contains the attributes `author`, `contact` and `specification`. Author should be the name of the responsible person for this image. Contact should be a valid e-mail address in order to get in touch with the responsible person and the specification attribute contains a free form text describing what this image is good for.

❸ The `preferences` element contains information needed to create the logical extend. The following sub-elements are defined:

`type`
The image type of the logical extend. When specifying multiple entries, the additional attribute `primary` setting the primary type needs to be filled in, otherwise the first entry of the `type` value is used.

The following values for `type` are valid:

a. `ext2`, `ext3`, `reiserfs`, `squashfs`, `cpio`

b. `iso`, `split`, `usb vmx`, `xen`, `pxe`

The second group of types requires additional attributes:

`primary`
attribute to specify the primary type. The `kiwi` option `--type` allows to select between the types

`boot`
attribute to specify the boot image (`initrd`) which should be used and created for this system image description. The boot images for kiwi are stored at `/usr/share/kiwi/image` and are grouped by function into the following directories: `isoboot`, `netboot`, `xennetboot`, `usbboot`, `vmxboot` and `xenboot`. The attribute value is the path relative to `/usr/share/kiwi/image`.

`flags`
attribute to specify flags for the image type. Currently only the `compressed` flag for the `iso` type exists. This flag causes the live media to be based on a squashfs compressed file system.

`filesystem`
attribute to specify the filesystem. Could be set to `ext2`, `ext3`, `reiserfs`, `squashfs` or `cpio`

The following list is an overview of different boot image (`initrd`) types. While creating the system image kiwi automatically creates the specified ...*boot* boot image. The description for the boot image must exist in */usr/share/kiwi/image/...boot/suse-...*

`<type boot="isoboot/suse-..." flags="unified">iso</type>`
If the optional attribute `flags` is set to `compressed` squashfs compression will be used for the read-only part of the iso. If set to `unified` the squashfs compressed read-only part will be used in combination with the union filesystem named aufs which allows a complete read-write system as long as the system runs.

`<type boot="usbboot/suse-..." filesystem="ext3">usb</type>`
The attribute `filesystem` specifies one of `ext2`,`ext3`,`squashfs` or `reiserfs` The created system and boot images are suitable to run on an USB stick and can be deployed to it with the kiwi `--bootstick / --bootstick-system` options

`<type boot="vmxboot/suse-..." filesystem="ext3">vmx</type>`
The attribute `filesystem` specifies one of `ext2`,`ext3`, or `reiserfs`. The final result is one file appearing as virtual disk which includes the system and boot images as well as the disk geometry partition information and the boot manager

`<type boot="xenboot/suse-..." filesystem="ext3">xen</type>`
The attribute `filesystem` specifies one of `ext2`,`ext3`, or `reiserfs`. The created system and boot images as well as the xen configuration file are suitable to run a Xen virtual machine using the `xm` program.

`<type boot="netboot/suse-..." filesystem="ext3">pxe</type>`
The attribute `filesystem` specifies one of `ext2`,`ext3`, or `reiserfs` and the value of `boot` is the path to a netboot or xennetboot boot image description. The created system and boot images are

suitable to drive a network client station. An appropriate network infrastructure including a DHCP and TFTP server is required in order to activate a network client.

`<type boot="netboot/suse-..." filesystem="type-rw, type-ro">split</type>`

The attribute `filesystem` specifies a filesystem pair whereas *`type-rw`* specifies one of `ext2`, `ext3` or `reiserfs` and *`type-ro`* specifies one of `ext2`, `ext3`, `reiserfs`, `cramfs` or `squashfs`. Booting split images is only supported by the netboot and xennetboot boot images which need to be created with a separate kiwi run.

`version`

The three-part version number with the following format: *`Major.Minor.Release`*. The following rules should be applied when incrementing the version number:

- Increment the `Release` number on minor modifications where no packages have been added or removed
- Increment the `Minor` number and reset the `Release` number to `0` in case packages have been added or removed
- Increment the `Major` number if the size of the image changes.

`size`

Specifies the size of an image with a numeral value in Megabytes or Gigabytes. Use the attribute `unit` to assign the unit: `M` for Megabytes or `G` for Gigabytes.

Note

kiwi supports the feature of extending the image size automatically if the specified value is too small. If the actual size is more than 100MB larger than the spcified one, kiwi will abort with an error message. On the other hand, kiwi does *not* automatically reduce the images size if the specified value is too large, because the extra space may be needed to, for example, run custom scripts. If no size is specified at all kiwi will use the required size plus approximately 10% free space.

`packagemanager`

Name of the packagemanager to be used for package installation. Currently `smart` is the default packagemanager but `zypper` is supported as well.

`compressed`

The compressed flag indicates whether the system image should be compressed or not. This affects only the kiwi output file itself and has no influence on the operating system living in the image. For network clients it makes sense to download the image as compressed image.

`keytable`

Contains the name of the console keymap to use. The value corresponds to a map file in `/usr/share/kbd/keymaps`. The variable `KEYTABLE` within the file `/etc/sysconfig/keyboard` will be set according to the keyboard mapping.

`timezone`

The time zone. Available time zones are located in the directory `/usr/share/zoneinfo`. Specify the attribute value relative to `/usr/share/zoneinfo`— e.g. `Europe/Berlin` for `/usr/share/zoneinfo/Europe/Berlin`. kiwi uses this value to set up the timezone in `/etc/localtime` for the image.

`locale`

Contains the name of the locale to use and therefore defines the contents of the `RC_LANG` system environment variable set in `/etc/sysconfig/language`

`defaultdestination`
: Optional attribute `defaultdestination` is used if the option `destdir` is not specified while calling kiwi.

`defaultroot`
: Optional attribute `defaultroot` is used if the option `root` is not specified while calling kiwi.

❹ The optional `profiles` element contains information which allows you to maintain one image description while still allowing for some variation in the set of packages and drivers that are included. A `profile` element has to be specified for each variation. The child element `profile` with the attributes `name` and `description` specifies an alias name which is used in sections to mark them as belonging to a profile and a more detailed description explaining what this profile does. To mark a set of packages and/or drivers as belonging to a profile, simply annotate them with the `profiles` attribute. If a `packages` tag has no `profiles` attribute, it is assumed to be present for all profiles. All of the above also goes for the `drivers` tag too.

❺ The optional `users` element contains the users to be added to the image. The attribute `group` specifies the group the user(s) belongs to. If this group doesn't exist it will be created. A `user` element has to be specified for each group. Specify the users belonging to a group with the child element `user` with the attributes `name`, `pwd` and `home` for username, password and the path to the home directory.

❻ The optional `drivers` element contains driver file names. The names are interpreted as general driver name and useed if they are contained in the kernel tree. The attribute `type` specifies one of the following driver types:

`netdrivers`
: Every file is specified relative to the directory `/lib/modules/<Version>/kernel/drivers/net`

`usbdrivers`
: Every file is specified relative to the directory `/lib/modules/<Version>/kernel/drivers/usb`

`drivers`
: Every file is specified relative to the directory `/lib/modules/<Version>/kernel`

❼ The `repository` element defines the source path and type used by the package manager. The attribute `type` specifies the type of the repository, for example, `type="yast2"`. The child element `source` contains the attribute `path` to setup the the location of the repository, for example, `source="/image/CDs/full-i386"`. The path specification can be one of:

- local path starting with `/`
- `this://` relative path name which is relative to the image description which is referenced
- `http://` or `ftp://` Network-Location
- opensuse://Project-Name
- The path can include the %arch macro if needed

Multiple repository tags are allowed. For information on how to setup a smart source refer to `http://labix.org/smart`.

❽ The `deploy` element make sense for network clients only and describes where to get the system image and how should the client be prepared for that image. Preparation in the sense of an image means how should the disk be partitioned or what configuration files should be included. According to this the attributes `server` and `blocksize` specifies the TFTP server which takes over control for the download. The `partitions` tag specifies the partition(s) for one disk device (`device`). Each partition is specified by one subtag named `partition` which defines the type (see sfdisk --list-type), partition number, size , optional mountpoint and optional information whether this partition is the system image target partition or not. With the kiwi netboot image the first partition is always the swap partition and the second partition is by default used for the system image. With the optional target flag it is possible to indicate another than the second partition to be the one

the system image is installed on. If *size* is set to "image" kiwi will calculate the required size for this partition in order to have enough space for the later image. The optional `union` is used if the system image is based on a read-only filesystem like squashfs. In this case kiwi can setup an aditional write partition and combine both with the given overlay filesystem. Right now there are two of such filesystems one is called *unionfs* and the other (preferred) one is called *aufs*. The partition which holds the read-only system image must be set as value to the attribute `ro` and the partition which should serve as write partition must be set as value to the attribute `rw`. The optional `configuration` tag can be used to integrate configuration files to a network client which are remotely stored on the server. The attribute `source` specifies the path on the server used by a tftp client program to download the file and the attribute `dest` specifies the target relatively to the root (/) of the network client.

❾ The `packages` element contains the list of packages and/or pattern names to be used with the image. There are three different types of package sets or patterns. The type is specified with the attribute `type`:

`image`
: Packages used to finish the image installation. All packages which make up the image are listed there.

`boot`
: Packages used to start creating a new operating system root tree. Basic components which are required to chroot into that system like `glibc` are listed here.

`xen`
: Packages used when the image needs support for Xen based virtualisation. The attributes `memory` and `disk` defines how much memory the virtual system requires and what kind of device the disk should appear in the virtual instance. This information is used to create the appropriate Xen configuration file.

`vmware`
: Packages used when the image needs support for VMware based virtualisation. The attributes `memory` and `disk` defines how much memory the virtual system requires and what kind of device the disk should appear in the virtual instance. This information is used to create the appropriate VMware configuration file.

Using a pattern name will enhance the package list with a number of additional packages belonging to this pattern. Support for patterns is SuSE-specific and available from openSUSE 10.2 or higher. If a pattern contains unwanted packages it is possible to specify a an ignore list with the element `ignore` and the attribute `name` containing the package name. Restricting a package to a specific architecture can be done by using the `arch` attribute a comma seperated list of allowed architectures in the `package` element.

5.4. Activating an Image

After a logical extend (an image) has been created from a physical extend there are in principal four possibilities to activate the image:

- On a *local system* the image can be installed by dumping (with the command `dd`) the image file on a previously created partition on a local harddisk. To activate the system a boot manager like grub or lilo needs to be used. An USB stick system is also treated as a local system but kiwi supports you with the installation of such a system by its own --bootstick option.

- In case of a *network enabled system* (netboot client) the image can be installed via a special boot image. The boot image which serves as initial ramdisk (`initrd`) and the appropriate kernel are downloaded from a network service. The linux kernel automatically calls a program named `linuxrc` which takes over all tasks needed to download and install the system image. The installation can be done persistently on disk or temporary into the RAM of the machine. The network boot protocol supported by kiwi is PXE

- In case of a *para virtualized target system* like Xen, the image can be be installed by copying the image file and the kiwi created xen configuration file onto the target system. To activate the virtual system the command

  ```
  xm create -c <xen config file>
  ```

  is used.

- In case of a *full virtualized target system* like VMware, or QEMU the image represents a virtual disk which can be “played” by the virtualisation system.

No matter which of the above mentioned scenarios applies, there is always a special image which takes over control of the activation or deployment process. These images are called boot images and the following sections will explain the kiwi provided boot images in detail.

The KIWI netboot image

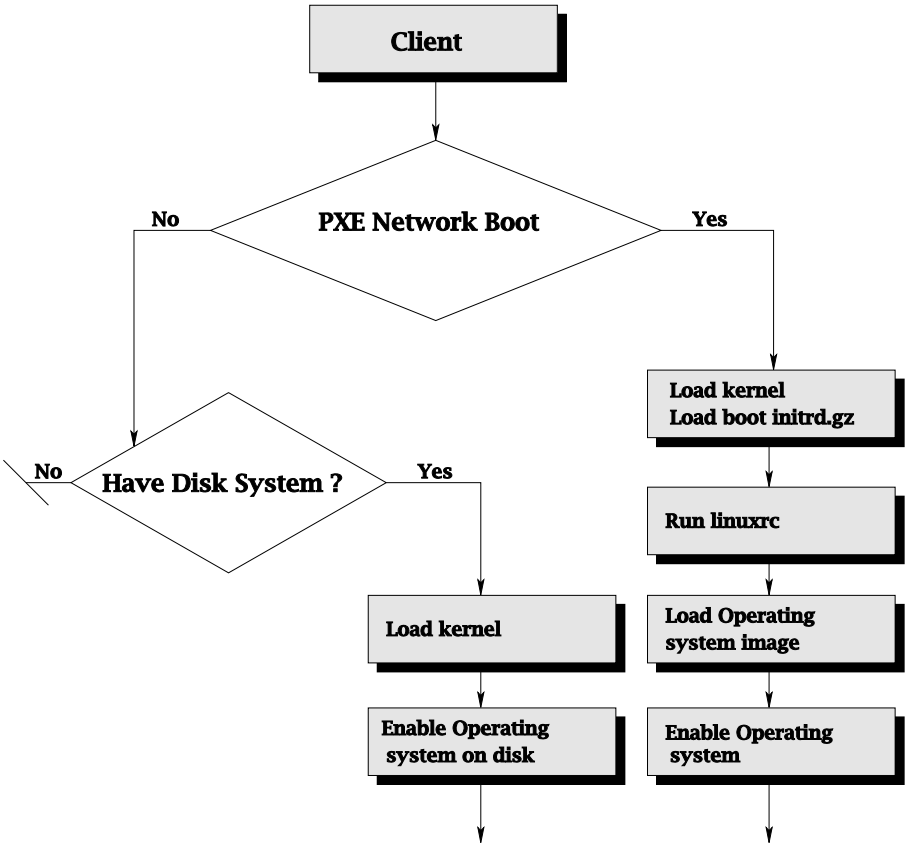
The KIWI netboot image can be used to install an operating system image to a network client. To establish communication with the client, a boot server infrastructure with the following services is required:

- a DHCP server to provide an IP address for the client
- a TFTP server to allow file transfer from and to the client

The Boot Process of a Netboot System

The following graphic illustrates a simplified boot process of a netboot client.

Figure 5.2. The Boot Process of a Netboot Client



If the system is able to boot via a network, it will load the kernel and the compressed boot image from the network. The “brain” of the boot image is the `linuxrc` script, which does all the stuff controlled by an image configuration file also obtained from the network. The major task is to download and activate the operating system image. The boot image is exchanged for the operating system image to be activated. The following overview describes the steps that take place when the netboot client is booted:

- Via PXE network boot or boot manager (GRUB), the client boots the initrd (`initrd.gz`) which is served by the TFTP server. If no PXE boot is possible, the client tries to boot from a local hard disk.

- Running `linuxrc` starting the following process:

1. The file systems required to receive system data, are mounted, for example the `/proc` file system.

2. The kernel parameters are imported. If there is an IMAGE parameter it is assumed the system boots locally and the variable `LOCAL_BOOT` is set.

3. If `LOCAL_BOOT` is not set, network support is activated. The network card is probed by using the `hwinfo` command. The appropriate module is loaded using `modprobe`. Any dependencies to other modules will be resolved.

4. If `LOCAL_BOOT` is not set, the network interface is set up via DHCP. After the interface has been established, the DHCP variables are exported into the file `/var/lib/dhcpcd/dhcpcd-eth0.info` and the contents of DOMAIN and DNS are used to generate a `/etc/resolv.conf`.

5. If `LOCAL_BOOT` is not set, The TFTP server address is acquired. During this step, a check is made to determine whether the kiwitftp *kernel* parameter is set. If this is not the case a check whether the host name tftp.`$DOMAIN` can be resolved is made. If both tests fail, the DHCP server is used as the TFTP server. For more infomation about the TFTP server structure, refer to the section called “The TFTP Server Structure”.

6. If `LOCAL_BOOT` is not set, the configuration file is loaded from the server directory `/var/lib/tftpboot/KIWI` via TFTP. At this point, the client expects the file `config.<MAC Address>`. If this file is not available and cannot be loaded, it is assumed that the client hasn't existed before and can be immediately registered by uploading a control file to the TFTP server's upload directory `/var/lib/tftpboot/upload`. After the upload, the client branches off into a loop in which the following steps are taken:

   - the DHCP lease file is restarted (`dhcpcd -n`).
   - a new attempt is made to load the file `config.<MAC address>` from the TFTP server.
   - if the file does not exist, there is a 60 second time-out before a new run begins.

   If the configuration file does load, it contains data on image, configuration, synchronization, or partition parameters. For more infomation about the file format of the configuration file, refer to the section called “The Netboot Client Configuration File `config.<MAC Address>`”.

7. All registered kernel modules are loaded. The kernel provides a system which allows to check for a module alias registered automatically by the kernel during boot time. If such an alias matches the modinfo information from a kernel module it will be loaded.

8. If `LOCAL_BOOT` is not set, the `PART:` line in the configuration is analyzed. If it is found a check is performed to see whether any local system needs to be updated. If not, the local boot process continues immediately. No image download occurs. If an update is required (or no operating system can be found), the client's hard disk is partitioned.

9. If `NFSROOT` and `NBDROOT` and `LOCAL_BOOT` is not set, the images are downloaded with TFTP. If `LOCAL_BOOT` is set this part of the initrd will only read the IMAGE information for later usage. If `NFSROOT` is set the image root device is set to a remote NFS path. If `NBDROOT` is set the nbd kernel module is loaded and the nbd-client program setup a new network block device called /dev/nd0. The image root device is then set to the network block device

10. If `LOCAL_BOOT` is not set, the checksums are checked. If the check fails another download attempt is started.

11. If `LOCAL_BOOT` is not set, a check for `RELOAD_CONFIG` is performed

12. The operating system image is mounted.

13. If `LOCAL_BOOT` is not set, the `CONF:` line is evaluated. All the specified files are loaded from the TFTP server and stored in a `/config/` path. Additionally the `KIWI_INITRD:` line is evaluated. The specified initrd file will be downloaded from the tftp server and stored in the system image as file `/boot/initrd`.

14. If `LOCAL_BOOT` is not set, all the user-land processes based on the boot image (`dhcpcd -k`) will be terminated.

15. Check if the image is a splitted image by evaluating the contents of the `COMBINED_IMAGE` variable. Both image partsare combined into one system by creating the appropriate filesystem links.

16. If `LOCAL_BOOT` is not set, the filesystem type of the system image as well as the available kernels are determined.

17. If `LOCAL_BOOT` is not set, important configuration files like `/etc/fstab`, `/boot/grub/menu.lst`, `/etc/grub.conf`, and `/etc/sysconfig/kernel` are created.

18. If `LOCAL_BOOT` is not set, the configuration files stored in the `/config/` directory are copied into the mounted operating system image.

19. The system switches to the mounted operating system image. The root file system is converted to the operating system image via `pivot_root` or `mount --move`. All required configuration files are now present, because they had been stored in the operating system image or have been downloaded via TFTP.

20. The boot image is unmounted using an `exec umount` call.

21. At termination of `linuxrc` or the exec call, the kernel initiates the `init` process that starts processing the boot scripts as specified in `/etc/inittab`, for example, to configure the network interface.

The TFTP Server Structure

The TFTP server directory structure is divided into the following main areas:

Image configurations
: The `/var/lib/tftpboot/KIWI/` directory contains the various `config.<MAC Address>` image configuration files.

Configuration files
: The `/var/lib/tftpboot/KIWI/<MAC Address>/` directory contains the various system configuration files, such as `xorg.conf`.

Boot files
: The `/var/lib/tftpboot/boot/` directory is where the `initrd.gz`, and the kernel to boot are kept.

PXE second stage boot loader(s)
: The `/var/lib/tftpboot/` directory is where the boot loaders for PXE are kept (`pxelinux.0`, `mboot.c32`)

PXE configuration file
: `/var/lib/tftpboot/pxelinux.cfg` is the location of the PXE configuration file.

Image files and checksums
: The `/var/lib/tftpboot/image/` directory is where all the image files and their checksums are kept.

Upload area
: The directory `/var/lib/tftpboot/upload/` is the directory into which the `hwtype.<MAC Address>` files for registering new netboot clients are uploaded.

The Netboot Client Configuration File `config.<MAC Address>`

The configuration file contains data about image, configuration, synchronization, and 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. This is an example for a cash register configuration file:

```
IMAGE=/dev/hda2;image/browser;1.1.1;192.168.1.1;4096
CONF=/KIWI/00:30:05:1D:75:D2/ntp.conf;/etc/ntp.conf;192.168.1.1;1024,     \
     /KIWI/00:30:05:1D:75:D2/xorg.xonf;/etc/X11/xorg.xonf;192.168.1.1;1024
PART=200;S;x,300;L;/,500;L;/opt,x;L;/home
DISK=/dev/hda
```

The following format is used:

```
IMAGE=device;name;version;srvip;bsize;compressed,...
SYNC=syncfilename;srvip;bsize
CONF=src;dest;srvip;bsize,...,src;dest;srvip;bsize
PART=size;id;Mount,...,size;id;Mount
JOURNAL=ext3
DISK=device
```

`IMAGE`
: Specifies which image (`name`) should be loaded with which version (`version`) and to which storage device (`device`) it should be linked to (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.

  `srvrip` specifies the server IP address for the TFTP download. It must always be specified, except in `PART`.

  `bsize` specifies the block size for the TFTP download. Must always be specified, 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. 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 do not 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 a TFTP.

`CONF`
: Specifies a comma-separated list of source:target configuration files. The source (`src`) corresponds to the path on the TFTP server and is loaded via TFTP. The download is made to the file on the netboot client indicated by the target (`dest`).

`PART`
: Specifies the partitioning data. The comma-separated list must contain the size (`size`), the type number (`id`), and the mount point (`Mount`). The size is measured in MB by default. Additionally all size specifications supported by the sfdisk program are allowed as well. The type number specifies the ID of the partition. Valid ID's are listed via the `sfdisk --list-types` command. Mount specifies the partition's mount point.

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

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

RELOAD_IMAGE
: If set to a non-empty string, RELOAD_IMAGE forces the configured image to be loaded from the server even if the image on the disk is up-to-date. Used mainly for debugging purposes, this option only makes sense on diskful systems.

RELOAD_CONFIG
: If set to an non-empty string, forces all configuration files to be loaded from the server. Used mainly for debugging purposes, this option only makes sense on diskful systems.

COMBINED_IMAGE
: If set to an non-empty string, COMBINED_IMAGE indicates that the two images specified need to be combined into one bootable image, where 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 a local boot of the system. The variable's value must be set to the name of the initrd file which is used via PXE network boot. If the standard tftp setup suggested with the kiwi-pxeboot package is used, all initrd files reside in /var/lib/tftpboot/boot/ directory. Because the tftpserver does a chroot into the tftp server path you need to specify the initrd file as follows:

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

UNIONFS_CONFIG
: For netboot and usbboot images there is the possibility to use unionfs or aufs as a 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 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 explains the interconnections:

```
IMAGE=/dev/sda3;image/browser;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
```

Since the second element of the PART list must define the root partition, it is 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, 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.

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 and uploaded to the TFTP servers upload directory /var/lib/tftpboot/upload.

The KIWI xennetboot image

TODO

The KIWI isoboot image

TODO

The KIWI isoinst image

TODO

The KIWI vmxboot image

TODO

The KIWI oemboot image

TODO

The KIWI xenboot image

TODO

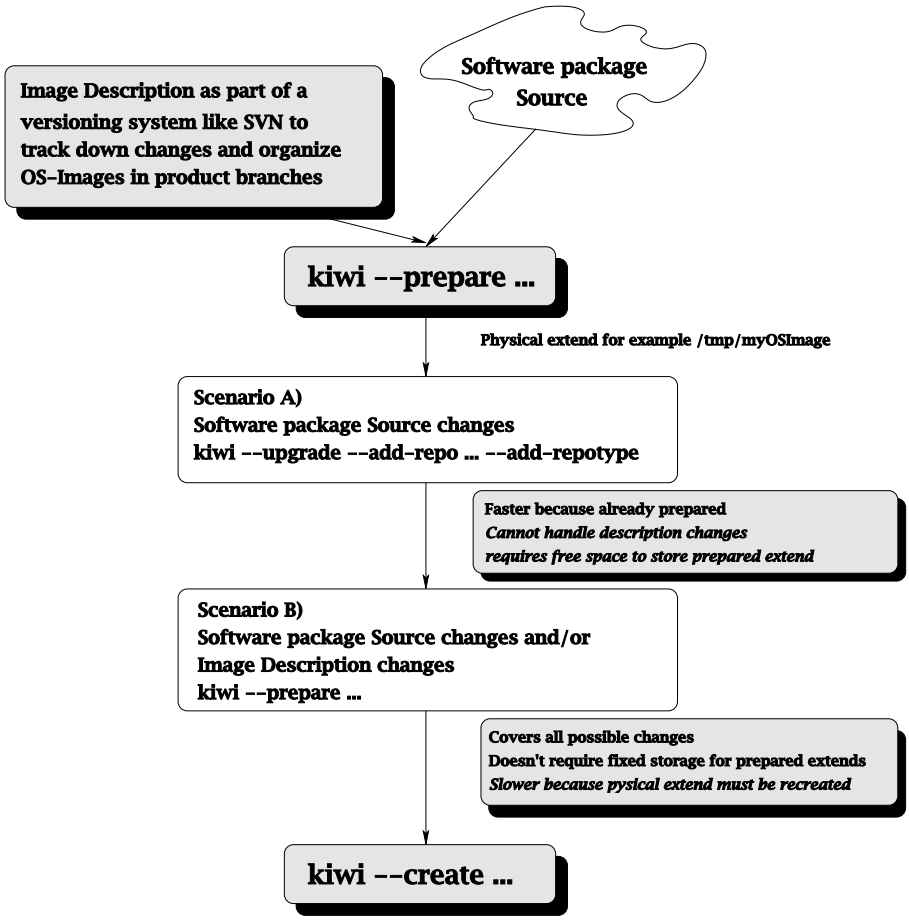
The KIWI usbboot image

TODO

5.5. Maintenance of Operating System Images

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

Figure 5.3. Image Maintenance Scenarios



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

Another reason for updating an image beside software updates are configuration changes or enhancements, for example an image should have replaced its browser with another better browser or a new service like apache should be enabled. In principal it's possible to do all those changes manually within the physical extend but concerning maintenance this would be a nightmare. Why, because it will leave the system in an unversioned condition. Nobody knows what has changed since the very first preparation of this image. So in short *dont't modify physical extends manually*. Changes to the image configuration should be done within the image description. The image description itself should be part of a versioning system like subversion. All changes can be tracked down then and maybe more important can be assigned to product tags and branches. As a consequence an image must be prepared from scratch and the old physical extend could be removed.

5.6. Real-Life Scenarios - A Tutorial

Creating an operating system image always implies the question how to activate the image on the target system. As there are many possible targets the so called image deployment higly depends on the system environment. If a customer buys a product for example SLEPOS the system environment is given and it is required to take care for the rules to successfully deploy the image. On the one hand this makes it easy for customers to control a complex system but on the other hand one will loose the flexibility to use the same system in another environment.

kiwi doesn't stick to a specific system environment but therefore it leaves out important tasks concerning the deployment architecture. So to say kiwi is an image creator but it doesn't setup the deployment infrastructure. This chapter provides information about the different possibilities to deploy an image and how kiwi fits into this picture.

Deploying via Network Using the PXE Protocol

PXE is a boot protocol mostly implemented in todays BIOS's or boot ROMs of some network cards. If activated it broadcasts the network for a DHCP to obtain an IP address and the information where to find a TFTP server to manage file transfers. If such a server exists the second stage bootloader controls the subsequently boot process. Using PXE with kiwi requires the infrastructure explained in section FIXME \ref{section:tftpstruct} and a TFTP server as well as a DHCP server running. kiwi provides the package `kiwi-pxeboot` which setup the boot structure and installs some prebuild boot images.

If the plan is to use a system image for which no prebuild boot image exist, it is needed to create a custom boot image before a system image could be deployed. Boot images consists of the image itself and the appropriate kernel to that image. Both file must be stored in the directory `/var/lib/tftpboot/boot`. Assuming there is no prebuild boot image for the openSuSE 10.2 distribution the steps to create it are as follows:

```
cd /usr/share/kiwi/image
kiwi --prepare netboot/suse-10.2 --root /tmp/myroot
kiwi --create /tmp/myroot -d /tmp
```

The result of this example is created in the /tmp directory:

```
ls -1 /tmp/initrd-netboot*
   /tmp/initrd-netboot-suse-10.2.i686-2.1.1.gz
   /tmp/initrd-netboot-suse-10.2.i686-2.1.1.kernel.2.6.18.2-31-default
```

If you don't want to prepare/create the boot image manually you can let kiwi do the job by setting up the `pxe` type in your system image `config.xml` file. For example:

```
<preferences>
  <type filesystem="ext3" boot="netboot/suse-10.2">pxe</type>
  ...
</preferences>
```

In this case the boot image will be created automatically and from the same source as the system image as soon as the \textbf{creation} step is performed for the system image.

The next step is to make the boot image known to the TFTP server. To do this the files must be copied to the `/var/lib/tftpboot/boot`} directory. Optionally they can be renamed. For the following example the boot image is renamed to `initrd` and the boot kernel is renamed to `linux`.

```
cp initrd-netboot.i686-2.1.1.gz /var/lib/tftpboot/boot/initrd
cp initrd-netboot.i686-2.1.1.kernel.2.6.18.2-31-default \
  /var/lib/tftpboot/boot/linux
```

Switching on the target machine, which needs to run PXE by default, will load the linux kernel and the kiwi created initrd. The initrd will register the machine if there is no configuration found in `/var/lib/tftpboot/KIWI`. Registration means a file including the MAC address of the machine is uploaded into the directory `/var/lib/tftpboot/upload`. For information about creating the configuration refer to section FIXME \ref{section:confmac}. The following example configuration fits for a machine including a disk on `/dev/sda`:

```
IMAGE=/dev/sda2;full-suse-10.2.i686;1.1.2;192.168.100.2;4096
PART=1024;S;x,x;L;/
DISK=/dev/sda
```

According to this configuration the kiwi boot image will try to download a system image named `full-suse-10.2.i686-1.1.2` from the TFTP server with IP address 192.168.100.2. On the TFTP server the system images are stored in `/var/lib/tftpboot/image`. The boot image will prepare the disk and create a 1GB swap partition and another full size linux partition. The process of creating this full-suse-10.2 image can be done by using kiwi. Once the boot image has successfully downloaded the system image it is getting activated and operates as configured.

Deploying via TFTP with Initial Boot from CD or USB Stick

Sometimes it is not possible to use PXE as boot protocol. This could happen if the target machine doesn't support PXE or the installed network card doesn't provide a PXE boot rom. Preconditioned your network provides a DHCP and a TFTP server the problem can be solved by storing the boot image on another bootable medium like a CD or an USB stick. Referring to the information from the section above it is easy to create a bootable CD / USB stick with kiwi:

```
kiwi --bootcd /tmp/initrd-netboot-suse-10.2.i686-2.1.1.gz
```

The command will create a bootable ISO image which only needs to be burned on a CD. The following file will be created after the call: `/tmp/initrd-netboot-suse-10.2.i686-2.1.1.cdboot.iso`.

Plug in an USB stick then call:

```
kiwi --bootstick /tmp/initrd-netboot-suse-10.2.i686-2.1.1.gz
```

The command will create a bootable USB stick. kiwi is searching for the USB stick plugged in before and lists all devices found. The user needs to select one of the devices by typing one of the suggested device names. It is important to be careful at that stage because all data on the given device will be lost. Because of this reason the user has to type the device name which hopefully increases the chance not to do something thoughtlessly.

Installation on a Client via CD

If there is no network infrastructure available a client can only receive the system image from a storage media like a CD or an USB stick. This section explains how to create a CD with kiwi which supports system image installation to the storage device of the client. The following steps needs to be processed:

1. Create the boot image which takes over the task of installing the system image. For that purpose kiwi provides the `isoinstboot` boot image type.

2. Next step is creating the system image which should make it on the client. The boot and the system image package source should be the same. The system image description requires a `deploy` section within the `config.xml` file. The deploy section describes how the clients storage device should be partitioned and which device is the root device for the later image.

3. Last step is to create an .iso file together with the boot- and system image. Therefore call kiwi like the following example:

   ```
   kiwi --installcd bootImageName --installcd-system systemImageName
   ```

Installation on a Client via USB Stick

*** not yet implemented ***

Split image system via PXE

kiwi supports system images to be splitted into two parts, a read-only part and a read-write part. This allows to put data on different filesystems which is mostly used to have read-only data available on a compressed filesystem like cramfs or squasfs. Please note that almost all compressed filesystems available have some kind of restrictions which needs attention before starting to use it in an image.

To turn a system image into a split image only the type of the image must be adapted. This information is part of the `config.xml` file and could be changed like the following example shows:

```
<preferences>
  <type filesystem="ext3,cramfs">split</type>
  ...
</preferences>
```

Creating an image from this description results in two image files whereas one of them will contain the `-read-only` extension in its name. Ay one may imagine booting such an image always requires a boot process which must be able to bring both images together again. Because of this, split images can only deployed in combination with one of the kiwi boot images.

To deploy the image only PXE or a boot CD / USB-stick can be used. The most important part while making use of split images is the configuration for the target machine. More information on this config.MAC file can be found in section FIXME \ref{section:confmac}. In case of a split image the following information must be provided:

- The `IMAGE` key must contain both images the read-write and the read-only image. The read-write image must appear as first entry in the list.
- The `PART` key has to specifiy a partition table with at least three partitions. A swap partition and two system partitions which provides enough space for the first and the second image portion
- The Option `COMBINED_IMAGE` to tell the boot image to combine both images into one entire system

The following example shows the configuration of a split image named minimal-10.1 / minimal-10.1-read-only

```
IMAGE=/dev/sda2;minimal-10.1.i686;1.1.2;192.168.100.2;4096,\
      /dev/sda3;minimal-10.1-read-only.i686;1.1.2;192.168.100.2;4096
PART=200;S;x,500;L;/,x;L;
DISK=/dev/sda
COMBINED_IMAGE=yes
```

Deploying via Network Using an NFS Mounted `root` System

In principal kiwi was designed to upload an image onto a client in different ways but out there you will find diskless machines as well. Those devices doesn't provide permanent storage and rely on the network. The most oft used process to activate such terminals is to NFS mount the system image via the network. kiwi supports that as well but it additionally requires a terminal server configuration which needs to export the system image using a NFS server. The following steps needs to be performed in order to activate a diskless station:

1. Prepare the system image using

   ```
   kiwi --root /tmp/kiwi.nfsroot --prepare ...
   ```

2. Setup an NFS server which exports the `/tmp/kiwi.nfsroot` path. The following export options in `/etc/exports` are recommended:

```
/tmp/kiwi.nfsroot  *(rw,no\_root\_squash,sync,no\_subtree\_check)
```

3. Create an appropriate netboot boot image (`initrd`) with kiwi. Appropriate means the package repository for the system image and the netboot image must be the same

4. Copy the boot image/kernel to the PXE server in `/var/lib/tftpboot/boot` The package `kiwi-pxeboot` helps you in setting up the PXE/TFTP server.

5. Create a `config.<MAC>` file in `/var/lib/tftpboot/KIWI` with the following contents:

   ```
   NFSROOT=129.168.100.7;/tmp/kiwi.nfsroot
   ```

6. Boot the client. If everything works the client will receive the boot image and kernel via PXE/TFTP. The boot image NFS mounts the system image according to the data in `config.<MAC>`. After that the mounted root filesystem will be activated.

USB stick system

A very popular method is storing complete operating systems on an USB stick. This means not only the boot image and the kernel is stored on the stick but also the entire system image is part of the stick. To be able to do this kiwi provides special boot image(s) located in `/usr/share/kiwi/image/usbboot`. Currently it is tested and available for SuSE Linux 10.2 only but can be adapted to other versions as well. To use your system image on a boot stick setup the system's `config.xml` as follows:

```
<preferences>
  <type filesystem="ext3" boot="usbboot/suse-10.2">usb</type>
  ...
</preferences>
```

After this the boot and system image needs to be created. Concerning to the size of the stick the image is not allowed to grow beyond it. Assuming the system image is named full-suse-10.2 the command is as follows:

```
kiwi --prepare full-suse-10.2 --root /tmp/mysystem
kiwi --create /tmp/mysystem -d /tmp
```

The result of the command above is represented in the image file `/tmp/full-suse-10.2.i686-1.1.2`. The usbboot image is created automatically and is stored as `/tmp/initrd-usbboot-suse-10.2.i686-2.1.1.gz`. To create a bootable stick with a SuSE Linux 10.2 operating system on it, plug in the stick and call:

```
kiwi --bootstick /tmp/initrd-usbboot-suse-10.2.i686-2.1.1.gz \
     --bootstick-system /tmp/full-suse-10.2.i686-1.1.2
```

Virtual disk system (QEMU or VMware)

To be able to use a virtualization system a virtual disk needs to be created. This can be done by specifying the following type in the system's image `config.xml`:

```
<preferences>
  <type filesystem="ext3" boot="vmxboot/suse-10.2">vmx</type>
  ...
</preferences>
```

After this the system image can be created. The process will create the system image and the specified vmxboot/suse-10.2 boot image. The result is then used to create the virtual disk. The following command needs to be called:

```
kiwi --prepare full-suse-10.2 --root /tmp/mysystem
kiwi --create /tmp/mysystem -d /tmp
```

The result of the command above is a set of virtual disks, one with suffix `.qemu` (QEMU) and the other with the suffix `.vmdk` (VMware). To run the system on the virtual disk with for example qemu call:

```
qemu /tmp/full-suse-10.2.i686-1.1.2.qemu
```

Para Virtual Image for Xen

To be able to use an image within Xen a system image and an `initrd` file needs to be created. This can be done by specifying the following type in the system's image `config.xml`:

```
<preferences>
  <type filesystem="ext3" boot="xenboot/suse-10.2">xen</type>
  ...
</preferences>
```

After this the system image can be created. The process will create the system image and the specified xenboot/suse-10.2 boot image. Additionally an appropriate Xen configuration file will be created. The config file will use the `.xenconfig` suffix. The following command needs to be called:

```
kiwi --prepare full-suse-10.2 --root /tmp/mysystem
kiwi --create /tmp/mysystem -d /tmp
```

To run the system within Xen one need to call:

```
xm create -c /tmp/full-suse-10.2.i686-1.1.2.xenconfig
```

Live CD system

Normally an image will be installed on a disk or into the main memory of a computer. This is done by a deployment architecture which transfers the image via a boot image into its final destination. such a boot image can also exist on a CD. The task of the CD-Boot image is to setup a system which is divided into two parts:

- Read-Only part which is obtained from the CD
- Read/Write part which is pushed into main memory

An image which works partially on disk and partially on RAM is called a live CD system. The CD-Boot structure of KIWI will put the directories `/bin`, `/boot`, `/lib`, `/opt`, `/sbin` and `/usr` on the CD and the rest into the main memory of the system.

How to Setup an Image as Live CD

To create an .iso image which can be burned on CD you only need to specify the boot image which should handle your live system. This is done by setting the `type` of the image in the `config.xml` file as follows:

```
<preferences>
  <type boot="isoboot/suse-10.3">iso</type>
  ...
</preferences>
```

The attribute `boot` refers to the CD boot image which must exist in `/usr/share/kiwi/image/isoboot`. Like for all boot images the most important point is that the boot image has to match the operating system image.

This means the kernel of the boot- and operating system image must be the same. If there is no boot image which matches you need to create your own boot image description. A good starting point for this is to use an existing boot image and adapt it to your needs.

How does the Boot Image Work

The boot process from a CD works in five steps:

1. After the image has been prepared which means a physical extend exists the process of building the logical extend will split the physical part into two physical extends:
   - read/write extend which is loaded into RAM
   - read-only extend consisting of the data stored in the `/bin`, `/boot`, `/lib`, `/opt`, `/sbin` and `/usr` directories
2. From the two physical extends kiwi will create two `ext2` based logical images.
3. Next kiwi prepares and creates in one step the boot image given as the type of the image configuration file `config.xml`.
4. Next kiwi will setup the CD directory structure and copy all image and kernel files as well as the isolinux data into this directory tree.
5. At last kiwi calls the `isolinux.sh` script provided by the ISO boot image to create an .iso file from the CD directory tree.

5.7. Troubleshooting

TODO