
KIWI NG Documentation

Release 9.20.5

Marcus Schäfer

Jul 14, 2020

CONTENTS

1 Quick Start	2
2 Installation	4
3 Overview	7
4 Concept and Workflow	11
5 Working with Images	46
6 Building Images	74
7 Command Line	110
8 Contributing	122
9 Image Description XML Schema	135
10 Python API	136
11 The Appliance Concept	300
12 Use Cases	301
13 Contact	302
Python Module Index	303
Index	306

Note: KIWI NG vs. KIWI Legacy

This documentation covers KIWI Next Generation (KIWI NG) 9.20.5- the command line utility to build Linux system appliances. KIWI NG is stable and all new features, bugfixes, and improvements will be developed here. Versions older or equal to v7.x.x are out of maintenance and do not get any updates or bugfixes. If you still need this version, refer to the documentation for [KIWI Legacy](#)

QUICK START

Note: Abstract

This document describes how to start working with KIWI NG, an OS appliance builder. This description applies for version 9.20.5.

1.1 Before you start

1. Install KIWI NG first, either via your distributions' package manager (see [Installation](#)) or via:

```
$ pip install kiwi
```

2. Clone the repository containing example appliances (see [Example Appliance Descriptions](#)):

```
$ git clone https://github.com/OSInside/kiwi-descriptions
```

1.2 Choose a First Image

Take a look which images are available in the example appliances repository and select one that matches your desired image as close as possible. Or just use the one given in the examples below.

1.3 Build your First Image

Your first image will be a simple system disk image which can run in any full virtualization system like QEMU. Invoke the following KIWI NG command in order to build it:

```
$ sudo kiwi-ng --type vmx system build \  
    --description kiwi-descriptions/suse/x86_64/suse-leap-15.1-JeOS_\  
→ \  
    --target-dir /tmp/myimage
```

The resulting image will be placed into the folder `/tmp/myimage` with the suffix `.raw`.

If you don't wish to create a openSUSE Leap 15.1 image, substitute the folder following the `--description` option with another folder that contains the image description which you selected.

1.4 Run your Image

Running an image actually means booting the operating system. In order to do that attach the disk image to a virtual system. In this example we use QEMU and boot it as follows:

```
$ qemu -boot c \  
    -drive file=LimeJeOS-Leap-15.1.x86_64-1.15.1.raw,format=raw,  
→ if=virtio \  
    -m 4096
```

1.5 Tweak and Customize your Image

Now that you have successfully built and started your first image, you can start tweaking it to match your needs.

Find the documentation of the appliance description files in the following sections.

INSTALLATION

Note: This document describes how to install KIWI NG. Apart from the preferred method to install KIWI NG via rpm, it is also available on [pypi](#) and can be installed via pip.

2.1 Installation from OBS

The most up to date packages of KIWI NG can be found on the Open Build Service in the [Virtualization:Appliances:Builder](#) project.

To install KIWI NG, follow these steps:

1. Open the URL <https://download.opensuse.org/repositories/Virtualization:/Appliances:/Builder> in your browser.
2. Right-click on the link of your preferred operating system and copy the URL. In Firefox it is the menu *Copy link address*.
3. Insert the copied URL from the last step into your shell. The `DIST` placeholder contains the respective distribution. Use **zypper addrepo** to add it to the list of your repositories:

```
$ sudo zypper addrepo http://download.opensuse.org/repositories/  
→Virtualization:/Appliances:/Builder/<DIST> appliance-builder
```

If your distribution is not using **zypper**, please use your package manager's appropriate command instead. For **dnf** that is:

```
$ sudo dnf config-manager --add-repo https://download.opensuse.  
→org/repositories/Virtualization:/Appliances:/Builder/<DIST>/  
→Virtualization:Appliances:Builder.repo
```

4. Add the repositories' signing-key to your package manager's database. For rpm run:

```
$ sudo rpm --import https://build.opensuse.org/projects/  
→Virtualization:Appliances:Builder/public_key
```

And verify that you got the correct key:


```
$ rpm -qi gpg-pubkey-74cbe823-* | gpg2
gpg: WARNING: no command supplied.  Trying to guess what you
↪mean ...
pub     dsa1024 2009-05-04 [SC] [expires: 2020-10-09]
        F7E82012C74FD0B85F5334DC994B195474CBE823
uid           Virtualization:Appliances OBS Project
        ↪<Virtualization:Appliances@build.opensuse.org>
```

5. Install KIWI NG:

```
$ sudo zypper in python3-kiwi
```

2.2 Installation from your distribution's repositories

Note: There are many packages that contain the name KIWI NG in their name, some of these are even python packages. Please double check the packages' description whether it is actually the KIWI NG Appliance builder before installing it.

Some Linux distributions ship KIWI NG in their official repositories. These include openSUSE Tumbleweed, openSUSE Leap, and Fedora since version 28. Note, these packages tend to not be as up to date as the packages from OBS, so some features described here might not exist yet.

To install KIWI NG on openSUSE, run the following command:

```
$ sudo zypper install python3-kiwi
```

On Fedora, use the following command instead:

```
$ sudo dnf install kiwi-cli
```

2.3 Installation from PyPI

KIWI NG can be obtained from the Python Package Index (PyPi) via Python's package manager pip:

```
$ pip install kiwi
```


2.4 Example Appliance Descriptions

There is a GitHub project hosting example appliance descriptions to be used with the next generation KIWI NG. Users who need an example to start with should clone the project as follows:

```
$ git clone https://github.com/OSInside/kiwi-descriptions
```


OVERVIEW

Note: Abstract

This document provides a conceptual overview about the steps of creating an image with KIWI NG. It also explains the terminology regarding the concept and process when building system images with KIWI NG 9.20.5.

3.1 Basic Workflow

Note: Abstract

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

This description applies for version 9.20.5.

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

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

- can be deployed inside an existing virtual environment without requiring configuration at start-up.
- automatically configure themselves in a known target environment.

- prompt the user for an interactive system configuration.

The image creation process with KIWI NG is automated and does not require any user interaction. The information required for the image creation process is provided by the primary configuration file named `config.xml`. This file is validated against the schema documented in [Schema Documentation](#) section. In addition, the image can optionally be customized using the `config.sh` and `images.sh` scripts and by using an *overlay tree (directory)* called `root`. See [Components of an Image Description](#) section for further details.

Note: Previous Knowledge

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

3.1.1 Components of an Image Description

A KIWI NG image description can be composed by several parts. The main part is the KIWI NG description file itself (named `config.xml` or an arbitrary name plus the `*.kiwi` extension). The configuration XML is the only required component, others are optional.

These are the optional components of an image description:

1. `config.sh` shell script

Is the configuration shell script that runs at the end of the *prepare step* if present. It can be used to fine tune the unpacked image.

Note that the script is directly invoked by the operating system if its executable bit is set. Otherwise it is called by `bash` instead.

2. `images.sh` shell script

Is the configuration shell script that runs at the beginning of the create step. So it is expected to be used to handle image type specific tasks. It is called in a similar fashion as `config.sh`

3. Overlay tree directory

The *overlay tree* is a folder (called `root`) or a tarball file (called `root.tar.gz`) that contains files and directories that will be copied to the target image build tree during the *prepare step*. It is executed after all the packages included in the `config.xml` file have been installed. Any already present file is overwritten.

4. CD root user data

For live ISO images and install ISO images an optional `cdroot` archive is supported. This is a tar archive matching the name `config-cdroot.tar[.compression_postfix]`. If present it will be unpacked as user data on the ISO image. This is mostly useful to add e.g. license files or user documentation on the CD/DVD which can be read directly without booting from the media.

5. Archives included in the `config.xml` file.

The archives that are included in the `<packages>` using the `<archive>` subsection:

```
<packages type="image">
  <archive name="custom-archive.tgz"/>
</packages>
```

3.2 Conceptual Overview

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

KIWI NG creates images in a two step process:

1. The first step, the *prepare operation*, generates a so-called *unpacked image tree* (directory) using the information provided in the image description.
2. The second step, the *create operation*, creates the *packed image* or *image* in the specified format based on the unpacked image and the information provided in the configuration file.

The image creation process with KIWI NG is automated and does not require any user interaction. The information required for the image creation process is provided by the image description.

3.3 Terminology

Appliance An appliance is a ready to use image of an operating system including a pre-configured application for a specific use case. The appliance is provided as an image file and needs to be deployed to, or activated in the target system or service.

Image The result of a KIWI NG build process.

Image Description Specification to define an appliance. The image description is a collection of human readable files in a directory. At least one XML file `config.xml` or `.kiwi` is required. In addition there may be as well other files like scripts or configuration data. These can be used to customize certain parts either of the KIWI NG build process or of the initial start-up behavior of the image.

Overlay Files A directory structure with files and subdirectories stored as part of the Image Description. This directory structure is packaged as a file `root.tar.gz` or stored inside a directory named `root`. The content of the directory structure is copied on top of the existing file system (overlayed) of the appliance root. This also includes permissions and attributes as a supplement.

KIWI NG An OS appliance builder.

Virtualization Technology Software simulated computer hardware. A virtual machine acts like a real computer, but is separated from the physical hardware. Within this documentation the QEMU virtualization system is used. Another popular alternative is Virtualbox.

3.4 System Requirements

To use and run KIWI NG, you need:

- A recent Linux distribution, see *Supported Distributions* for details. Alternatively a Linux distribution which supports the docker container system, where KIWI NG can be run inside a container, see: *Building in a Self-Contained Environment*
- Enough free disk space to build and store the image. We recommend a minimum of 15GB.
- Python version 3.4 or higher
- Git (package `git`) to clone a repository.
- Any virtualization technology to start the image. We recommend QEMU.

CONCEPT AND WORKFLOW

Note: Abstract

The following sections describe the concept and general workflow of building appliances with KIWI NG 9.20.5.

4.1 The Image Description

The image description is a XML file that defines properties of the appliance that will be build by KIWI NG, for example:

- image type (e.g. QEMU disk image, PXE bootable image, Vagrant box, etc.)
- partition layout
- packages to be installed on the system
- users to be added

The following sections will walk you through the major elements and attributes of the RELAX NG schema¹. A complete description of the schema can be found in *Image Description XML Schema*.

We will follow the standard nomenclature when addressing components of the XML file:

- An *element* is a XML “tag”: `<example/>`, which we address by the name *example*.
- Elements can have *attributes* which take values: `<example attr1="val1" attr2=val2"/>`.
- Elements can have *children*:

```
<element>
  <child/>
</element>
```

¹ RELAX NG is a so-called schema language: it describes the structure of a XML document.

- Some elements have a *content*: `<element_with_content>CONTENT`, while others are *empty-element tags*: `<empty_element/>`.

4.2 The image Element

The image description consists of the root element `image` and its children, for example:

```
<?xml version="1.0" encoding="utf-8"?>

<image schemaversion="7.1" name="LimeJeOS-Leap-15.1">
  <!-- all settings belong here -->
</image>
```

The image element requires the following two attributes (as shown in the above example):

- `name`: A name for this image that must not contain spaces or `/`.
- `schemaversion`: The used version of the RNG schema. KIWI NG will automatically convert your image description from an older schema version to the most recent one (it will perform this only internally and won't modify your `config.xml`). If in doubt, use the latest schema version.

The `name` attribute will be used to create the bootloader entry, however it can be inconvenient to use as it must be POSIX-safe. You can therefore provide an alternative name that will be displayed in the bootloader via the attribute `displayName`, which doesn't have the same strict rules as `name` (it can contain spaces and slashes):

```
<?xml version="1.0" encoding="utf-8"?>

<image schemaversion="7.1" name="LimeJeOS-Leap-15.1" displayName=
↳ "LimeJeOS-Leap-15.1">
  <!-- all setting belong here -->
</image>
```

4.3 The description Element

The description element, contains some high level information about the image:

```
<image schemaversion="7.1" name="LimeJeOS-Leap-15.1">
  <description type="system">
    <author>Jane Doe</author>
    <contact>jane@myemaildomain.xyz</contact>
    <specification>
      LimeJeOS-Leap-15.1, a small image
    </specification>
    <license>GPLv3</license>
  </description>
```

(continues on next page)

(continued from previous page)

```
<!-- snip -->
</image>
```

The `description` element must always contain a `type` attribute. This attribute accepts the values `system` or `boot`. The value `boot` is used by the KIWI NG developers and is not relevant for the end user, thus `type` should be always set to `system`.

`description` allows the following optional children:

- `author`: The name of the author of this image.
- `contact`: Some means how to contact the author of the image (e.g. an email address, an IM nickname and network, etc.)
- `specification`: A detailed description of this image, e.g. its use case.
- `license`: If applicable, you can specify a license for the image.

4.4 The preferences Element

The mandatory `preferences` element contains the definition of the various enabled image types (so-called build types). Each of these build types can be supplied with attributes specific to that image type, which we described in the section *Build Types*.

4.4.1 Build Types

A build type defines the type of an appliance that is produced by KIWI NG, for instance, a live ISO image or a virtual machine disk.

For example, a live ISO image is specified as follows:

```
<image schemaversion="7.1" name="LimeJeOS-Leap-15.1">
  <preferences>
    <type image="iso" primary="true" flags="overlay"
    ↪hybridpersistent_filesystem="ext4" hybridpersistent="true"/>
    <!-- additional preferences -->
  </preferences>
  <!-- additional image settings -->
</image>
```

A build type is defined via a single `type` element whose only required attribute is `image`, that defines which image type is created. All other attributes are optional and can be used to customize an image further. In the above example we created an ISO image, with the an `ext4` filesystem².

² A hybrid persistent filesystem contains a copy-on-write file to keep data persistent over a reboot.

It is possible to provide **multiple** `type` elements with **different** image attributes inside the preferences section. The following XML snippet can be used to create a live image, an OEM installation image, and a virtual machine disk of the same appliance:

```
<image schemaversion="7.1" name="LimeJeOS-Leap-15.1">
  <preferences>
    <!-- Live ISO -->
    <type image="iso" primary="true" flags="overlay"
    ↪hybridpersistent_filesystem="ext4" hybridpersistent="true"/>

    <!-- Virtual machine -->
    <type image="vmx" filesystem="ext4" bootloader="grub2"
    ↪kernelcmdline="splash" firmware="efi"/>

    <!-- OEM installation image -->
    <type image="oem" filesystem="ext4" initrd_system="dracut"
    ↪installiso="true" bootloader="grub2" kernelcmdline="splash"
    ↪firmware="efi">
      <oemconfig>
        <oem-systemsize>2048</oem-systemsize>
        <oem-swap>true</oem-swap>
        <oem-device-filter>/dev/ram</oem-device-filter>
        <oem-multipath-scan>>false</oem-multipath-scan>
      </oemconfig>
      <machine memory="512" guestOS="suse" HWversion="4"/>
    </type>
    <!-- additional preferences -->
  </preferences>

  <!-- additional image settings -->
</image>
```

Note the additional attribute `primary` in the Live ISO image build type. KIWI NG will by default build the image which `primary` attribute is set to `true`.

KIWI NG supports the following values for the `image` attribute (further attributes of the `type` element are documented inside the referenced sections):

- `iso`: a live ISO image, see *Build an ISO Hybrid Live Image*
- `vmx`: build a virtual machine image, see: *Build a Virtual Disk Image*
- `oem`: results in an expandable image that can be deployed via a bootable installation medium, e.g. a USB drive or a CD. See *Build an OEM Expandable Disk Image*
- `pxe`: creates an image that can be booted via PXE (network boot), see *Build a PXE Root File System Image*
- `docker`, `oci`: container images, see *Build a Docker Container Image*
- `btrfs`, `ext2`, `ext3`, `ext4`, `xfs`: KIWI NG will convert the image into a mountable filesystem of the specified type.
- `squashfs`, `clircfs`: creates the image as a filesystem that can be used in live systems

- `tbz`, `cpio`: the unpacked source tree will be compressed into a [XZ](#) or [CPIO](#) archive.

The `type` element furthermore supports the following subelements (as shown above, `oemconfig` is a subelement of `<type image="oem" ...>`):

- `containerconfig`: contains settings specific for the creation of container images, see [Build a Docker Container Image](#)
- `oemconfig`: configurations relevant for building OEM images, see: [Build an OEM Expandable Disk Image](#)
- `pxedeploy`: settings for PXE booting, see [Build a PXE Root File System Image](#)
- `vagrantconfig`: instructs KIWI NG to build a Vagrant box instead of a standard virtual machine image, see [Image Description for Vagrant](#)
- `systemdisk`: used to define LVM or Btrfs (sub)volumens, see [Custom Disk Volumes](#)
- `machine`: for configurations of the virtual machines, see [Customizing the Virtual Machine](#)
- `size`: for adjusting the size of the final image, see [Modifying the Size of the Image](#).

Common attributes of the `type` element

The `type` element supports a plethora of optional attributes, some of these are only relevant for certain build types and will be covered in the appropriate place. Certain attributes are however useful for nearly all build types and will be covered here:

- `bootloader`: Specifies the bootloader used for booting the image. At the moment `grub2`, `zipl` and `grub2_s390x_emu` (a combination of `zipl` and a userspace GRUB2) are supported. The special `custom` entry allows to skip the bootloader configuration and installation and leaves this up to the user which can be done by using the `editbootinstall` and `editbootconfig` custom scripts.
- `boottimeout`: Specifies the boot timeout in seconds prior to launching the default boot option. By default the timeout is set to 10 seconds. It makes sense to set this value to 0 for images intended to be started non-interactively (e.g. virtual machines).
- `bootpartition`: Boolean parameter notifying KIWI NG whether an extra boot partition should be used or not (the default depends on the current layout). This will override KIWI NG's default layout.
- `btrfs_quota_groups`: Boolean parameter to activate filesystem quotas if the filesystem is `btrfs`. By default quotas are inactive.
- `btrfs_root_is_snapshot`: Boolean parameter that tells KIWI NG to install the system into a btrfs snapshot. The snapshot layout is compatible with `snapper`. By default snapshots are turned off.
- `btrfs_root_is_readonly_snapshot`: Boolean parameter notifying KIWI NG that the btrfs root filesystem snapshot has to be made read-only. If this option is set to true, the root filesystem snapshot will be turned into read-only mode, once all data has been

placed to it. The option is only effective if `btrfs_root_is_snapshot` is also set to true. By default the root filesystem snapshot is writable.

- `compressed`: Specifies whether the image output file should be compressed or not. This option is only used for filesystem only images or for the `pxe` or `cpio` types.
- `editbootconfig`: Specifies the path to a script which is called right before the bootloader is installed. The script runs relative to the directory which contains the image structure.
- `editbootinstall`: Specifies the path to a script which is called right after the bootloader is installed. The script runs relative to the directory which contains the image structure.
- `filesystem`: The root filesystem, the following file systems are supported: `btrfs`, `ext2`, `ext3`, `ext4`, `squashfs` and `xfs`.
- `firmware`: Specifies the boot firmware of the appliance, supported options are: `bios`, `ec2`, `efi`, `uefi`, `ofw` and `opal`. This attribute is used to differentiate the image according to the firmware which boots up the system. It mostly impacts the disk layout and the partition table type. By default `bios` is used on x86, `ofw` on PowerPC and `efi` on ARM.
- `force_mbr`: Boolean parameter to force the usage of a MBR partition table even if the system would default to GPT. This is occasionally required on ARM systems that use a EFI partition layout but which must not be stored in a GPT. Note that forcing a MBR partition table incurs limitations with respect to the number of available partitions and their sizes.
- `fsmountoptions`: Specifies the filesystem mount options which are passed via the `-o` flag to **mount** and are included in `/etc/fstab`.
- `fscreateoptions`: Specifies the filesystem options used to create the filesystem. In KIWI NG the filesystem utility to create a filesystem is called without any custom options. The default options are filesystem specific and are provided along with the package that provides the filesystem utility. For the Linux `ext[234]` filesystem, the default options can be found in the `/etc/mke2fs.conf` file. Other filesystems provides this differently and documents information about options and their defaults in the respective manual page, e.g **man mke2fs**. With the `fscreateoptions` attribute it's possible to directly influence how the filesystem will be created. The options provided as a string are passed to the command that creates the filesystem without any further validation by KIWI NG. For example, to turn off the journal on creation of an `ext4` filesystem the following option would be required:

```
<type fscreateoptions="-O ^has_journal"/>
```

- `kernelcmdline`: Additional kernel parameters passed to the kernel by the bootloader.
- `luks`: Supplying a value will trigger the encryption of the partitions using the LUKS extension and using the provided string as the password. Note that the password must be entered when booting the appliance!
- `primary`: Boolean option, KIWI NG will by default build the image which `primary`

attribute is set to `true`.

- `target_blocksize`: Specifies the image blocksize in bytes which has to match the logical blocksize of the target storage device. By default 512 Bytes is used, which works on many disks. You can obtain the blocksize from the `SSZ` column in the output of the following command:

```
blockdev --report $DEVICE
```

4.4.2 Common Elements

Now that we have covered the `type` element, we shall return to the remaining child-elements of `preferences`:

- `version`: A version number of this image. We recommend to use the following format: **Major.Minor.Release**, however other versioning schemes are possible, e.g. one can use the version of the underlying operating system.
- `packagemanager`: Specify the package manager that will be used to download and install the packages for your appliance. Currently the following package managers are supported: `apt-get`, `zypper` and `dnf`. Note that the package manager must be installed on the system **calling** KIWI NG, it is **not** sufficient to install it inside the appliance.
- `locale`: Specify the locale that the resulting appliance will use.
- `timezone`: Override the default timezone of the image to a more suitable value, e.g. the timezone in which the image's users reside.
- `rpm-check-signatures`: Boolean value that defines whether the signatures of the downloaded RPM packages will be verified before installation. Note that when building appliances for a different distribution you will have to either import the other distribution's signing-key or set this to `false` (RPM will otherwise fail to verify the package signatures, as it does will not trust the signature key of other distributions or even other versions of the same distribution).
- `rpm-excludedocs`: Boolean value that instructs RPM whether to install documentation with packages or not. Please bear in mind that enabling this can have quite a negative impact on user-experience and should thus be used with care.
- `bootloader-theme` and `bootsplash-theme`: themes for the bootloader and the bootsplash-screen. These themes have to be either built-in to the bootloader or installed via the `packages` section.

An example excerpt from a image description using these child-elements of `preferences`, results in the following image description:

```
<image schemaversion="7.1" name="LimeJeOS-Leap-15.1">
  <!-- snip -->
  <preferences>
    <version>15.0</version>
    <packagemanager>zypper</packagemanager>
```

(continues on next page)

(continued from previous page)

```

    <locale>en_US</locale>
    <keytable>us</keytable>
    <timezone>Europe/Berlin</timezone>
    <rpm-excludedocs>true</rpm-excludedocs>
    <rpm-check-signatures>>false</rpm-check-signatures>
    <bootsplash-theme>openSUSE</bootsplash-theme>
    <bootloader-theme>openSUSE</bootloader-theme>
    <type image="vmx" filesystem="ext4" format="qcow2"
    ↪boottimeout="0" bootloader="grub2">
    </preferences>
    <!-- snip -->
</image>

```

4.5 Image Profiles

In the previous section we have covered build types, that are represented in the image description as the `type` element. We have also shown how it is possible to include multiple build types in the same appliance. Unfortunately that approach has one significant limitation: one can only include multiple build types with **different** settings for the attribute `image`.

In certain cases this is undesirable, for instance when building multiple very similar virtual machine disks. Then one would have to duplicate the whole `config.xml` for each virtual machine. KIWI NG supports *profiles* to work around this issue.

A *profile* is a namespace for additional settings that can be applied by KIWI NG on top of the default settings (or other profiles), thereby allowing to build multiple appliances with the same build type but with different configurations.

In the following example, we create two virtual machine images: one for QEMU (using the `qcow2` format) and one for VMWare (using the `vmdk` format).

```

<image schemaversion="7.1" name="LimeJeOS-Leap-15.1">
  <!-- snip -->
  <profiles>
    <profile name="QEMU" description="virtual machine for QEMU"/
    ↪
    <profile name="VMWare" description="virtual machine for
    ↪VMWare"/>
  </profiles>
  <preferences>
    <version>15.0</version>
    <packagemanager>zypper</packagemanager>
  </preferences>
  <preferences profiles="QEMU">
    <type image="vmx" format="qcow2" filesystem="ext4"
    ↪bootloader="grub2">
  </preferences>

```

(continues on next page)

(continued from previous page)

```

<preferences profiles="VMWare">
  <type image="vmx" format="vmdk" filesystem="ext4"
  ↪bootloader="grub2">
  </preferences>
  <!-- snip -->
</image>

```

Each profile is declared via the element `profile`, which itself must be a child of `profiles` and must contain the `name` and `description` attributes. The `description` is only present for documentation purposes, `name` on the other hand is used to instruct KIWI NG which profile to build via the command line. Additionally, one can provide the boolean attribute `import`, which defines whether this profile should be used by default when KIWI NG is invoked via the command line.

A profile inherits the default settings which do not belong to any profile. It applies only to elements that contain the profile in their `profiles` attribute. The attribute `profiles` expects a comma separated list of profiles for which the settings of this element apply. The attribute is present in the following elements only:

- `preferences`
- `drivers`
- `repository` and `packages` (see *Defining Repositories and Adding or Removing Packages*)
- `users`

Profiles can furthermore inherit settings from another profile via the `requires` sub-element:

```

<profiles>
  <profile name="VM" description="virtual machine"/>
  <profile name="QEMU" description="virtual machine for QEMU">
    <requires profile="VM"/>
  </profile>
</profiles>

```

The profile `QEMU` would inherit the settings from `VM` in the above example.

We cover the usage of *profiles* when invoking KIWI NG and when building in the Open Build Service in *Building Images with Profiles*.

4.6 Adding Users

User accounts can be added or modified via the `users` element, which supports a list of multiple user child elements:

```
<image schemaversion="7.1" name="LimeJeOS-Leap-15.1">
  <users>
    <user
      password="this_is_soo_insecure"
      home="/home/me" name="me"
      groups="users" pwdformat="plain"
    />
    <user
      password="$1$wYJUgpM5$RXMMeASDc035eX.NbYWF10"
      home="/root" name="root" groups="root"
    />
  </users>
</image>
```

Each user element represents a specific user that is added or modified. The following attributes are mandatory:

- `name`: the UNIX username
- `home`: the path to the user's home directory

Additionally, the following optional attributes can be specified:

- `groups`: A comma separated list of UNIX groups. The first element of the list is used as the user's primary group. The remaining elements are appended to the user's supplementary groups. When no groups are assigned then the system's default primary group will be used.
- `id`: The numeric user id of this account.
- `pwdformat`: The format in which password is provided, either `plain` or `encrypted` (the latter is the default).
- `password`: The password for this user account. It can be provided either in cleartext form (`pwdformat="plain"`) or in `crypt`'ed form (`pwdformat="encrypted"`). Plain passwords are discouraged, as everyone with access to the image description would know the password. It is recommended to generate a hash of your password, e.g. using the `mkpasswd` tool (available in most Linux distributions via the `whois` package):

```
$ mkpasswd -m sha-512 -S $(date +%N) -s <<< INSERT_YOUR_
→PASSWORD_HERE
```

The `users` element furthermore accepts a list of profiles (see *Image Profiles*) to which it applies via the `profiles` attribute, as shown in the following example:

```
<image schemaversion="7.1" name="LimeJeOS-Leap-15.1">
  <profiles>
```

(continues on next page)

(continued from previous page)

```

    <profile name="VM" description="standard virtual machine"/>
    <profile name="shared_VM" description="virtual machine_
↳shared by everyone"/>
  </profiles>
  <!-- snip -->
  <users>
    <user
      password="$1$wYJUgpM5$RXMMeASDc035eX.NbYWF10"
      home="/root" name="root" groups="root"
    />
  </users>
  <users profiles="VM">
    <user
      password="$1$b1ab1ab1$FRTFJZxMPfM6LA1g0EZ5h1"
      home="/home/devel" name="devel"
    />
  </users>
  <users profiles="shared_VM">
    <user
      password="super_secr4t" pwdformat="plain"
      home="/share/devel" name="devel" groups="users,devel"
    />
  </users>
</image>

```

Here the settings for the root user are shared among all appliances. The configuration of the `devel` user on the other hand depends on the profile.

4.7 Defining Repositories and Adding or Removing Packages

A crucial part of each appliance is the package and repository selection. KIWI NG allows the end user to completely customize the selection of repositories and packages via the `repository` and `packages` elements.

4.7.1 Adding repositories

KIWI NG installs packages into your appliance from the repositories defined in the image description. Therefore at least one repository **must** be defined, as KIWI NG will otherwise not be able to fetch any packages.

A repository is added to the description via the `repository` element, which is a child of the top-level `image` element:


```
<image schemaversion="7.1" name="LimeJeOS-Leap-15.1">
  <!-- snip -->
  <repository type="rpm-md" alias="kiwi" priority="1">
    <source path="obs://Virtualization:Appliances:Builder/
↪openSUSE_Leap_15.1"/>
  </repository>
  <repository type="rpm-md" alias="OS" imageinclude="true">
    <source path="obs://openSUSE:Leap:15.1/standard"/>
  </repository>
</image>
```

In the above snippet we defined two repositories:

1. The repository belonging to the KIWI NG project: *obs://Virtualization:Appliances:Builder/openSUSE_Leap_15.1* at the Open Build Service (OBS)
2. The RPM repository belonging to the OS project: *obs://openSUSE:Leap:15.1/standard*, at the Open Build Service (OBS). The translated http URL will also be included in the final appliance.

The `repository` element accepts one `source` child element, which contains the URL to the repository in an appropriate format and the following optional attributes:

- `type`: repository type, accepts one of the following values: `apt-deb`, `apt-rpm`, `deb-dir`, `mirrors`, `rpm-dir`, `rpm-md`. For ordinary RPM repositories use `rpm-md`, for ordinary APT repositories `apt-deb`.
- `imageinclude`: Specify whether this repository should be added to the resulting image, defaults to `false`.
- `imageonly`: A repository with `imageonly="true"` will not be available during image build, but only in the resulting appliance. Defaults to `false`.
- `priority`: An integer priority for all packages in this repository. If the same package is available in more than one repository, then the one with the highest priority is used.
- `alias`: Name to be used for this repository, it will appear as the repository's name in the image, which is visible via `zypper repos` or `dnf repolist`. KIWI NG will construct an alias from the path in the `source` child element (replacing each `/` with a `_`), if no value is given.
- `repository_gpgcheck`: Specify whether or not this specific repository is configured to run repository signature validation. If not set, the package manager's default is used.
- `package_gpgcheck`: Boolean value that specifies whether each package's GPG signature will be verified. If omitted, the package manager's default will be used.
- `components`: Distribution components used for `deb` repositories, defaults to `main`.
- `distribution`: Distribution name information, used for `deb` repositories.
- `profiles`: List of profiles to which this repository applies.

Supported repository paths

The actual location of a repository is specified in the `source` child element of `repository` via its only attribute `path`. KIWI NG supports the following paths types:

- `http://URL` or `https://URL` or `ftp://URL`: a URL to the repository available via HTTP(s) or FTP.
- `obs://$PROJECT/$REPOSITORY`: evaluates to the repository `$REPOSITORY` of the project `$PROJECT` available on the Open Build Service (OBS). By default KIWI NG will look for projects on build.opensuse.org, but this can be overridden using the runtime configuration file (see *The Runtime Configuration File*). Note that it is not possible to add repositories using the `obs://` path from **different** OBS instances (use direct URLs to the `.repo` file instead in this case).
- `obsrepositories://`: special path only available for builds using the Open Build Service. The repositories configured for the OBS project in which the KIWI NG image resides will be available inside the appliance. This allows you to configure the repositories of your image from OBS itself and not having to modify the image description.
- `dir:///path/to/directory` or `file:///path/to/file`: an absolute path to a local directory or file available on the host building the appliance.
- `iso:///path/to/image.iso`: the specified ISO image will be mounted during the build of the KIWI NG image and a repository will be created pointing to the mounted ISO.

4.7.2 Adding and removing packages

Now that we have defined the repositories, we can define which packages should be installed on the image. This is achieved via the `packages` element which includes the packages that should be installed, ignore or removed via individual `package` child elements:

```
<image schemaversion="7.1" name="LimeJeOS-Leap-15.1">
  <packages type="bootstrap">
    <package name="udev"/>
    <package name="filesystem"/>
    <package name="openSUSE-release"/>
    <!-- additional packages installed before the chroot is
→created -->
  </packages>
  <packages type="image">
    <package name="patterns-openSUSE-base"/>
    <!-- additional packages to be installed into the chroot -->
  </packages>
</image>
```

The `packages` element provides a collection of different child elements that instruct KIWI NG when and how to perform package installation or removal. Each `packages` element acts as a group, whose behavior can be configured via the following attributes:

- `type`: either `bootstrap`, `image`, `delete`, `uninstall` or one of the following build types: `docker`, `iso`, `oem`, `pxe`, `vmx`, `oci`.

Packages for `type="bootstrap"` are pre-installed to populate the images' root file system before chrooting into it.

Packages in `type="image"` are installed immediately after the initial chroot into the new root file system.

Packages in `type="delete"` and `type="uninstall"` are removed from the image, for details see *Uninstall System Packages*.

And packages which belong to a build type are only installed when that specific build type is currently processed by KIWI NG.

- `profiles`: a list of profiles to which this package selection applies (see *Image Profiles*).
- `patternType`: selection type for patterns, supported values are: `onlyRequired`, `plusRecommended`, see: *The product and namedCollection element*.

We will describe the different child elements of packages in the following sections.

The package element

The package element represents a single package to be installed (or removed), whose name is specified via the mandatory `name` attribute:

```
<image schemaversion="7.1" name="LimeJeOS-Leap-15.1">
  <!-- snip -->
  <packages type="bootstrap">
    <package name="udev"/>
  </packages>
</image>
```

which adds the package `udev` to the list of packages to be added to the initial filesystem. Note, that the value that you pass via the `name` attribute is passed directly to the used package manager. Thus, if the package manager supports other means how packages can be specified, you may pass them in this context too. For example, RPM based package managers (like **dnf** or **zypper**) can install packages via their `Provides`:. This can be used to add a package that provides a certain capability (e.g. `Provides: /usr/bin/my-binary`) via:

```
<image schemaversion="7.1" name="LimeJeOS-Leap-15.1">
  <!-- snip -->
  <packages type="bootstrap">
    <package name="/usr/bin/my-binary"/>
  </packages>
</image>
```

Whether this works depends on the package manager and on the environment that is being used. In the Open Build Service, certain `Provides` either are not visible or cannot be properly extracted from the KIWI NG description. Therefore, relying on `Provides` is not recommended.

Packages can also be included only on specific architectures via the `arch` attribute. KIWI NG compares the `arch` attributes value with the output of `uname -m`.

```
<image schemaversion="7.1" name="LimeJeOS-Leap-15.1">
  <!-- snip -->
  <packages type="image">
    <package name="grub2"/>
    <package name="grub2-x86_64-efi" arch="x86_64"/>
    <package name="shim" arch="x86_64"/>
  </packages>
</image>
```

which results in `grub2-x86_64-efi` and `shim` being only installed on 64 Bit images, but GRUB2 also on 32 Bit images.

The archive element

It is sometimes necessary to include additional packages into the image which are not available in the package manager's native format. KIWI NG supports the inclusion of ordinary archives via the `archive` element, whose `name` attribute specifies the filename of the archive (KIWI NG looks for the archive in the image description folder).

```
<packages type="image">
  <archive name="custom-program1.tgz"/>
  <archive name="custom-program2.tar"/>
</packages>
```

KIWI NG will extract the archive into the root directory of the image using [GNU tar](#), thus only archives supported by it can be included. When multiple `archive` elements are specified then they will be applied in a top to bottom order. If a file is already present in the image, then the file from the archive will overwrite it (same as with the image overlay).

Uninstall System Packages

KIWI NG supports two different methods how packages can be removed from the appliance:

1. Packages present as a child element of `<packages type="uninstall">` will be gracefully uninstalled by the package manager alongside with dependent packages and orphaned dependencies.
2. Packages present as a child element of `<packages type="delete">` will be removed by RPM/DPKG without any dependency check, thus potentially breaking dependencies and compromising the underlying package database.

Both types of removals take place after `config.sh` is run in the *prepare step* (see also *User Defined Scripts*).

Warning: An `uninstall` packages request deletes:

- the listed packages,
- the packages dependent on the listed ones, and
- any orphaned dependency of the listed packages.

Use this feature with caution as it can easily cause the removal of sensitive tools leading to failures in later build stages.

Removing packages via `type="uninstall"` can be used to completely remove a build time tool (e.g. a compiler) without having to specify all dependencies of that tool (as one would have when using `type="delete"`). Consider the following example where we wish to compile a custom program in `config.sh`. We ship its source code via an archive element and add the build tools (ninja, meson and clang) to `<packages type="image">` and `<packages type="uninstall">`:

```
<image schemaversion="7.1" name="LimeJeOS-Leap-15.1">
  <!-- snip -->
  <packages type="image">
    <package name="ca-certificates"/>
    <package name="coreutils"/>
    <package name="ninja"/>
    <package name="clang"/>
    <package name="meson"/>
    <archive name="foo_app_sources.tar.gz"/>
  </packages>
  <!-- These packages will be uninstalled after running config.sh -->
  <packages type="uninstall">
    <package name="ninja"/>
    <package name="meson"/>
    <package name="clang"/>
  </packages>
</image>
```

The tools `meson`, `clang` and `ninja` are then available during the *prepare step* and can thus be used in `config.sh` (for further details, see *User Defined Scripts*), for example to build `foo_app`:

```
pushd /opt/src/foo_app
mkdir build
export CC=clang
meson build
cd build && ninja && ninja install
popd
```

The `<packages type="uninstall">` element will make sure that the final appliance will no longer contain our tools required to build `foo_app`, thus making our image smaller.

There are also other use cases for `type="uninstall"`, especially for specialized appliances. For containers one can often remove the package shadow (it is required to setup new user accounts) or any left over partitioning tools (`parted` or `fdisk`). All networking tools can be safely uninstalled in images for embedded devices without a network connection.

The `product` and `namedCollection` element

KIWI NG supports the inclusion of openSUSE products or of `namedCollections` (*patterns* in SUSE based distributions or *groups* for RedHat based distributions). These can be added via the `product` and `namedCollection` child elements, which both take the mandatory `name` attribute and the optional `arch` attribute.

`product` and `namedCollection` can be utilized to shorten the list of packages that need to be added to the image description tremendously. A named pattern, specified with the `namedCollection` element is a representation of a predefined list of packages. Specifying a pattern will install all packages listed in the named pattern. Support for patterns is distribution specific and available in SLES, openSUSE, CentOS, RHEL and Fedora. The optional `patternType` attribute on the `packages` element allows you to control the installation of dependent packages. You may assign one of the following values to the `patternType` attribute:

- `onlyRequired`: Incorporates only patterns and packages that the specified patterns and packages require. This is a “hard dependency” only resolution.
- `plusRecommended`: Incorporates patterns and packages that are required and recommended by the specified patterns and packages.

The `ignore` element

Packages can be explicitly marked to be ignored for installation inside a `packages` collection. This is useful to exclude certain packages from being installed when using patterns with `patternType="plusRecommended"` as shown in the following example:

```
<image schemaversion="7.1" name="LimeJeOS-Leap-15.1">
  <packages type="image" patternType="plusRecommended">
    <namedCollection name="network-server"/>
    <package name="grub2"/>
    <package name="kernel"/>
    <ignore name="ejabberd"/>
    <ignore name="puppet-server"/>
  </packages>
</image>
```

Packages can be marked as ignored during the installation by adding a `ignore` child element with the mandatory `name` attribute set to the package’s name. Optionally one can also specify the architecture via the `arch` similarly to *The package element*.

Warning: Adding ignore elements as children of a `<packages type="delete">` or a `<packages type="uninstall">` element has no effect! The packages will still get deleted.

4.8 User Defined Scripts

Note: Abstract

This chapter describes the usage of the user defined scripts `config.sh` and `image.sh`, which can be used to further customize an image in ways that are not possible via the image description alone.

KIWI NG supports up to two user defined scripts that it runs in the change root environment (chroot) containing your new appliance:

1. `config.sh` runs the end of the *prepare step* if present. It can be used to fine tune the unpacked image.
2. `images.sh` is executed at the beginning of the *image creation process*. It is run on the top level of the target root tree. The script is usually used to remove files that are not needed in the final image. For example, if an appliance is being built for a specific hardware, unnecessary kernel drivers can be removed using this script.

KIWI NG will execute both scripts via the operating system if their executable bit is set (in that case a shebang is mandatory) otherwise they will be invoked via the BASH.

4.8.1 Image Customization via the `config.sh` Shell Script

The KIWI NG image description allows to have an `config.sh` script in place. It can be used for changes appropriate for all images to be created from a given unpacked image (`config.sh` runs prior to the *create step*). The script should add operating system configuration files which would be otherwise added by a user driven installer, like the activation of services, creation of configuration files, preparation of an environment for a firstboot workflow, etc.

The `config.sh` script is called at the end of the *prepare step* (after users have been set and the *overlay tree directory* has been applied). If `config.sh` exits with a non-zero exit code then KIWI NG will report the failure and abort the image creation.

Find a common template for `config.sh` script below:

```
#=====
# Include functions & variables
#-----
test -f /.kconfig && . /.kconfig
test -f /.profile && . /.profile
```

(continues on next page)

(continued from previous page)

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

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

Configuration Tips

1. Stateless systemd UUIDs:

Machine ID files are created and set (/etc/machine-id, /var/lib/dbus/machine-id) during the image package installation when *systemd* and/or *dbus* are installed. Those UUIDs are intended to be unique and set only once in each deployment. KIWI NG follows the [systemd recommendations](#) and wipes any /etc/machine-id content, leaving it as an empty file. Note, this only applies to images based on a dracut initrd, it does not apply for container images.

In case this setting is also required for a non dracut based image, the same result can be achieved by removing /etc/machine-id in config.sh.

Note: Avoid interactive boot

It is important to remark that the file /etc/machine-id is set to an empty file instead of deleting it. **systemd** may trigger **systemd-firstboot** service if this file is not present, which leads to an interactive firstboot where the user is asked to provide some data.

Note: Avoid inconsistent /var/lib/dbus/machine-id

Note that /etc/machine-id and /var/lib/dbus/machine-id **must** contain the same unique ID. On modern systems /var/lib/dbus/machine-id is already a symlink to /etc/machine-id. However on older systems those might be two different files. This is the case for SLE-12 based images. If you are targeting these older operating systems, it is recommended to add the symlink creation into config.sh:

```
#=====
# Make machine-id consistent with dbus
#-----
if [ -e /var/lib/dbus/machine-id ]; then
    rm /var/lib/dbus/machine-id
```

(continues on next page)

(continued from previous page)

```
fi
ln -s /etc/machine-id /var/lib/dbus/machine-id
```

4.8.2 Image Customization via the `images.sh` Shell Script

The KIWI NG image description allows to have an optional `images.sh` bash script in place. It can be used for changes appropriate for certain images/image types on a case-by-case basis (since it runs at beginning of *create step*).

Warning: Modifications of the unpacked root tree

Keep in mind that there is only one unpacked root tree the script operates in. This means that all changes are permanent and will not be automatically restored!

The script should be designed to take over control of handling image type specific tasks. For example, if building the OEM type requires some additional packages or configurations then that can be handled in `images.sh`. Additionally, the script authors task is to check if changes performed beforehand do not interfere in a negative way if another image type is created from the same unpacked image root tree.

If `images.sh` exits with a non-zero exit code, then KIWI NG will report an error and abort the image creation.

See below a common template for `images.sh` script:

```
#####
# Include functions & variables
#-----
test -f /.kconfig && . /.kconfig
test -f /.profile && . /.profile

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

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

#####
# Exit successfully
#-----
exit 0
```


4.8.3 Functions and Variables Provided by KIWI NG

KIWI NG creates the `.kconfig` and `.profile` files to be sourced by the shell scripts `config.sh` and `images.sh`. `.kconfig` contains various helper functions which can be used to simplify the image configuration and `.profile` contains environment variables which get populated from the settings provided in the image description.

Provided Functions

The `.kconfig` file provides a common set of functions. Functions specific to SUSE Linux begin with the name `suse`, functions applicable to all Linux distributions start with the name `base`.

The following list describes all functions provided by `.kconfig`:

baseCleanMount Unmount the filesystems `/proc`, `/dev/pts`, `/sys` and `/proc/sys/fs/binfmt_misc`.

baseGetPackagesForDeletion Return the name(s) of the packages marked for deletion in the image description.

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

baseSetRunlevel {value} Set the default run level.

baseSetupUserPermissions Set the ownership of all home directories and their content to the correct users and groups listed in `/etc/passwd`.

baseStripAndKeep {list of info-files to keep} Helper function for the `baseStrip*` functions, reads the list of files to check from stdin for removing params: files which should be kept

baseStripDocs {list of docu names to keep} Remove all documentation files, except for the ones given as the parameter.

baseStripInfos {list of info-files to keep} Remove all info files, except for the one given as the parameter.

baseStripLocales {list of locales} Remove all locales, except for the ones given as the parameter.

baseStripTranslations {list of translations} Remove all translations, except for the ones given as the parameter.

baseStripMans {list of manpages to keep} Remove all manual pages, except for the ones given as the parameter.

Example:

```
baseStripMans more less
```

suseImportBuildKey Add the SUSE build keys to the RPM database.

baseStripUnusedLibs Remove libraries which are not directly linked against applications in the bin directories.

baseUpdateSysConfig {filename} {variable} {value} Update the contents of a sysconfig variable

suseConfig This function is deprecated and is a NOP.

baseSystemdServiceInstalled {service} Prints the path of the first found systemd unit or mount with name passed as the first parameter.

baseSysVServiceInstalled {service} Prints the name \${service} if a SysV init service with that name is found, otherwise it prints nothing.

baseSystemdCall {service_name} {args} Calls `systemctl ${args} ${service_name}` if a systemd unit, a systemd mount or a SysV init service with the `${service_name}` exist.

baseInsertService {servicename} Activate the given service via **systemctl**.

baseRemoveService {servicename} Deactivate the given service via **systemctl**.

baseService {servicename} {on|off} Activate or deactivate a service via **systemctl**. The function requires the service name and the value on or off as parameters.

Example to enable the sshd service on boot:

```
baseService sshd on
```

suseInsertService {servicename} Calls **baseInsertService** and exists only for compatibility reasons.

suseRemoveService {servicename} Calls **baseRemoveService** and exists only for compatibility reasons.

suseService {servicename} {on|off} Calls **baseService** and exists only for compatibility reasons.

suseActivateDefaultServices Activates the `network` and `cron` services to run at boot.

suseSetupProduct Creates the `/etc/products.d/baseproduct` link pointing to the product referenced by either `/etc/SuSE-brand` or `/etc/os-release` or the latest `prod` file available in `/etc/products.d`

suseSetupProductInformation Uses **zypper** to search for the installed product and installs all product specific packages. This function fails when **zypper** is not the appliances package manager.

Debug {message} Helper function to print the supplied 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 log the deletion.

Rpm {rpm commandline} Helper function for calling `rpm`: forwards all commandline arguments to `rpm` and logs the call.

Functions for Custom non-dracut Based Boot

KIWI NG also provides the following functions (mostly for compatibility reasons) which can be used to customize the boot process when using the custom boot option (see [Customizing the Boot Process](#)):

baseStripInitrd Removes various tools binaries and libraries which are not required to boot a SUSE system with KIWI NG. This function is not required when using the dracut initrd system and is kept for compatibility reasons.

baseStripFirmware Check all kernel modules if they require a firmware and strip out all firmware files which are not referenced by a kernel module

baseStripModules Search for updated modules and remove the old version which might be provided by the standard kernel

baseStripKernel Strips the kernel:

1. create the `vmlinux.gz` and `vmlinuz` files which are used as a fallback for the kernel extraction
2. handle `<strip type="delete">` requests. Because this information is generic not only files of the kernel are affected but also other data which are unwanted get deleted here
3. only keep kernel modules matching the `<drivers>` patterns from the kiwi boot image description
4. lookup kernel module dependencies and bring back modules which were removed but still required by other modules that were kept in the system
5. search for duplicate kernel modules due to kernel module updates and keep only the latest version
6. search for kernel firmware files and keep only those for which a kernel driver is still present in the system

suseStripKernel Removes all kernel drivers which are not listed in the drivers sections of `config.xml`.

baseStripTools `{list of toolpath} {list of tools}` Helper function for `suseStripInitrd` function parameters: `toolpath`, `tools`.

Profile Environment Variables

The `.profile` environment file is created by KIWI NG and contains a specific set of variables which are listed below.

\$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 children of the `packages` element with `type="delete"` in `config.xml`.

\$kiwi_drivers A comma separated list of the driver entries as listed in the `drivers` section of the `config.xml`.

\$kiwi_iname The name of the image as listed in `config.xml`.

\$kiwi_iversion The image version as a string.

\$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 comma separated list of profiles used to build this image.

\$kiwi_timezone The contents of the timezone setup as done in `config.xml`.

\$kiwi_type The image type as extracted from the `type` element in `config.xml`.

4.9 The Runtime Configuration File

KIWI NG supports an additional configuration file for runtime specific settings that do not belong into the image description but which are persistent and would be unsuitable for command line parameters.

The runtime configuration file must adhere to the **YAML** syntax. KIWI NG searches for the runtime configuration file in the following locations:

1. `~/.config/kiwi/config.yml`
2. `/etc/kiwi.yml`

A default runtime config file in `/etc/kiwi.yml` is provided with the `python3-kiwi` package. The file contains all settings as comments including a short description of each setting.

4.10 Customizing the Boot Process

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

Depending on the image type, KIWI NG creates the boot image automatically during the `create` step. It uses a tool called `dracut` to create this `initrd`. Dracut generated `initrd` archives can be extended by custom modules to add functionality which is not natively provided by dracut itself. In the scope of KIWI NG the following dracut modules are used:

kiwi-dump Serves as an image installer. It provides the required implementation to install a KIWI NG image on a selectable target. This module is required if one of the attributes `installiso`, `installstick` or `installpxe` is set to `true` in the image type definition

kiwi-dump-reboot Serves to boot the system into the installed image after installation is completed.

kiwi-live Boots up a KIWI NG live image. This module is required if the `iso` image type is selected

kiwi-overlay Allows to boot disk images configured with the attribute `overlayroot` set to `true`. Such a disk has its root partition compressed and readonly and boots up using overlays for the root filesystem using an extra partition on the same disk for persistent data.

kiwi-repart Resizes an OEM disk image after installation onto the target disk to meet the size constraints configured in the `oemconfig` section of the image description. The module takes over the tasks to repartition the disk, resizing of RAID, LVM, LUKS and other layers and resizing of the system filesystems.

kiwi-lib Provides functions of general use and serves as a library usable by other dracut modules. As the name implies, its main purpose is to function as library for the above mentioned kiwi dracut modules.

Note: Using Custom Boot Image Support

Apart from the standard dracut based creation of the boot image, KIWI NG supports the use of custom boot images for the image types `oem` and `pxe`. The use of a custom boot image is activated by setting the following attribute in the image description:

```
<type ... initrd_system="kiwi"/>
```

Along with this setting it is now mandatory to provide a reference to a boot image description in the `boot` attribute like in the following example:

```
<type ... boot="netboot/suse-leap15.1"/>
```

Such boot descriptions for the OEM and PXE types are currently still provided by the KIWI NG packages but will be moved into its own repository and package soon.

The custom boot image descriptions allows a user to completely customize what and how the `initrd` behaves by its own implementation. This concept is mostly used in PXE environments which are usually highly customized and requires a specific boot and deployment workflow.

4.10.1 Boot Image Hook-Scripts

The dracut `initrd` system uses `systemd` to implement a predefined workflow of services which are documented in the `bootup` man page at:

<http://man7.org/linux/man-pages/man7/dracut.bootup.7.html>

To hook in a custom boot script into this workflow it's required to provide a dracut module which is picked up by dracut at the time KIWI NG calls it. The module files can be either provided as a package or as part of the overlay directory in your image description

The following example demonstrates how to include a custom hook script right before the system rootfs gets mounted.

1. Create a subdirectory for the dracut module:

```
$ mkdir -p root/usr/lib/dracut/modules.d/90my-module
```

2. Register the dracut module in a configuration file:

```
$ vi root/etc/dracut.conf.d/90-my-module.conf

add_dracutmodules+=" my-module "
```

3. Create the hook script:

```
$ touch root/usr/lib/dracut/modules.d/90my-module/my-script.sh
```

4. Create a module setup file in `root/usr/lib/dracut/modules.d/90my-module/module-setup.sh` with the following content:

```
#!/bin/bash

# called by dracut
check() {
    # check module integrity
}

# called by dracut
depends() {
    # return list of modules depending on this one
}

# called by dracut
installkernel() {
    # load required kernel modules when needed
    instmods _kernel_module_list_
}

# called by dracut
install() {
    declare moddir=${moddir}
    inst_multiple _tools_my_module_script_needs_

    inst_hook pre-mount 30 "${moddir}/my-script.sh"
}
```

That's it! At the time KIWI NG calls dracut the 90my-module will be taken into account and is installed into the generated initrd. At boot time systemd calls the scripts as part of the `dracut-pre-mount.service`.

The dracut system offers a lot more possibilities to customize the initrd than shown in the example above. For more information, visit the [dracut project page](#).

4.10.2 Boot Image Parameters

A dracut generated initrd in a KIWI NG image build process includes one or more of the KIWI NG provided dracut modules. The following list documents the available kernel boot parameters for this modules:

rd.kiwi.debug Activates the debug log file for the KIWI NG part of the boot process at `/run/initramfs/log/boot.kiwi`.

rd.kiwi.install.pxe Tells an OEM installation image to lookup the system image on a remote location specified in `rd.kiwi.install.image`.

rd.kiwi.install.image=URI Specifies the remote location of the system image in a PXE based OEM installation

rd.kiwi.install.pass.bootparam Tells an OEM installation image to pass an additional boot parameters to the kernel used to boot the installed image. This can be used e.g. to pass on first boot configuration for a PXE image. Note, that options starting with `rd.kiwi` are not passed on to avoid side effects.

rd.kiwi.oem.maxdisk=size[KMG] Configures the maximum disk size an unattended OEM installation should consider for image deployment. Unattended OEM deployments default to deploying on `/dev/sda` (more exactly, the first device not filtered out by `oem-device-filter`). With RAID controllers, it can happen that your bunch of big JBOD disks is for example `/dev/sda` to `/dev/sdi` and the 480G RAID1 configured for OS deployment is `/dev/sdj`. With `rd.kiwi.oem.maxdisk=500G` the deployment will land on that RAID disk.

rd.live.overlay.persistent Tells a live ISO image to prepare a persistent write partition.

rd.live.overlay.cowfs Tells a live ISO image which filesystem should be used to store data on the persistent write partition.

rd.live.cowfile.mbsize Tells a live ISO image the size of the COW file in MB. When using tools like `live-grub-stick` the live ISO will be copied as a file on the target device and a GRUB loopback setup is created there to boot the live system from file. In such a case the persistent write setup, which usually creates an extra write partition on the target, will fail in almost all cases because the target has no free and unpartitioned space available. Because of that a cow file(`live_system.cow`) instead of a partition is created. The cow file will be created in the same directory the live iso image file was read from by grub and takes the configured size or the default size of 500MB.

rd.live.dir Tells a live ISO image the directory which contains the live OS root directory. Defaults to `LiveOS`.

rd.live.squashimg Tells a live ISO image the name of the squashfs image file which holds the OS root. Defaults to `squashfs.img`.

Boot Debugging

If the boot process encounters a fatal error, the default behavior is to stop the boot process without any possibility to interact with the system. Prevent this behavior by activating dracut's builtin debug mode in combination with the kiwi debug mode as follows:

```
rd.debug rd.kiwi.debug
```

This should be set at the Kernel command line. With those parameters activated, the system will enter a limited shell environment in case of a fatal error during boot. The shell contains a basic set of commands and allows for a closer look to:

```
less /run/initramfs/log/boot.kiwi
```

4.11 Legacy KIWI vs. KIWI Next Generation (KIWI NG)

Note: Abstract

Users currently have the choice for the kiwi legacy version or this next generation kiwi. This document describes the maintenance state of the legacy kiwi version and under which circumstances the use of the legacy kiwi version is required.

There is still the former [KIWI Legacy](#) version and we decided to rewrite it.

The reasons to rewrite software from scratch could be very different and should be explained in order to let users understand why it makes sense. We are receiving feedback and defect reports from a variety of groups with different use cases and requirements. It became more and more difficult to handle those requests in good quality and without regressions. At some point we asked ourselves:

Is KIWI Legacy really well prepared for future challenges?

The conclusion was that the former version has some major weaknesses which has to be addressed prior to continue with future development. The following issues are most relevant:

- Not based on a modern programming language
- Major design flaws but hardly any unit tests. The risk for regressions on refactoring is high
- No arch specific build integration tests
- Lots of legacy code for old distributions

In order to address all of these the questions came up:

How to modernize the project without producing regressions?

How to change/drop features without making anybody unhappy?

As there is no good way to achieve this in the former code base the decision was made to start a rewrite of KIWI Legacy with a maintained and stable version in the background.

Users will be able to use both versions in parallel. In addition, KIWI NG will be fully compatible with the current format of the appliance description. This means, users can build an appliance from the same appliance description with KIWI Legacy and KIWI NG, if the distribution and all configured features are supported by the used version.

This provides an opportunity for users to test KIWI NG with their appliance descriptions without risk. If it builds and works as expected, I recommend to switch to the KIWI NG. If not, please open an issue on <https://github.com/OSInside/kiwi>.

The KIWI Legacy version will be further developed in maintenance mode. There won't be any new features added in that code base though. Packages will be available at the known place: [KIWI Legacy packages](#)

4.11.1 When Do I need to use KIWI Legacy

- If you are building images using one of the features of the dropped features list below.
- If you are building images for an older distribution compared to the list on the main page, see *Supported Distributions*.

4.11.2 Dropped Features

The following features have been dropped. If you make use of them consider to use the KIWI Legacy version.

Split systems The KIWI Legacy version supports building of split systems which uses a static definition of files and directories marked as read-only or read-write. Evolving technologies like overlayfs makes this feature obsolete.

ZFS filesystem The successor for ZFS is Btrfs in the opensource world. All major distributions put on Btrfs. This and the proprietary attitude of ZFS obsoletes the feature.

Reiserfs filesystem The number of people using this filesystem is decreasing. For image building reiserfs was an interesting filesystem however with Btrfs and XFS there are good non inode based alternatives out there. Therefore we don't continue supporting Reiserfs.

Btrfs seed based live systems A Btrfs seed device is an alternative for other copy on write filesystems like overlayfs. Unfortunately the stability of the seed device when used as cow part in a live system was not as good as we provide with overlayfs and clicfs. Therefore this variant is no longer supported. We might think of adding this feature back if people demand it.

lxc container format lxc has a successor in docker based on the former lxc technology. Many distributions also dropped the lxc tools from the distribution in favour of docker.

OEM Recovery/Restore Recovery/Restore in the world of images has been moved from the operating system layer into higher layers. For example, in private and public Cloud environments disk and image recovery as well as backup strategies are part of Cloud services. Pure operating system recovery and snapshots for consumer machines are provided as features of the distribution. SUSE as an example provides this via Rear (Relax-and-Recover) and snapshot based filesystems (btrfs+snapper). Therefore the recovery feature offered in the KIWI Legacy version will not be continued.

Partition based install method in OEM install image The section *Deployment Methods* describes the supported OEM installation procedures. The KIWI Legacy version also provided a method to install an image based on the partitions of the OEM disk image. Instead of selecting one target disk to dump the entire image file to, the user selects target partitions. Target partitions could be located on several disks. Each partition of the OEM disk image must be mapped on a selectable target partition. It turned out, users needed a lot of experience in a very sensitive area of the operating system. This is contrary to the idea of images to be dumped and be happy. Thus the partition based install method will not be continued.

4.11.3 Compatibility

The KIWI Legacy version can be installed and used together with KIWI NG.

Note: Automatic Link Creation for **kiwi** Command

Note the python3-kiwi package uses the alternatives mechanism to setup a symbolic link named **kiwi** to the real executable named **kiwi-ng**. If the link target `/usr/bin/kiwi` already exists on your system, the alternative setup will skip the creation of the link target because it already exists.

From an appliance description perspective, both versions are fully compatible. Users can build their appliances with both versions and the same appliance description. If the appliance description uses features KIWI NG does not provide, the build will fail with an exception early. If the appliance description uses next generation features like the selection of the initrd system, it's not possible to build that with the KIWI Legacy, unless the appliance description properly encapsulates the differences into a profile.

KIWI NG also provides the `--compat` option and the **kiwicompat** tool to be able to use the same commandline as provided with the KIWI Legacy version.

4.12 Overview

KIWI NG builds so-called *system images* (a fully installed and optionally configured system in a single file) of a Linux distribution in two steps (for further details, see *Image Building Process*):

1. *Prepare operation*: generate an *unpacked image tree* of your image. The unpacked tree is a directory containing the future file system of your image, generated from your image description.
2. *Create operation*: the unpacked tree generated in step 1 is packaged into the format required for the final usage (e.g. a qcow2 disk image to launch the image with QEMU).

KIWI NG executes these steps using the following components, which it expects to find in the *description directory*:

1. `config.xml`: *The Image Description*

This XML file contains the image description, which is a collection of general settings of the final image, like the partition table, installed packages, present users, etc.

The filename `config.xml` is not mandatory, the image description file can also have an arbitrary name plus the `*.kiwi` extension. KIWI NG first looks for a `config.xml` file. If it cannot be found, it picks the first `*.kiwi` file.

2. `config.sh` and `images.sh`: *User Defined Scripts*

If present, these configuration shell scripts run at the end of the *prepare operation* (`config.sh`) or at the beginning of the *create operation* (`images.sh`). They can be used to fine tune the image in ways that are not possible via the settings provided in `config.xml`.

3. Overlay tree directory

The *overlay tree* is a folder (called `root`) or a tarball (called `root.tar.gz`) that contains files and directories that will be copied into the *unpacked image tree* during the *Prepare operation*. The copying is executed after all the packages included in `config.xml` have been installed. Any already present files are overwritten.

4. CD root user data

For live ISO images and install ISO images an optional archive is supported. This is a tar archive matching the name `config-cdroot.tar[.compression_postfix]`.

If present, the archive will be unpacked as user data on the ISO image. For example, this is used to add license files or user documentation. The documentation can then be read directly from the CD/DVD without booting from the media.

5. Archives included in the `config.xml` file.

The archives that are included in `<packages>` using the `<archive>` element (see *The archive element*):


```
<packages type="image">
  <archive name="custom-archive.tgz"/>
</packages>
```

4.13 Image Building Process

KIWI NG creates images in a two step process: The first step, the *prepare* operation, generates a so-called *unpacked image tree* (directory) using the information provided in the `config.xml` configuration file (see [The Image Description](#))

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

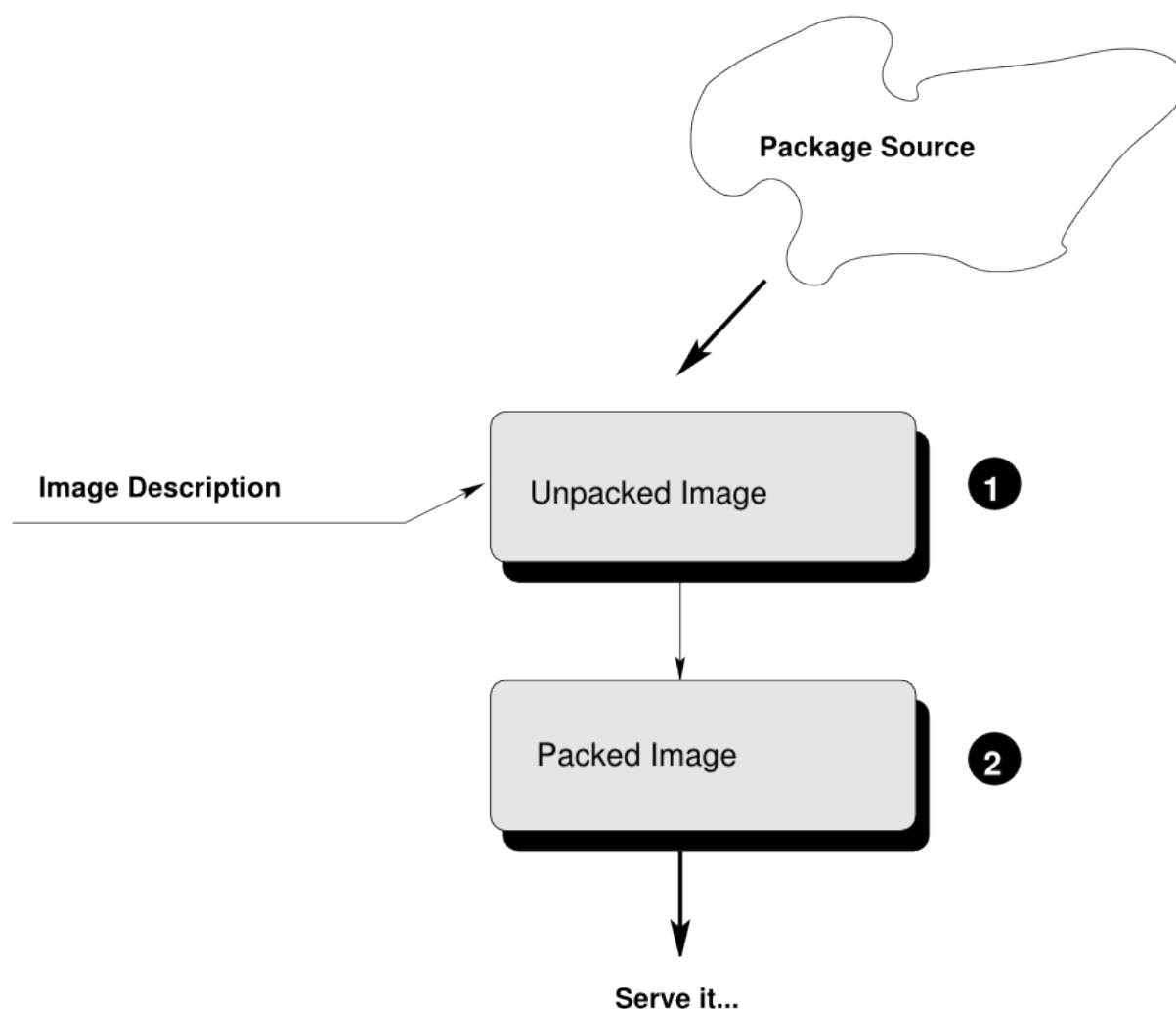


Fig. 1: Image Creation Architecture

4.13.1 The Prepare Step

As the first step, KIWI NG creates an *unpacked image tree*, also called “root tree”. This directory will be the installation target for software packages to be installed during the image creation process.

For the package installation, KIWI NG relies on the package manager specified in the `packagemanager` element in `config.xml`. KIWI NG supports the following package managers: `dnf`, `zypper` (default) and `apt-get`.

The prepare step consists of the following substeps:

1. **Create Target Root Directory**

KIWI NG aborts with an error if the target root tree already exists to avoid accidental deletion of an existing unpacked image.

2. **Install Packages**

First, KIWI NG configures the package manager to use the repositories specified in the configuration file, via the command line, or both. After the repository setup, the packages specified in the `bootstrap` section of the image description are installed in a temporary directory external to the target root tree. This establishes the initial environment to support the completion of the process in a chroot setting. The essential packages are `filesystem` and `glibc-locale` to specify as part of the bootstrap. The dependency chain of these two packages is usually sufficient to populate the bootstrap environment with all required software to support the installation of packages into the new root tree. The aforementioned two packages might not be enough for every distribution. Consult the [kiwi-descriptions repository](#) containing examples for various Linux distributions.

The installation of software packages through the selected package manager may install unwanted packages. Removing these packages can be accomplished by marking them for deletion in the image description, see [Adding and removing packages](#).

3. **Apply the Overlay Tree**

Next, KIWI NG applies all files and directories present in the overlay directory named `root` or in the compressed overlay `root.tar.gz` to the target root tree. Files already present in the target root directory are overwritten. This allows you to overwrite any file that was installed by one of the packages during the installation phase.

4. **Apply Archives**

All archives specified in the `archive` element of the `config.xml` file are applied in the specified order (top to bottom) after the overlay tree copy operation is complete (see [The archive element](#)). Files and directories are extracted relative to the top level of the new root tree. As with the overlay tree, it is possible to overwrite files already existing in the target root tree.

5. **Execute the user-defined script `config.sh`**

At the end of the preparation stage the script `config.sh` is executed (if present). It is run in the top level directory of the target root tree. The script’s primary function

is to complete the system configuration, for example, to activate services. See [Image Customization via the `config.sh` Shell Script](#) section for further details.

6. Modify the Root Tree

The unpacked image tree is now finished to be converted into the final image in the *create step*. It is possible to make manual modifications to the unpacked tree before it is converted into the final image.

Since the unpacked image tree is just a directory, it can be modified using the standard tools. Optionally, it is also possible to “change root (**chroot**)” into it, for instance to invoke the package manager. Beside the standard file system layout, the unpacked image tree contains an additional directory named `/image` that is not present in a regular system. It contains information KIWI NG requires during the create step, including a copy of the `config.xml` file.

By default, KIWI NG will not stop after the *prepare step* and will directly proceed with the *create step*. Therefore to perform manual modifications, proceed as follows:

```
$ kiwi-ng system prepare $ARGS
$ # make your changes
$ kiwi-ng system create $ARGS
```

Warning: Modifications of the unpacked root tree

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

4.13.2 The Create Step

KIWI NG creates the final image during the *create step*: it converts the unpacked root tree into one or multiple output files appropriate for the respective build type.

It is possible to create multiple images from the same unpacked root tree, for example, a self installing OEM image and a virtual machine image from the same image description. The only prerequisite is that both image types are specified in `config.xml`.

During the *create step* the following operations are performed by KIWI NG:

1. Execute the User-defined Script `images.sh`

At the beginning of the image creation process the script named `images.sh` is executed (if present). It is run in the top level directory of the unpacked root tree. The script is usually used to remove files that are no needed in the final image. For example, if an appliance is being built for a specific hardware, unnecessary kernel drivers can be removed using this script.

See [Image Customization via the `images.sh` Shell Script](#) for further details.

2. Create the Requested Image Type

KIWI NG converts the unpacked root into an output format appropriate for the requested build type.

WORKING WITH IMAGES

These sections contains some “low level” topics which are useful for different image types.

5.1 Deploy ISO Image on an USB Stick

Abstract

This page provides further information for handling ISO images built with KIWI NG and references the following articles:

- *[Build an ISO Hybrid Live Image](#)*

In KIWI NG all generated ISO images are created to be hybrid. This means, the image can be used as a CD/DVD or as a disk. This works because the ISO image also has a partition table embedded. With more and more computers delivered without a CD/DVD drive this becomes important.

The very same ISO image can be copied onto a USB stick and used as a bootable disk. The following procedure shows how to do this:

1. Plug in a USB stick

Once plugged in, check which Unix device name the stick was assigned to. The following command provides an overview about all linux storage devices:

```
$ lsblk
```

2. Dump the ISO image on the USB stick:

Warning: Make sure the selected device really points to your stick because the following operation can not be revoked and will destroy all data on the selected device

```
$ dd if=LimeJeOS-Leap-15.1.x86_64-1.15.1.iso of=/dev/  
→<stickdevice>
```


3. Boot from your USB Stick

Activate booting from USB in your BIOS/UEFI. As many firmware has different procedures on how to do it, look into your user manual. Many firmware offers a boot menu which can be activated at boot time.

5.2 Deploy ISO Image as File on a FAT32 Formated USB Stick

Abstract

This page provides further information for handling ISO images built with KIWI NG and references the following articles:

- [*Build an ISO Hybrid Live Image*](#)

In KIWI NG, all generated ISO images are created to be hybrid. This means, the image can be used as a CD/DVD or as a disk. The deployment of such an image onto a disk like an USB stick normally destroys all existing data on this device. Most USB sticks are pre-formatted with a FAT32 Windows File System and to keep the existing data on the stick untouched a different deployment needs to be used.

The following deployment process copies the ISO image as an additional file to the USB stick and makes the USB stick bootable. The ability to boot from the stick is configured through a SYSLINUX feature which allows to loopback mount an ISO file and boot the kernel and initrd directly from the ISO file.

The initrd loaded in this process must also be able to loopback mount the ISO file to access the root filesystem and boot the live system. The dracut initrd system used by KIWI NG provides this feature upstream called as “iso-scan”. Therefore all KIWI NG generated live ISO images supports this deployment mode.

For copying the ISO file on the USB stick and the setup of the SYSLINUX bootloader to make use of the “iso-scan” feature an extra tool named `live-grub-stick` exists. The following procedure shows how to setup the USB stick with `live-grub-stick`:

1. Install the `live-grub-stick` package from software.opensuse.org:
2. Plug in a USB stick

Once plugged in, check which Unix device name the FAT32 partition was assigned to. The following command provides an overview about all storage devices and their filesystems:

```
$ sudo lsblk --fs
```

3. Call the `live-grub-stick` command as follows:

Assuming “/dev/sdz1” was the FAT32 partition selected from the output of the `lsblk` command:


```
$ sudo live-grub-stick LimeJeOS-Leap-15.1.x86_64-1.15.1.iso /  
→dev/sdz1
```

4. Boot from your USB Stick

Activate booting from USB in your BIOS/UEFI. As many firmware has different procedures on how to do it, look into your user manual. EFI booting from iso image is not supported at the moment, for EFI booting use `–isohybrid` option with `live-grub-stick`, however note that all the data on the stick will be lost. Many firmware offers a boot menu which can be activated at boot time. Usually this can be reached by pressing the `Esc` or `F12` keys.

5.3 Image Description for Amazon EC2

Abstract

This page provides further information for handling vmx images built with KIWI NG and references the following articles:

- *[Build a Virtual Disk Image](#)*

A virtual disk image which is able to boot in the Amazon EC2 cloud framework has to comply the following constraints:

- Xen tools and libraries must be installed
- cloud-init package must be installed
- cloud-init configuration for Amazon must be provided
- Grub bootloader modules for Xen must be installed
- AWS tools must be installed
- Disk size must be set to 10G
- Kernel parameters must allow for xen console

To meet this requirements add or update the KIWI NG image description as follows:

1. Software packages

Make sure to add the following packages to the package list

Note: Package names used in the following list matches the package names of the SUSE distribution and might be different on other distributions.

```
<package name="aws-cli"/>  
<package name="grub2-x86_64-xen"/>
```

(continues on next page)

(continued from previous page)

```
<package name="xen-libs"/>
<package name="xen-tools-domU"/>
<package name="cloud-init"/>
```

2. Image Type definition

Update the vmx image type setup as follows

```
<type image="vmx"
  filesystem="ext4"
  bootloader="grub2"
  kernelcmdline="console=xvc0 multipath=off net.ifnames=0"
  boottimeout="1"
  devicepersistency="by-label"
  firmware="ec2">
  <size unit="M">10240</size>
  <machine xen_loader="hvmloader"/>
</type>
```

3. Cloud Init setup

Cloud init is a service which runs at boot time and allows to customize the system by activating one ore more cloud init modules. For Amazon EC2 the following configuration file `/etc/cloud/cloud.cfg` needs to be provided as part of the overlay files in your KIWI NG image description

```
users:
  - default

disable_root: true
preserve_hostname: false
syslog_fix_perms: root:root

datasource_list: [ NoCloud, Ec2, None ]

cloud_init_modules:
  - migrator
  - bootcmd
  - write-files
  - growpart
  - resizefs
  - set_hostname
  - update_hostname
  - update_etc_hosts
  - ca-certs
  - rsyslog
  - users-groups
  - ssh

cloud_config_modules:
```

(continues on next page)

(continued from previous page)

```
- mounts
- ssh-import-id
- locale
- set-passwords
- package-update-upgrade-install
- timezone

cloud_final_modules:
- scripts-per-once
- scripts-per-boot
- scripts-per-instance
- scripts-user
- ssh-authkey-fingerprints
- keys-to-console
- phone-home
- final-message
- power-state-change

system_info:
  default_user:
    name: ec2-user
    gecost: "cloud-init created default user"
    lock_passwd: True
    sudo: ["ALL=(ALL) NOPASSWD:ALL"]
    shell: /bin/bash
  paths:
    cloud_dir: /var/lib/cloud/
    templates_dir: /etc/cloud/templates/
  ssh_svcname: sshd
```

An image built with the above setup can be uploaded into the Amazon EC2 cloud and registered as image. For further information on how to upload to EC2 see: [ec2uploadimg](#)

5.4 Image Description for Microsoft Azure

Abstract

This page provides further information for handling vmx images built with KIWI NG and references the following articles:

- *Build a Virtual Disk Image*

A virtual disk image which is able to boot in the Microsoft Azure cloud framework has to comply the following constraints:

- Hyper-V tools must be installed

- Microsoft Azure Agent must be installed
- Disk size must be set to 30G
- Kernel parameters must allow for serial console

To meet this requirements update the KIWI NG image description as follows:

1. Software packages

Make sure to add the following packages to the package list

Note: Package names used in the following list matches the package names of the SUSE distribution and might be different on other distributions.

```
<package name="hyper-v"/>
<package name="python-azure-agent"/>
```

2. Image Type definition

Update the vmx image type setup as follows

```
<type image="vmx"
  filesystem="ext4"
  boottimeout="1"
  kernelcmdline="console=ttyS0 rootdelay=300 net.ifnames=0"
  devicepersistency="by-uuid"
  format="vhd-fixed"
  formatoptions="force_size"
  bootloader="grub2"
  bootpartition="true"
  bootpartsize="1024">
  <size unit="M">30720</size>
</type>
```

An image built with the above setup can be uploaded into the Microsoft Azure cloud and registered as image. For further information on how to upload to Azure see: [azurectl](#)

5.5 Image Description for Google Compute Engine

Abstract

This page provides further information for handling vmx images built with KIWI NG and references the following articles:

- *Build a Virtual Disk Image*

A virtual disk image which is able to boot in the Google Compute Engine cloud framework has to comply the following constraints:

- KIWI NG type must be an expandable disk
- Google Compute Engine init must be installed
- Disk size must be set to 10G
- Kernel parameters must allow for serial console

To meet this requirements update the KIWI NG image description as follows:

1. Software packages

Make sure to add the following packages to the package list

Note: Package names used in the following list matches the package names of the SUSE distribution and might be different on other distributions.

```
<package name="google-compute-engine-init"/>
```

2. Image Type definition

To allow the image to be expanded to the configured disk geometry of the instance started by Google Compute Engine it is suggested to let KIWI NG's OEM boot code take over that task. It would also be possible to try cloud-init's resize module but we found conflicts when two cloud init systems, google-compute-engine-init and cloud-init were used together. Thus for now we stick with KIWI NG's boot code which can resize the disk from within the initrd before the system gets activated through systemd.

Update the vmx image type setup to be changed into an expandable (oem) type as follows:

```
<type image="oem"
  initrd_system="dracut"
  filesystem="ext4" boottimeout="1"
  kernelcmdline="console=ttyS0,38400n8 net.ifnames=0"
  format="gce"
  bootloader="grub2">
  <size unit="M">10240</size>
  <oemconfig>
    <oem-swap>>false</oem-swap>
  </oemconfig>
</type>
```

An image built with the above setup can be uploaded into the Google Compute Engine cloud and registered as image. For further information on how to upload to Google see: `google-cloud-sdk` on software.opensuse.org

5.6 Setting Up a Network Boot Server

Abstract

This page provides further information for handling PXE images built with KIWI NG and references the following articles:

- *Build a PXE Root File System Image*

To be able to deploy PXE boot images created with KIWI NG, you need to set up a network boot server providing the services DHCP and tftp.

5.6.1 Installing and Configuring tftp

1. Install the packages tftp and kiwi-pxeboot.
2. Start the tftpd service by calling:

```
$ systemctl start tftpd.socket  
$ systemctl start tftpd
```

5.6.2 Installing and Configuring DHCP

Contrary to the tftp server setup the following instructions can only serve as an example. Depending on your network structure, the IP addresses, ranges and domain settings need to be adapted to allow the DHCP server to work within your network. If you already have a DHCP server running in your network, make sure that the `filename` and `next-server` directives are correctly set on this server.

The following steps describe how to set up a new DHCP server instance using dnsmasq:

1. Install the dnsmasq package.
2. Create the file `/etc/dnsmasq.conf` and insert the following content:

Note: Placeholders

Replace all placeholders (written in uppercase) with data fitting your network setup.

```
# Don't function as a DNS server:  
port=0  
  
# Log lots of extra information about DHCP transactions.  
log-dhcp  
  
# Set the root directory for files available via FTP,
```

(continues on next page)

(continued from previous page)

```
# usually "/srv/tftpboot":
tftp-root=TFTP_ROOT_DIR

# The boot filename, Server name, Server Ip Address
dhcp-boot=pxelinux.0,,BOOT_SERVER_IP

# Disable re-use of the DHCP servername and filename fields as
# extra
# option space. That's to avoid confusing some old or broken
# DHCP clients.
dhcp-no-override

# PXE menu. The first part is the text displayed to the user.
# The second is the timeout, in seconds.
pxe-prompt="Booting FOG Client", 1

# The known types are x86PC, PC98, IA64_EFI, Alpha, Arc_x86,
# Intel_Lean_Client, IA32_EFI, BC_EFI, Xscale_EFI and X86-64_EFI
# This option is first and will be the default if there is no
# input
# from the user.
pxe-service=X86PC, "Boot to FOG", pxelinux.0
pxe-service=X86-64_EFI, "Boot to FOG UEFI", ipxe
pxe-service=BC_EFI, "Boot to FOG UEFI PXE-BC", ipxe

dhcp-range=BOOT_SERVER_IP,proxy
```

3. Run the dnsmasq server by calling:

```
systemctl start dnsmasq
```

5.7 Setting Up YaST at First Boot

Abstract

This page provides information how to setup the KIWI NG XML description to start the SUSE YaST system setup utility at first boot of the image

To be able to use YaST in a non interactive way, create a YaST profile which tells the autoyast module what to do. To create the profile, run:

```
yast autoyast
```

Once the YaST profile exists, update the KIWI NG XML description as follows:

1. Edit the KIWI NG XML file and add the following package to the <packages


```
type="image"> section:
```

```
<package name="yast2-firstboot"/>
```

2. Copy the YaST profile file as overlay file to your KIWI NG image description overlay directory:

```
cd IMAGE_DESCRIPTION_DIRECTORY
mkdir -p root/etc/YaST2
cp PROFILE_FILE root/etc/YaST2/firstboot.xml
```

3. Copy and activate the YaST firstboot template. This is done by the following instructions which needs to be written into the KIWI NG `config.sh` which is stored in the image description directory:

```
sysconfig_firsboot=/etc/sysconfig/firstboot
sysconfig_template=/var/adm/fillup-templates/sysconfig.firstboot
if [ ! -e "${sysconfig_firsboot}" ]; then
    cp "${sysconfig_template}" "${sysconfig_firsboot}"
fi

touch /var/lib/YaST2/reconfig_system
```

5.8 PXE Client Setup Configuration

Abstract

This page provides further information for handling PXE images built with KIWI NG and references the following articles:

- *Build a PXE Root File System Image*

All PXE boot based deployment methods are controlled by configuration files located in `/srv/tftpboot/KIWI` on the PXE server. Such a configuration file can either be client-specific (`config.MAC_ADDRESS`, for example `config.00.AB.F3.11.73.C8`), or generic (`config.default`).

In an environment with heterogeneous clients, this allows to have a default configuration suitable for the majority of clients, to have configurations suitable for a group of clients (for example machines with similar or identical hardware), and individual configurations for selected machines.

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

```
IMAGE="device;name;version;srvip;bsize;compressed, ..., "
DISK="device"
```

(continues on next page)

(continued from previous page)

```
PART="size;id;Mount,...,size;id;Mount"
RAID="raid-level;device1;device2;..."

AOEROOT=ro-device[,rw-device]
NBDROOT="ip-address;export-name;device;swap-export-name;swap-device;
↪write-export-name;write-device"
NFSROOT="ip-address;path"

UNIONFS_CONFIGURATION="rw-partition,compressed-partition,overlayfs"

CONF="src;dest;srvip;bsize;[hash],...,src;dest;srvip;bsize;[hash]"

KIWI_BOOT_TIMEOUT="seconds"
KIWI_KERNEL_OPTIONS="opt1 opt2 ..."

REBOOT_IMAGE=1
RELOAD_CONFIG=1
RELOAD_IMAGE=1
```

Note: Quoting the Values

The configuration file is sourced by Bash, so the same quoting rules as for Bash apply.

Not all configuration options needs to be specified. It depends on the setup of the client which configuration values are required. The following is a collection of client setup examples which covers all supported PXE client configurations.

5.8.1 Setup Client with Remote Root

To serve the image from a remote location and redirect all write operations on a tmpfs, the following setup is required:

```
# When using AoE, see vblade toolchain for image export

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

# When using NFS, see exports manual page for image export

NFSROOT="192.168.100.2;/srv/tftpboot/image/root"
UNIONFS_CONFIG=tmpfs,nfs,overlay

# When using NBD, see nbd-server manual page for image export

NBDROOT=192.168.100.2;root_export;/dev/nbd0
UNIONFS_CONFIG=tmpfs,nbd,overlay
```


The above setup shows the most common use case where the image built with KIWI NG is populated over the network using either AoE, NBD or NFS in combination with overlayfs which redirects all write operations to be local to the client. In any case a setup of either AoE, NBD or NFS on the image server is required beforehand.

5.8.2 Setup Client with System on Local Disk

To deploy the image on a local disk the following setup is required:

Note: In the referenced suse-leap-15.1-JeOS XML description the `pxe` type must be changed as follows and the image needs to be rebuild:

```
<type image="pxe" filesystem="ext3" boot="netboot/suse-leap15.1"/>
```

```
IMAGE="/dev/sda2;LimeJeOS-Leap-15.1.x86_64;1.15.1;192.168.100.2;4096  
↪"  
DISK="/dev/sda"  
PART="5;S;X,X;L;/"
```

The setup above will create a partition table on sda with a 5MB swap partition (no mountpoint) and the rest of the disk will be a Linux(L) partition with / as mountpoint. The (X) in the PART setup specifies a place holder to indicate the default behaviour.

5.8.3 Setup Client with System on Local MD RAID Disk

To deploy the image on a local disk with prior software RAID configuration, the following setup is required:

Note: In the referenced suse-leap-15.1-JeOS XML description the `pxe` type must be changed as follows and the image needs to be rebuild:

```
<type image="pxe" filesystem="ext3" boot="netboot/suse-leap15.1"/>
```

```
RAID="1;/dev/sda;/dev/sdb"  
IMAGE="/dev/md1;LimeJeOS-Leap-15.1.x86_64;1.15.1;192.168.100.2;4096"  
PART="5;S;x,x;L;/"
```

The first parameter of the RAID line is the RAID level. So far only raid1 (mirroring) is supported. The second and third parameter specifies the raid disk devices which make up the array. If a RAID line is present all partitions in PART will be created as RAID partitions. The first RAID is named md0, the second one md1 and so on. It is required to specify the correct RAID partition in the IMAGE line according to the PART setup. In this case md0 is reserved for the SWAP space and md1 is reserved for the system.

5.8.4 Setup Loading of Custom Configuration File(s)

In order to load for example a custom `/etc/hosts` file on the client, the following setup is required:

```
CONF="hosts;/etc/hosts;192.168.1.2;4096;ffffffff"
```

On boot of the client KIWI NG's boot code will fetch the `hosts` file from the root of the server (192.168.1.2) with 4k blocksize and deploy it as `/etc/hosts` on the client. The protocol is by default `tftp` but can be changed via the `kiwiservertype` kernel commandline option. For details, see *Setup a Different Download Protocol and Server*

5.8.5 Setup Client to Force Reload Image

To force the reload of the system image even if the image on the disk is up-to-date, the following setup is required:

```
RELOAD_IMAGE=1
```

The option only applies to configurations with a `DISK/PART` setup

5.8.6 Setup Client to Force Reload Configuration Files

To force the reload of all configuration files specified in `CONF`, the following setup is required:

```
RELOAD_CONFIG=1
```

By default only configuration files which has changed according to their `md5sum` value will be reloaded. With the above setup all files will be reloaded from the PXE server. The option only applies to configurations with a `DISK/PART` setup

5.8.7 Setup Client for Reboot After Deployment

To reboot the system after the initial deployment process is done the following setup is required:

```
REBOOT_IMAGE=1
```

5.8.8 Setup custom kernel boot options

To deactivate the kernel mode setting on local boot of the client the following setup is required:

```
KIWI_KERNEL_OPTIONS="nomodeset"
```

Note: This does not influence the kernel options passed to the client if it boots from the network. In order to setup those the PXE configuration on the PXE server needs to be changed

5.8.9 Setup a Custom Boot Timeout

To setup a 10sec custom timeout for the local boot of the client the following setup is required.

```
KIWI_BOOT_TIMEOUT="10"
```

Note: This does not influence the boot timeout if the client boots off from the network.

5.8.10 Setup a Different Download Protocol and Server

By default all downloads controlled by the KIWI NG linuxrc code are performed by an atftp call using the TFTP protocol. With PXE the download protocol is fixed and thus you cannot change the way how the kernel and the boot image (`initrd`) is downloaded. As soon as Linux takes over, the download protocols http, https and ftp are supported too. KIWI NG uses the curl program to support the additional protocols.

To select one of the additional download protocols the following kernel parameters need to be specified

```
kiwiserver=192.168.1.1 kiwiservertype=ftp
```

To set up this parameters edit the file `/srv/tftpboot/pxelinux.cfg/default` on your PXE boot server and change the append line accordingly.

Note: Once configured 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

5.9 Image Description for Vagrant

Abstract

This page provides further information for handling VMX images built with KIWI NG and references the following article:

- *Build a Virtual Disk Image*

Vagrant is a framework to implement consistent processing/testing work environments based on Virtualization technologies. To run a system, Vagrant needs so-called **boxes**. A box is a TAR archive containing a virtual disk image and some metadata.

To build Vagrant boxes, you can use **Packer** which is provided by Hashicorp itself. Packer is based on the official installation media (DVDs) as shipped by the distribution vendor.

The KIWI NG way of building images might be helpful, if such a media does not exist or does not suit your needs. For example, if the distribution is still under development or you want to use a collection of your own repositories. Note, that in contrast to Packer KIWI NG only supports the libvirt and VirtualBox providers. Other providers require a different box layout that is currently not supported by KIWI NG.

In addition, you can use the KIWI NG image description as source for the **Open Build Service** which allows building and maintaining boxes.

Vagrant expects boxes to be setup in a specific way (for details refer to the [Vagrant box documentation](#).), applied to the referenced KIWI NG image description from *Build a Virtual Disk Image*, the following steps are required:

1. Update the image type setup

```
<type image="vmx" filesystem="ext4" format="vagrant" ↵
↪boottimeout="0">
  <vagrantconfig provider="libvirt" virtualsize="42"/>
  <size unit="G">42</size>
</type>
```

This modifies the type to build a Vagrant box for the libvirt provider including a pre-defined disk size. The disk size is optional, but recommended to provide some free space on disk.

For the VirtualBox provider, the additional attribute `virtualbox_guest_additions_present` can be set to `true` when the VirtualBox guest additions are installed in the KIWI NG image:

```
<type image="vmx" filesystem="ext4" format="vagrant" ↵
↪boottimeout="0">
  <vagrantconfig
    provider="virtualbox"
    virtualbox_guest_additions_present="true"
    virtualsize="42"
  />
  <size unit="G">42</size>
</type>
```

The resulting Vagrant box then uses the `vboxfs` module for the synchronized folder instead of `rsync`, that is used by default.

2. Add mandatory packages

```
<package name="sudo" />
<package name="openssh" />
```


3. Add additional packages

If you have set the attribute `virtualbox_guest_additions_present` to `true`, add the VirtualBox guest additions. For openSUSE the following packages are required:

```
<package name="virtualbox-guest-tools"/>
<package name="virtualbox-guest-x11"/>
<package name="virtualbox-guest-kmp-default"/>
```

Otherwise, you must add `rsync`:

```
<package name="rsync"/>
```

Note that KIWI NG cannot verify whether these packages are installed. If they are missing, the resulting Vagrant box will be broken.

4. Add Vagrant user

```
<users group='vagrant'>
  <user name='vagrant' password='vh4vw1N4alxKQ' home='/home/
  ↪vagrant' />
</users>
```

This adds the **vagrant** user to the system and applies the name of the user as the password for login.

5. Integrate public SSH key

Vagrant requires an insecure public key pair¹ to be added to the authorized keys for the user `vagrant` so that Vagrant itself can connect to the box via `ssh`. The key can be obtained from [GitHub](#) and should be inserted into the file `home/vagrant/.ssh/authorized_keys`, which can be added as an overlay file into the image description.

Keep in mind to set the file system permissions of `home/vagrant/.ssh/` and `home/vagrant/.ssh/authorized_keys` correctly, otherwise Vagrant will not be able to connect to your box. The following snippet can be added to `config.sh`:

```
chmod 0600 /home/vagrant/.ssh/authorized_keys
chown -R vagrant:vagrant /home/vagrant/
```

6. Create the default shared folder

Vagrant boxes usually provide a default shared folder under `/vagrant`. Consider adding this empty folder to your overlay files and ensure that the user `vagrant` has write permissions to it.

Note, that the boxes that KIWI NG produces **require** this folder to exist, otherwise Vagrant will not be able to start them properly.

7. Setup and start SSH daemon

In `config.sh` add the start of `sshd`:

¹ The insecure key is removed from the box when the it is first booted via Vagrant.


```
#=====
# Activate services
#-----
baseInsertService sshd
```

Also make sure to add the line **UseDNS=no** into `/etc/ssh/sshd_config`. This can be done by an overlay file or by patching the file in the above mentioned `config.sh` file.

8. Configure sudo for the Vagrant user

Vagrant expects to have passwordless root permissions via `sudo` to be able to setup your box. Add the following line to `/etc/sudoers` or add it into a new file `/etc/sudoers.d/vagrant`:

```
vagrant ALL=(ALL) NOPASSWD: ALL
```

You can also use **visudo** to verify that the resulting `/etc/sudoers` or `/etc/sudoers.d/vagrant` are valid:

```
visudo -cf /etc/sudoers
if [ $? -ne 0 ]; then
    exit 1
fi
```

An image built with the above setup creates a Vagrant box file with the extension `.vagrant.libvirt.box` or `.vagrant.virtualbox.box`. Add the box file to Vagrant with the command:

```
vagrant box add my-box image-file.vagrant.libvirt.box
```

Note: Using the box with the libvirt provider requires alongside a correct Vagrant installation:

- the plugin `vagrant-libvirt` to be installed
 - a running `libvirtd` daemon
-

Once added to Vagrant, boot the box and log in with the following sequence of **vagrant** commands:

```
vagrant init my-box
vagrant up --provider libvirt
vagrant ssh
```


5.9.1 Customizing the embedded Vagrantfile

Warning: This is an advanced topic and not required for most users

Vagrant ship with an embedded `Vagrantfile` that carries settings specific to this box, for instance the synchronization mechanism for the shared folder. KIWI NG generates such a file automatically for you and it should be sufficient for most use cases.

If a box requires different settings in the embedded `Vagrantfile`, then the user can provide KIWI NG with a path to an alternative via the attribute `embedded_vagrantfile` of the `vagrantconfig` element: it specifies a relative path to the `Vagrantfile` that will be included in the finished box.

In the following example snippet from `config.xml` we add a custom `MyVagrantfile` into the box (the file should be in the image description directory next to `config.sh`):

```
<type image="vmx" filesystem="ext4" format="vagrant" boottimeout="0
  <vagrantconfig
    provider="libvirt"
    virtualsize="42"
    embedded_vagrantfile="MyVagrantfile"
  />
  <size unit="G">42</size>
</type>
```

The option to provide a custom `Vagrantfile` can be combined with the usage of *profiles* (see *Image Profiles*), so that certain builds can use the automatically generated `Vagrantfile` (in the following example that is the Virtualbox build) and others get a customized one (the libvirt profile in the following example):

```
<?xml version="1.0" encoding="utf-8"?>

<image schemaversion="7.1" name="LimeJeOS-Leap-15.1">
  <!-- description goes here -->
  <profiles>
    <profile name="libvirt" description="Vagrant Box for Libvirt"/>
    <profile name="virtualbox" description="Vagrant Box for
    VirtualBox"/>
  </profiles>

  <!-- general preferences go here -->

  <preferences profiles="libvirt">
    <type
      image="vmx"
      filesystem="ext4"
      format="vagrant"
```

(continues on next page)

(continued from previous page)

```
boottimeout="0"
bootloader="grub2">
  <vagrantconfig
    provider="libvirt"
    virtualsize="42"
    embedded_vagrantfile="LibvirtVagrantfile"
  />
  <size unit="G">42</size>
</type>
</preferences>
<preferences profiles="virtualbox">
  <type
    image="vmx"
    filesystem="ext4"
    format="vagrant"
    boottimeout="0"
    bootloader="grub2">
      <vagrantconfig
        provider="virtualbox"
        virtualbox_guest_additions_present="true"
        virtualsize="42"
      />
      <size unit="G">42</size>
    </type>
  </preferences>

  <!-- remaining box description -->
</image>
```

5.10 Booting a Live ISO Image from Network

Abstract

This page provides further information for handling ISO images built with KIWI NG and references the following articles:

- *Build an ISO Hybrid Live Image*

In KIWI NG, live ISO images can be configured to boot via PXE. This functionality requires a network boot server setup on the system. Details how to setup such a server can be found in *Setting Up a Network Boot Server*.

After the live ISO was built as shown in *Build an ISO Hybrid Live Image*, the following configuration steps are required to boot from the network:

1. Extract initrd/kernel From Live ISO

The PXE boot process loads the configured kernel and initrd from the PXE server. For this reason, those two files must be extracted from the live ISO image and copied to the PXE server as follows:

```
$ mount LimeJeOS-Leap-15.1.x86_64-1.15.1.iso /mnt
$ cp /mnt/boot/x86_64/loader/initrd /srv/tftpboot/boot/initrd
$ cp /mnt/boot/x86_64/loader/linux /srv/tftpboot/boot/linux
$ umount /mnt
```

Note: This step must be repeated with any new build of the live ISO image

2. Export Live ISO To The Network

Access to the live ISO file is implemented using the AoE protocol in KIWI NG. This requires the export of the live ISO file as remote block device which is typically done with the **vblade** toolkit. Install the following package on the system which is expected to export the live ISO image:

```
$ zypper in vblade
```

Note: Not all versions of AoE are compatible with any kernel. This means the kernel on the system exporting the live ISO image must provide a compatible aoe kernel module compared to the kernel used in the live ISO image.

Once done, export the live ISO image as follows:

```
$ vbladed 0 1 INTERFACE LimeJeOS-Leap-15.1.x86_64-1.15.1.iso
```

The above command exports the given ISO file as a block storage device to the network of the given INTERFACE. On any machine except the one exporting the file, it will appear as `/dev/etherd/e0.1` once the **aoe** kernel module was loaded. The two numbers, 0 and 1 in the above example, classifies a major and minor number which is used in the device node name on the reading side, in this case `e0.1`. The numbers given at export time must match the AOEINTERFACE name as described in the next step.

Note: Only machines in the same network of the given INTERFACE can see the exported live ISO image. If virtual machines are the target to boot the live ISO image they could all be connected through a bridge. In this case INTERFACE is the bridge device. The availability scope of the live ISO image on the network is in general not influenced by KIWI NG and is a task for the network administrators.

3. Setup live ISO boot entry in PXE configuration

Edit the file `/srv/tftpboot/pxelinux.cfg/default` and create a boot entry of the form:


```

LABEL Live-Boot
    kernel boot/linux
    append initrd=boot/initrd rd.kiwi.live.pxe_
    ↪root=live:AOEINTERFACE=e0.1

```

- The boot parameter `rd.kiwi.live.pxe` tells the KIWI NG boot process to setup the network and to load the required `aoe` kernel module.
- The boot parameter `root=live:AOEINTERFACE=e0.1` specifies the interface name as it was exported by the **vb1aded** command from the last step. Currently only AoE (Ata Over Ethernet) is supported.

4. Boot from the Network

Within the network which has access to the PXE server and the exported live ISO image, any network client can now boot the live system. A test based on QEMU is done as follows:

```
$ qemu -boot n
```

5.11 Deploy and Run System in a RamDisk

Abstract

This page provides further information for handling oem images built with KIWI NG and references the following articles:

- *[Build an OEM Expandable Disk Image](#)*

If a machine should run the OS completely in memory without the need for any persistent storage, the approach to deploy the image into a ramdisk serves this purpose. KIWI NG allows to create a bootable ISO image which deploys the image into a ramdisk and activates that image with the following oem type definition:

```

<type image="oem" filesystem="ext4" installiso="true" bootloader=
    ↪"grub2" initrd_system="dracut" installboot="install" boottimeout=
    ↪"1" kernelcmdline="rd.kiwi.ramdisk ramdisk_size=2048000">
    <oemconfig>
        <oem-skip-verify>true</oem-skip-verify>
        <oem-unattended>true</oem-unattended>
        <oem-unattended-id>/dev/ram1</oem-unattended-id>
        <oem-swap>>false</oem-swap>
        <oem-multipath-scan>>false</oem-multipath-scan>
    </oemconfig>
</type>

```

The type specification above builds an installation ISO image which deploys the System Image into the specified ramdisk device (`/dev/ram1`). The setup of the ISO image boots with a short

boot timeout of 1sec and just runs through the process without asking any questions. In a ramdisk deployment the optional target verification, swap space and multipath targets are out of scope and therefore disabled.

The configured size of the ramdisk specifies the size of the OS disk and must be at least of the size of the System Image. The disk size can be configured with the following value in the kernelcmdline attribute:

- ramdisk_size=kbyte-value”

An image built with the above setup can be tested in QEMU as follows:

```
$ qemu -cdrom LimeJeOS-Leap-15.1.x86_64-1.15.1.install.iso
```

Note: Enough Main Memory

The machine, no matter if it's a virtual machine like QEMU or a real machine, must provide enough RAM to hold the image in the ramdisk as well as have enough RAM available to operate the OS and its applications. The KIWI NG build image with the extension .raw provides the System Image which gets deployed into the RAM space. Subtract the size of the System Image from the RAM space the machine offers and make sure the result is still big enough for the use case of the appliance. In case of a virtual machine, attach enough main memory to fit this calculation. In case of QEMU this can be done with the `-m` option

Like all other oem KIWI NG images, also the ramdisk setup supports all the deployments methods as explained in [Deployment Methods](#). This means it's also possible to dump the ISO image on a USB stick let the system boot from it and unplug the stick from the machine because the system was deployed into RAM

Note: Limitations Of RamDisk Deployments

Only standard images which can be booted by a simple root mount and root switch can be used. Usually KIWI NG calls kexec after deployment such that the correct, for the image created dracut initrd, will boot the image. In case of a RAM only system kexec does not work because it would loose the ramdisk contents. Thus the dracut initrd driving the deployment is also the environment to boot the system. There are cases where this environment is not suitable to boot the system.

5.12 Custom Disk Partitions

Abstract

This page provides some details about what KIWI NG supports and does not support regarding customization over the partition scheme. It also provides some guidance in case the user requires some custom layout beyond KIWI NG supported features.

By design, KIWI NG does not support a customized partition table. Alternatively, KIWI NG supports the definition of user-defined volumes which covers most of common use cases. See [Custom Disk Volumes](#) for further details about that.

KIWI NG has its own partitioning schema which is defined according to several different user configurations: boot firmware, boot partition, expandable layouts, etc. Those supported features have an impact on the partitioning schema. MBR or GUID partition tables are not flexible, carry limitations and are tied to some specific disk geometry. Because of that the preferred alternative to disk layouts based on traditional partition tables is using flexible approaches like logic volumes.

As an example, expandable OEM images is a relevant KIWI NG feature that is incompatible with the idea of adding user defined partitions on the system area.

Despite no full customization is supported, some aspects of the partition schema can be customized. KIWI NG supports:

1. Adding a spare partition *before* the root (/) partition.

It can be achieved by using the `spare_part` type attribute, see [Schema Documentation](#).

2. Leaving some unpartitioned area at the *end* of the disk.

Setting some unpartitioned free space on the disk can be done using the `unpartitioned` attribute of `size` element in type's section. [\[LINK\]](#)

3. Expand built disks to a new size adding unpartitioned free space at the *end* of the disk.

A built image can be resized by using the `kiwi-ng image resize` command and set a new extended size for the disk. See [KIWI NG commands docs here](#).

5.12.1 Custom Partitioning at Boot Time

Adding additional partitions at boot time of KIWI NG images is also possible, however, setting the tools and scripts for doing so needs to be handled by the user. A possible strategy to add partitions on system area are described below.

The main idea consists on running a first boot service that creates the partitions that are needed. Adding custom services is simple, use the following steps:

1. Create a unit file for a systemd service:

```
[Unit]
Description=Add a data partition
After=basic.target
Wants=basic.target

[Service]
Type=oneshot
ExecStart=/bin/bash /usr/local/bin/create_part
```


This systemd unit file will run at boot time once systemd reaches the basic target. At this stage all basic services are up and running (devices mounted, network interfaces up, etc.). In case the service is required to run on earlier stages for some reason, default dependencies need to be disabled, see [systemd man pages](#).

2. Create partitioner shell script matching your specific needs

Consider the following steps for a partitioner shell script that creates a new partition. Following the above unit file example the `/usr/local/bin/create_part` script should cover the following steps:

a. Verify partition exists

Verify the required partition is not mounted neither exists. Exit zero (0) if it is already there.

Use tools such `findmnt` to find the root device and `blkid` or `lsblk` to find a partition with certain label or similar criteria.

b. Create a new partition

Create a new partition. On error, exit with non zero.

Use partitioner tools such as `sgdisk` that can be easily used in non interactive scripts. Using `partprobe` to reload partition table to make OS aware of the changes is handy.

c. Make filesystem

Add the desired filesystem to the new partitions. On error, exit with non zero.

Regular filesystem formatting tools (`mkfs.ext4` just to mention one) can be used to apply the desired filesystem to the just created new partition. At this stage it is handy to add a label to the filesystem for easy recognition on later stages or script reruns.

d. Update fstab file

Just echo and append the desired entry in `/etc/fstab`.

e. Mount partition

`mount --all` will try to mount all fstab volumes, it just omits any already mounted device.

3. Add additional files into the root overlay tree.

The above described unit files and partition creation shell script need to be included into the overlay tree of the image, thus they should be placed into the expected paths in root folder (or in `root.tar.gz` tarball).

4. Activate the service in `config.sh`

The service needs to be enabled during image built time to be run during the very first boot. It can be done by adding the following snippet inside the

config.sh.

5.13 Custom Disk Volumes

Abstract

This chapter provides high level explanations on how to handle volumes or subvolumes definitions for disk images using KIWI NG.

KIWI NG supports defining custom volumes by using the logical volume manager (LVM) for the Linux kernel or by setting volumes at filesystem level when filesystem supports it (e.g. btrfs).

Volumes are defined in the KIWI NG description file `config.xml`, using `systemdisk`. This element is a child of the `type`. Volumes themselves are added via (multiple) volume child elements of the `systemdisk` element:

```
<image schemaversion="7.1" name="openSUSE-Leap-15.1">
  <type image="oem" filesystem="btrfs" preferlvm="true">
    <systemdisk name="vgroup">
      <volume name="usr/lib" size="1G" label="library"/>
      <volume name="@root" freespace="500M"/>
      <volume name="etc_volume" mountpoint="etc" copy_on_write=
→ "false"/>
      <volume name="bin_volume" size="all" mountpoint="/usr/bin"/>
    </systemdisk>
  </type>
</image>
```

Additional non-root volumes are created for each volume element. Volume details can be defined by setting the following volume attributes:

- **name:** Required attribute representing the volume's name. Additionally, this attribute is interpreted as the mountpoint if the `mountpoint` attribute is not used.
- **mountpoint:** Optional attribute that specifies the mountpoint of this volume.
- **size:** Optional attribute to set the size of the volume. If no suffix (M or G) is used, then the value is considered to be in megabytes.

Note: Special name for the root volume

One can use the `@root` name to refer to the volume mounted at `/`, in case some specific size attributes for the root volume have to be defined. For instance:

```
<volume name="@root" size="4G"/>
```

- `freespace`: Optional attribute defining the additional free space added to the volume. If no suffix (M or G) is used, the value is considered to be in megabytes.
- `label`: Optional attribute to set filesystem label of the volume.
- `copy_on_write`: Optional attribute to set the filesystem copy-on-write attribute for this volume.

Warning: The size attributes for filesystem volumes, as for btrfs, are ignored and have no effect.

The `systemdisk` element additionally supports the following optional attributes:

- `name`: The volume group name, by default `kiwiVG` is used. This setting is only relevant for LVM volumes.
- `preferlvm`: Boolean value instructing KIWI NG to prefer LVM even if the used filesystem has its own volume management system.

5.14 Add or Update the Fstab File

Abstract

This page provides further information for customizing the `/etc/fstab` file which controls the mounting of filesystems at boot time.

In KIWI NG, all major filesystems that were created at image build time are handled by KIWI NG itself and setup in `/etc/fstab`. Thus there is usually no need to add entries or change the ones added by KIWI NG. However depending on where the image is deployed later it might be required to pre-populate fstab entries that are unknown at the time the image is build.

Possible use cases are for example:

- Adding NFS locations that should be mounted at boot time. Using `autofs` would be an alternative to avoid additional entries to `fstab`. The information about the NFS location will make this image specific to the target network. This will be independent of the mount method, either `fstab` or the automount map has to provide it.
- Adding or changing entries in a read-only root system which becomes effective on first boot but can't be added at that time because of the read-only characteristics.

Note: Modifications to the `fstab` file are a critical change. If done wrongly the risk exists that the system will not boot. In addition this type of modification makes the image specific to its target and creates a dependency to the target hardware, network, etc. . . . Thus this feature should be used with care.

The optimal way to provide custom fstab information is through a package. If this can't be done the files can also be provided via the overlay file tree of the image description.

KIWI NG supports three ways to modify the contents of the `/etc/fstab` file:

Providing an `/etc/fstab.append` file If that file exists in the image root tree, KIWI NG will take its contents and append it to the existing `/etc/fstab` file. The provided `/etc/fstab.append` file will be deleted after successful modification.

Providing an `/etc/fstab.patch` file The `/etc/fstab.patch` represents a patch file that will be applied to `/etc/fstab` using the `patch` program. This method also allows to change the existing contents of `/etc/fstab`. On success `/etc/fstab.patch` will be deleted.

Providing an `/etc/fstab.script` file The `/etc/fstab.script` represents an executable which is called as chrooted process. This method is the most flexible one and allows to apply any change. On success `/etc/fstab.script` will be deleted.

Note: All three variants to handle the fstab file can be used together. Appending happens first, patching afterwards and the script call is last. When using the script call, there is no validation that checks if the script actually handles fstab or any other file in the image rootfs.

5.15 Image Description Encrypted Disk

Abstract

This page provides further information for handling vmx images with an encrypted root filesystem setup. The information here is based on top of the following article:

- [*Build a Virtual Disk Image*](#)

A virtual disk image can be partially or fully encrypted using the LUKS extension supported by KIWI NG. A fully encrypted image also includes the data in `/boot` to be encrypted. Such an image requests the passphrase for the master key to be entered at the bootloader stage. A partially encrypted image keeps `/boot` unencrypted and on an extra boot partition. Such an image requests the passphrase for the master key later in the boot process when the root partition gets accessed by the `systemd` mount service. In any case the master passphrase is requested only once.

Update the KIWI NG image description as follows:

1. Software packages

Make sure to add the following package to the package list

Note: Package names used in the following list match the package names of the SUSE

distribution and might be different on other distributions.

```
<package name="cryptsetup"/>
```

2. Image Type definition

Update the vmx image type setup as follows

Full disk encryption including /boot:

```
<type image="vmx"
  image="vmx"
  filesystem="ext4"
  bootloader="grub2"
  luks="linux"
  bootpartition="false">
</type>
```

Encrypted root partition with an unencrypted extra /boot partition:

```
<type image="vmx"
  image="vmx"
  filesystem="ext4"
  bootloader="grub2"
  luks="linux"
  bootpartition="true">
</type>
```

Note: The value for the `luks` attribute sets the master passphrase for the LUKS keyring. Therefore the XML description becomes security critical and should only be readable by trustworthy people

BUILDING IMAGES

Note: This document provides an overview about the supported KIWI NG image types. Before building an image with KIWI NG it's important to understand the different image types and their meaning.

6.1 Build an ISO Hybrid Live Image

Abstract

This page explains how to build a live image. It contains:

- how to build an ISO image
- how to run it with QEMU

A Live ISO image is a system on a removable media, e.g CD/DVD or USB stick. Once built and deployed it boots off from this media without interfering with other system storage components making it a useful pocket system for testing and demo- and debugging-purposes.

To add a live ISO build to your appliance, create a `type` element with `image` set to `iso` in your `config.xml` (see *Build Types*) as shown below:

```
<image schemaversion="6.9" name="JeOS-Tumbleweed">
  <!-- snip -->
  <preferences>
    <type image="iso" primary="true" flags="overlay"
    ↪hybridpersistent_filesystem="ext4" hybridpersistent="true"/>
    <!-- additional preferences -->
  </preferences>
  <!-- snip -->
</image>
```

The following attributes of the `type` element are relevant when building live ISO images:

- `flags`: Specifies the live ISO technology and dracut module to use, can be set to `overlay` or to `dmsquash`.

If set to `overlay`, the kiwi-live dracut module will be used to support a live ISO system based on squashfs and overlayfs. If set to `dmsquash`, the dracut standard `dmsquash-live` module will be used to support a live ISO system based on squashfs and the device mapper. Note, both modules support a different set of live features. For details see [Decision for a live ISO technology](#)

- `hybridpersistent`: Accepts `true` or `false`, if set to `true` then the resulting image will be created with a COW file to keep data persistent over a reboot
- `hybridpersistent_filesystem`: The filesystem used for the COW file. Possible values are `ext4` or `xfs`, with `ext4` being the default.

With the appropriate settings present in `config.xml` KIWI NG can now build the image:

```
$ sudo kiwi-ng --type iso system build \
    --description kiwi-descriptions/suse/x86_64/suse-leap-15.1-
↪JeOS \
    --target-dir /tmp/myimage
```

The resulting image is saved in the folder `/tmp/myimage` and can be tested with QEMU:

```
$ qemu -cdrom LimeJeOS-Leap-15.1.x86_64-1.15.1.iso -m 4096
```

The image is now complete and ready to use. See [Deploy ISO Image on an USB Stick](#) and [Deploy ISO Image as File on a FAT32 Formated USB Stick](#) for further information concerning deployment.

6.1.1 Decision for a live ISO technology

The decision for the `overlay` vs. `dmsquash` dracut module depends on the features one wants to use. From a design perspective the `overlay` module is conceived for live ISO deployments on disk devices which allows the creation of a write partition or cow file. The `dmsquash` module is conceived as a generic mapping technology using device-mapper snapshots. The following list describes important live ISO features and their support status compared to the `overlay` and `dmsquash` modules.

ISO scan Usable in the same way with both dracut modules. This feature allows to boot the live ISO as a file from a grub loopback configured bootloader. The `live-grub-stick` tool is just one example that uses this feature. For details how to setup ISO scan with the `overlay` module see [Deploy ISO Image as File on a FAT32 Formated USB Stick](#)

ISO in RAM completely Usable with the `dmsquash` module through `rd.live.ram`. The `overlay` module does not support this mode but KIWI NG supports RAM only systems as OEM deployment into RAM from an install ISO media. For details how to setup RAM only deployments in KIWI NG see: [Deploy and Run System in a RamDisk](#)

Overlay based on overlayfs Usable with the `overlay` module. A squashfs compressed readonly root gets overlayed with a readwrite filesystem using the kernel overlayfs filesystem.

Overlay based on device mapper snapshots Usable with the `dmsquash` module. A squashfs compressed readonly root gets overlayed with a readwrite filesystem using a device mapper snapshot. This method was the preferred one before overlays existed in the Linux kernel.

Media Checksum Verification Boot the live iso only for ISO checksum verification. This is possible with both modules but the `overlay` module uses the `checkmedia` tool whereas the upstream `dmsquash` module uses `checkisomd5`. The activation of the verification process is done by passing the kernel option `mediacheck` for the `overlay` module and `rd.live.check` for the `dmsquash` module.

Live ISO through PXE boot Boot the live image via the network. This is possible with both modules but uses different technologies. The `overlay` module supports network boot only in combination with the AoE (Ata Over Ethernet) protocol. For details see [Booting a Live ISO Image from Network](#). The `dmsquash` module supports network boot by fetching the ISO image into memory from `root=live:` using the `livenet` module.

Persistent Data Keep new data persistent on a writable storage device. This can be done with both modules but in different ways. The `overlay` module activates persistency with the kernel boot parameter `rd.live.overlay.persistent`. If the persistent setup cannot be created the fallback to the non persistent mode applies automatically. The `overlay` module auto detects if it is used on a disk or ISO scan loop booted from a file. If booted as disk, persistency is setup on a new partition of that disk. If loop booted from file, persistency is setup on a new cow file. The cow file/partition setup can be influenced with the kernel boot parameters: `rd.live.overlay.cowfs` and `rd.live.cowfile.mbsize`. The `dmsquash` module configures persistency through the `rd.live.overlay` option exclusively and does not support the automatic creation of a write partition in disk mode.

dmsquash documentation

Documentation for the upstream `dmsquash` module can be found [here](#). Options to setup `dmsquash` are marked with `rd.live`

6.2 Build a Virtual Disk Image

Abstract

This chapter explains how to build a simple disk image, including:

- how to define a vmx image in the image description
- how to build a vmx image
- how to run it with QEMU

A Virtual Disk Image is a compressed system disk with additional metadata useful for cloud

frameworks like Amazon EC2, Google Compute Engine, or Microsoft Azure.

To instruct KIWI NG to build a VMX image add a `type` element with `image="vmx"` in `config.xml`. An example configuration for a 42 GB large VMDK image with 512 MB RAM, an IDE controller and a bridged network interface is shown below:

```
<image schemaversion="7.1" name="JeOS-Tumbleweed">
  <!-- snip -->
  <preferences>
    <type image="vmx" filesystem="ext4"
      format="vmdk" boottimeout="0"
      bootloader="grub2">
    <size unit="G">42</size>
    <machine memory="512" guestOS="suse" HWversion="4">
      <vmdisk id="0" controller="ide"/>
      <vmnic driver="e1000" interface="0" mode="bridged"/>
    </machine>
  </type>
  <!-- additional preferences -->
</preferences>
<!-- snip -->
</image>
```

The following attributes of the `type` element are of special interest when building VMX images:

- `format`: Specifies the format of the virtual disk, possible values are: `gce`, `ova`, `qcow2`, `vagrant`, `vmdk`, `vdi`, `vhd`, `vhd-x` and `vhd-fixed`.
- `formatoptions`: Specifies additional format options passed to **qemu-img**. `formatoptions` is a comma separated list of format specific options in a `name=value` format like **qemu-img** expects it. KIWI NG will forward the settings from this attribute as a parameter to the `-o` option in the **qemu-img** call.

The `size` and `machine` child-elements of `type` can be used to customize the virtual machine image further. We describe them in the following sections (see [Modifying the Size of the Image](#) and [Customizing the Virtual Machine](#)).

Once your image description is finished (or you are content with a image from the [example descriptions](#) and use one of them) build the image with KIWI NG:

```
$ sudo kiwi-ng --type vmx system build \
  --description kiwi-descriptions/suse/x86_64/suse-leap-15.1-JeOS_
→ \
  --target-dir /tmp/myimage
```

The created image will be in the target directory `/tmp/myimage` with the file extension `.raw`.

The live image can then be tested with QEMU:


```
$ qemu \
  -drive file=LimeJeOS-Leap-15.1.x86_64-1.15.1.raw,format=raw,
  ↪if=virtio \
  -m 4096
```

For further information how to setup the image to work within a cloud framework see:

- *Image Description for Amazon EC2*
- *Image Description for Microsoft Azure*
- *Image Description for Google Compute Engine*

For information how to setup a Vagrant box, see: *Image Description for Vagrant*.

6.2.1 Modifying the Size of the Image

The `size` child element of `type` specifies the size of the resulting disk image. The following example shows a image description where 20 GB are added to the virtual machine image of which 5 GB are left unpartitioned:

```
<image schemaversion="7.1" name="JeOS-Tumbleweed">
  <!-- snip -->
  <preferences>
    <type image="vmx" format="vmdk">
      <size unit="G" additive="true" unpartitioned="5">20</size>
    </type>
    <!-- additional preferences -->
  </preferences>
  <!-- snip -->
</image>
```

The following optional attributes can be used to customize the image size further:

- `unit`: Defines the unit used for the provided numerical value, possible settings are M for megabytes and G for gigabytes. The default unit are megabytes.
- `additive`: boolean value that determines whether the provided value will be added to the current image's size (`additive="true"`) or whether it is the total size (`additive="false"`). The default is false.
- `unpartitioned`: Specifies the image space in the image that will not be partitioned. This value uses the same unit as defined in the attribute `unit` or the default.

6.2.2 Customizing the Virtual Machine

The `machine` child element of `type` can be used to customize the virtual machine configuration which is used when the image is run, like the number of CPUs or the connected network interfaces.

The following attributes are supported by the `machine` element:

- `ovftype`: The OVF configuration type. The Open Virtualization Format is a standard for describing virtual appliances and distribute them in an archive called Open Virtual Appliance (OVA). The standard describes the major components associated with a disk image. The exact specification depends on the product using the format.

Supported values are `zvm`, `powervm`, `xen` and `vmware`.

- `HWversion`: The virtual machine's hardware version (`vmdk` and `ova` formats only), see <https://kb.vmware.com/s/article/1003746> for further details which value to choose.
- `arch`: the VM architecture (`vmdk` format only), possible values are: `i386` (= `i585` and `i686`) and `x86_64`.
- `xen_loader`: the Xen target loader which is expected to load this guest, supported values are: `hvmloader`, `pygrub` and `pvgrub`.
- `guestOS`: The virtual guest OS' identification string for the VM (only applicable for `vmdk` and `ova` formats, note that the name designation is different for the two formats).
- `min_memory`: The virtual machine's minimum memory in MB (`ova` format only).
- `max_memory`: The virtual machine's maximum memory in MB (`ova` format only).
- `min_cpu`: The virtual machine's minimum CPU count (`ova` format only).
- `max_cpu`: The virtual machine's maximum CPU count (`ova` format only).
- `memory`: The virtual machine's memory in MB (all formats).
- `ncpus`: The number of virtual CPUs available to the virtual machine (all formats).

Additionally, `machine` supports additional child elements that are covered in the following subsections.

Modifying the VM Configuration Directly

The `vmconfig-entry` element is used to add entries directly into the virtual machine's configuration file. This is currently only supported for the `vmdk` format where the provided strings are directly pasted into the `.vmx` file.

The `vmconfig-entry` element has no attributes and can appear multiple times, the entries are added to the configuration file in the provided order. Note, that KIWI NG does not check the entries for correctness. KIWI NG only forwards them.

The following example adds the two entries `numvcpus = "4"` and `cpuid.coresPerSocket = "2"` into the VM configuration file:


```
<image schemaversion="7.1" name="openSUSE-15.1" displayname="Bob">
  <preferences>
    <type image="vmx" filesystem="ext4" format="vmdk"
      bootloader="grub2" kernelcmdline="splash"
      bootpartition="false">
      <machine memory="512" guestOS="suse" HWversion="4">
        <vmconfig-entry>numvcpus = "4"</vmconfig-entry>
        <vmconfig-entry>cpuid.coresPerSocket = "2"</vmconfig-entry>
      </machine>
    </type>
  </preferences>
</image>
```

Adding Network Interfaces to the VM

Network interfaces can be explicitly specified for the VM when required via the `vmnic` element. This can be used to add another bridged interface or to specify the driver which is being used.

Note, that this element is only used for the `vmdk` image format.

In the following example we add a bridged network interface using the `e1000` driver:

```
<image schemaversion="7.1" name="openSUSE-15.1" displayname="Bob">
  <preferences>
    <type image="vmx" filesystem="btrfs"
      bootloader="grub2" kernelcmdline="splash">
      <machine memory="4096" guestOS="suse" HWversion="4">
        <vmnic driver="e1000" interface="0" mode="bridged"/>
      </machine>
    </type>
  </preferences>
</image>
```

The `vmnic` element supports the following attributes:

- `interface`: **mandatory** interface ID for the VM's network interface.
- `driver`: optionally the driver which will be used can be specified
- `mac`: this interfaces' MAC address
- `mode`: this interfaces' mode.

Note that KIWI NG will **not** verify the values that are passed to these attributes, it will only paste them into the appropriate configuration files.

Specifying Disks and Disk Controllers

The `vmdisk` element can be used to customize the disks and disk controllers for the virtual machine. This element can be specified multiple times, each time for each disk or disk controller present.

Note that this element is only used for `vmrk` and `ova` image formats.

The following example adds a disk with the ID 0 using an IDE controller:

```
<image schemaversion="7.1" name="openSUSE-15.1" displayname="Bob">
  <preferences>
    <type image="vmx" filesystem="ext4" format="vmrk"
      bootloader="grub2" kernelcmdline="splash"
      bootpartition="false">
      <machine memory="512" guestOS="suse" HWversion="4">
        <vmdisk id="0" controller="ide"/>
      </machine>
    </type>
  </preferences>
</image>
```

Each `vmdisk` element can be further customized via the following optional attributes:

- `controller`: The disk controller used for the VM guest (`vmrk` format only). Supported values are: `ide`, `buslogic`, `lsilogic`, `lsisas1068`, `legacyESX` and `pvscsi`.
- `device`: The disk device to appear in the guest (`xen` format only).
- `diskmode`: The disk mode (`vmrk` format only), possible values are: `monolithicSparse`, `monolithicFlat`, `twoGbMaxExtentSparse`, `twoGbMaxExtentFlat` and `streamOptimized` (see also https://www.vmware.com/support/developer/converter-sdk/conv60_apireference/vim.OvfManager.CreateImportSpecParams.DiskProvisioningType.html).
- `disktype`: The type of the disk as it is internally handled by the VM (`ova` format only). This attribute is currently unused.
- `id`: The disk ID of the VM disk (`vmrk` format only).

Adding CD/DVD Drives

KIWI NG supports the addition of IDE and SCSI CD/DVD drives to the virtual machine using the `vmvdvd` element for the `vmrk` image format. In the following example we add two drives: one with a SCSI and another with a IDE controller:

```
<image schemaversion="7.1" name="openSUSE-15.1" displayname="Bob">
  <preferences>
    <type bootloader="grub2" filesystem="ext4"
      image="vmx" kernelcmdline="splash">
```

(continues on next page)

(continued from previous page)

```
<machine memory="512" xen_loader="hvmloader">
  <vmdvd id="0" controller="scsi"/>
  <vmdvd id="1" controller="ide"/>
</machine>
</type>
</preferences>
</image>
```

The `vmdvd` element features just these two **mandatory** attributes:

- `id`: The CD/DVD ID of the drive
- `controller`: The CD/DVD controller used for the VM guest, supported values are `ide` and `scsi`.

6.3 Build an OEM Expandable Disk Image

Abstract

This page explains how to build an OEM disk image. It contains:

- how to build an OEM image
- how to deploy an OEM image
- how to run the deployed system

An OEM disk represents the system disk with the capability to auto expand the disk and its filesystem to a custom disk geometry. This allows deploying the same OEM image on target systems of a different hardware setup.

The following example shows how to build and deploy an OEM disk image based on openSUSE Leap using a QEMU virtual machine as OEM target system:

1. Make sure you have checked out the example image descriptions, see [Example Appliance Descriptions](#).
2. Build the image with KIWI NG:

```
$ sudo kiwi-ng --type oem system build \
  --description kiwi-descriptions/suse/x86_64/suse-leap-15.1-
→JeOS \
  --target-dir /tmp/myimage
```

Find the following result images below `/tmp/myimage`.

- The OEM disk image with the suffix `.raw` is an expandable virtual disk. It can expand itself to a custom disk geometry.

- The OEM installation image with the suffix `install.iso` is a hybrid installation system which contains the OEM disk image and is capable to install this image on any target disk.

6.3.1 Deployment Methods

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.

There are the following basic deployment strategies:

1. *Manual Deployment*

Manually deploy the OEM disk image onto the target disk

2. *CD/DVD Deployment*

Boot the OEM installation image and let KIWI NG's OEM installer deploy the OEM disk image from CD/DVD or USB stick onto the target disk

3. *Network Deployment*

PXE boot the target system and let KIWI NG's OEM installer deploy the OEM disk image from the network onto the target disk

6.3.2 Manual Deployment

The manual deployment method can be tested using virtualization software such as QEMU, and an additional virtual target disk of a larger size. The following steps shows how to do it:

1. Create a target disk

```
$ qemu-img create target_disk 20g
```

Note: Retaining the Disk Geometry

If the target disk geometry is less or equal to the geometry of the OEM disk image itself, the disk expansion performed for a physical disk install during the OEM boot workflow will be skipped and the original disk geometry stays untouched.

2. Dump OEM image on target disk

```
$ dd if=LimeJeOS-Leap-15.1.x86_64-1.15.1.raw of=target_disk_  
→conv=notrunc
```

3. Boot the target disk

```
$ qemu -hda target_disk -m 4096
```


At first boot of the target_disk the system is expanded to the configured storage layout. By default the system root partition and filesystem is resized to the maximum free space available.

6.3.3 CD/DVD Deployment

The deployment from CD/DVD via the installation image can also be tested using virtualization software such as QEMU. The following steps shows how to do it:

1. Create a target disk

Follow the steps above to create a virtual target disk

2. Boot the OEM installation image as CD/DVD with the target disk attached

```
$ qemu -cdromLimeJeOS-Leap-15.1.x86_64-1.15.1.install.iso -hda ↵  
↪target_disk -boot d -m 4096
```

Note: USB Stick Deployment

Like any other iso image built with KIWI NG, also the OEM installation image is a hybrid image. Thus it can also be used on USB stick and serve as installation stick image like it is explained in *Build an ISO Hybrid Live Image*

6.3.4 Network Deployment

The deployment from the network downloads the OEM disk image from a PXE boot server. This requires a PXE network boot server to be setup as explained in *Setting Up a Network Boot Server*

If the PXE server is running the following steps shows how to test the deployment process over the network using a QEMU virtual machine as target system:

1. Make sure to create an installation PXE TAR archive along with your OEM image by replacing the following setup in kiwi-descriptions/suse/x86_64/suse-leap-15.1-JeOS/config.xml

Instead of

```
<type image="oem" installiso="true"/>
```

setup

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

2. Rebuild the image, unpack the resulting LimeJeOS-Leap-15.1.x86_64-1.15.1.install.tar.xz file to a temporary directory and copy the initrd and kernel images to the PXE server:

a) Unpack installation tarball

```
mkdir /tmp/pxe && cd /tmp/pxe
tar -xzf LimeJeOS-Leap-15.1.x86_64-1.15.1.install.tar.xz
```

b) Copy kernel and initrd used for pxe boot

```
scp pxeboot.LimeJeOS-Leap-15.1.x86_64-1.15.1.initrd.xz PXE_
↪SERVER_IP:/srv/tftpboot/boot/initrd
scp pxeboot.LimeJeOS-Leap-15.1.x86_64-1.15.1.kernel PXE_
↪SERVER_IP:/srv/tftpboot/boot/linux
```

3. Copy the OEM disk image, MD5 file, system kernel, initrd and bootoptions to the PXE boot server:

Activation of the deployed system is done via `kexec` of the kernel and initrd provided here.

a) Copy system image and MD5 checksum

```
scp LimeJeOS-Leap-15.1.x86_64-1.15.1.xz PXE_SERVER_IP:/srv/
↪tftpboot/image/
scp LimeJeOS-Leap-15.1.x86_64-1.15.1.md5 PXE_SERVER_IP:/srv/
↪tftpboot/image/
```

b) Copy kernel, initrd and bootoptions used for booting the system via kexec

```
scp LimeJeOS-Leap-15.1.x86_64-1.15.1.initrd PXE_SERVER_IP:/
↪srv/tftpboot/image/
scp LimeJeOS-Leap-15.1.x86_64-1.15.1.kernel PXE_SERVER_IP:/
↪srv/tftpboot/image/
scp LimeJeOS-Leap-15.1.x86_64-1.15.1.config.bootoptions PXE_
↪SERVER_IP:/srv/tftpboot/image/
```

Note: The `config.bootoptions` file is used together with `kexec` to boot the previously dumped image. The information in that file references the root of the dumped image and can also include any other type of boot options. The file provided with the KIWI NG built image is by default connected to the image present in the PXE TAR archive. If other images got deployed the contents of this file must be adapted to match the correct root reference.

4. Add/Update the kernel command line parameters

Edit your PXE configuration (for example `pxelinux.cfg/default`) on the PXE server and add these parameters to the append line, typically looking like this:

```
append initrd=boot/initrd rd.kiwi.install.pxe rd.kiwi.install.
↪image=tftp://192.168.100.16/image/LimeJeOS-Leap-15.1.x86_64-1.
↪15.1.xz
```


The location of the image is specified as a source URI which can point to any location supported by the `curl` command. KIWI NG calls `curl` to fetch the data from this URI. This also means your image, MD5 file, system kernel and initrd could be fetched from any server and doesn't have to be stored on the `PXE_SERVER`.

By default KIWI NG does not use specific `curl` options or flags. However it is possible to add custom ones by adding the `rd.kiwi.install.pxe.curl_options` flag into the kernel command line. `curl` options are passed as comma separated values. Consider the following example:

```
rd.kiwi.install.pxe.curl_options=--retry,3,--retry-delay,3,--  
→speed-limit,2048
```

The above tells KIWI NG to call `curl` with:

```
curl --retry 3 --retry-delay 3 --speed-limit 2048 -f <url>
```

This is specially handy when the deployment infrastructure requires some fine tuned download behavior. For example, setting retries to be more robust over flaky network connections.

Note: KIWI NG just replaces commas with spaces and appends it to the `curl` call. This is relevant since command line options including commas will always fail.

Note: The initrd and Linux Kernel for pxe boot are always loaded via tftp from the `PXE_SERVER`.

4. Create a target disk

Follow the steps above to create a virtual target disk

5. Connect the client to the network and boot QEMU with the target disk attached to the virtual machine.

```
$ qemu -boot n -hda target_disk -m 4096
```

Note: QEMU bridged networking

In order to let `qemu` connect to the network we recommend to setup a network bridge on the host system and let `qemu` connect to it via a custom `/etc/qemu-ifup`. For details see <https://en.wikibooks.org/wiki/QEMU/Networking>

6.4 Build a PXE Root File System Image

Abstract

This page explains how to build a file system image for use with KIWI NG's PXE boot infrastructure. It contains:

- how to build a PXE file system image
- how to setup the PXE file system image on the PXE server
- how to run it with QEMU

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

A root filesystem image which can be deployed via KIWI NG's PXE netboot infrastructure represents the system rootfs in a linux filesystem. A user could loop mount the image and access the contents of the root filesystem. The image does not contain any information about the system disk its partitions or the bootloader setup. All of these information is provided by a client configuration file on the PXE server which controls how the root filesystem image should be deployed.

Many different deployment strategies are possible, e.g root over **NBD** (network block device), **AoE** (ATA over Ethernet), or NFS for diskless and diskfull clients. This particular example shows how to build an overlayfs-based union system based on openSUSE Leap for a diskless client which receives the squashfs compressed root file system image in a ramdisk overlayed via overlayfs and writes new data into another ramdisk on the same system. As diskless client, a QEMU virtual machine is used.

Things to know before

- To use the image, all image parts need to be copied to the PXE boot server. If you have not set up such a server, refer to *[Setting Up a Network Boot Server](#)* for instructions.
 - The following example assumes you will create the PXE image on the PXE boot server itself (if not, use **scp** to copy the files on the remote host).
 - To let QEMU connect to the network, we recommend to setup a network bridge on the host system and let QEMU connect to it via a custom `/etc/qemu-ifup`. For details, see <https://en.wikibooks.org/wiki/QEMU/Networking>
 - The PXE root filesystem image approach is considered to be a legacy setup. The required netboot initrd code will be maintained outside of the KIWI NG appliance builder code base. If possible, we recommend to switch to the OEM disk image deployment via PXE.
1. Make sure you have checked out the example image descriptions, see *[Example Appliance Descriptions](#)*.

2. Build the image with KIWI NG:

```
$ sudo kiwi-ng --type pxe system build \  
  --description kiwi-descriptions/suse/x86_64/suse-leap-15.1-  
→JeOS \  
  --target-dir /tmp/mypxe-result
```

3. Change into the build directory:

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

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

```
$ cp *.initrd.xz /srv/tftpboot/boot/initrd  
$ cp *.kernel /srv/tftpboot/boot/linux
```

5. Copy the system image and its MD5 sum to /srv/tftpboot/image:

```
$ cp LimeJeOS-Leap-15.1.x86_64-1.15.1/srv/tftpboot/image  
$ cp LimeJeOS-Leap-15.1.x86_64-1.15.1.md5 /srv/tftpboot/image
```

6. Adjust the PXE configuration file. The configuration file controls which kernel and initrd is loaded and which kernel parameters are set. A template has been installed at /srv/tftpboot/pxelinux.cfg/default from the kiwi-pxeboot package. The minimal configuration required to boot the example image looks like to following:

```
DEFAULT KIWI-Boot  
  
LABEL KIWI-Boot  
  kernel boot/linux  
  append initrd=boot/initrd  
  IPAPPEND 2  
  
Additional configuration files can be found at :ref:`pxe_client_  
→config`.
```

7. Create the image client configuration file:

```
$ vi /srv/tftpboot/KIWI/config.default  
  
IMAGE=/dev/ram1;LimeJeOS-Leap-15.1.x86_64;1.15.1;192.168.100.2;  
→4096  
UNIONFS_CONFIG=/dev/ram2,/dev/ram1,overlay
```

All PXE boot based deployment methods are controlled by a client configuration file. The above configuration tells the client where to find the image and how to activate it. In this case the image will be deployed into a ramdisk (ram1) and overlay mounted such that all write operations will land in another ramdisk (ram2). KIWI NG supports a variety of different deployment strategies based on the rootfs image created beforehand. For details, refer to *PXE Client Setup Configuration*

8. Connect the client to the network and boot. This can also be done in a virtualized environment using QEMU as follows:

```
$ qemu -boot n -m 4096
```

6.5 Build a Docker Container Image

Abstract

This page explains how to build a Docker base image. It contains

- basic configuration explanation
- how to build a Docker image
- how to run it with the Docker daemon

KIWI NG is capable of building native Docker images, from scratch and derived ones. KIWI NG Docker images are considered to be native since the KIWI NG tarball image is ready to be loaded to a Docker daemon, including common container configurations.

The Docker configuration metadata is provided to KIWI NG as part of the *XML description file* using the `<containerconfig>` tag. The following configuration metadata can be specified:

`containerconfig` attributes:

- `name`: Specifies the repository name of the Docker image.
- `tag`: Sets the tag of the Docker image.
- `maintainer`: Specifies the author field of the container.
- `user`: Sets the user name or user id (UID) to be used when running `entrypoint` and subcommand. Equivalent of the `USER` directive of a Docker file.
- `workingdir`: Sets the working directory to be used when running `cmd` and `entrypoint`. Equivalent of the `WORKDIR` directive of a Docker file.

`containerconfig` child tags:

- `subcommand`: Provides the default execution parameters of the container. Equivalent of the `CMD` directive of a Docker file.
- `labels`: Adds custom metadata to an image using key-value pairs. Equivalent to one or more `LABEL` directives of a Docker file.
- `expose`: Informs at which ports is the container listening at runtime. Equivalent to one or more `EXPOSE` directives of a Docker file.
- `environment`: Sets an environment values using key-value pairs. Equivalent to one or more the `env` directives of a Docker file.

- `entrypoint`: Sets the command that the container will run, it can include parameters. Equivalent of the `ENTRYPOINT` directive of a Docker file.
- `volumes`: Create mountpoints with the given name and mark it to hold external volumes from the host or from other containers. Equivalent to one or more `VOLUME` directives of a Docker file.

Other Docker file directives such as `RUN`, `COPY` or `ADD`, can be mapped to KIWI NG by using the *config.sh* script file to run bash commands or the *overlay tree* to include extra files.

The following example shows how to build a Docker base image based on openSUSE Leap:

1. Make sure you have checked out the example image descriptions, see *Example Appliance Descriptions*.
2. Include the `Virtualization/containers` repository to your list:

```
$ zypper addrepo http://download.opensuse.org/repositories/  
→Virtualization:/containers/<DIST> container-tools
```

where the placeholder `<DIST>` is the preferred distribution.

3. Install **umoci** and **skopeo** tools

```
$ zypper in umoci skopeo
```

4. Build the image with KIWI NG:

```
$ sudo kiwi-ng --type docker system build \  
    --description kiwi-descriptions/suse/x86_64/suse-tumbleweed-  
→docker \  
    --target-dir /tmp/myimage
```

5. Test the Docker image.

First load the new image

```
$ docker load -i openSUSE-Tumbleweed-container-image.x86_64-1.0.  
→4.docker.tar.xz
```

then run the loaded image:

```
$ docker run -it opensuse:42.2 /bin/bash
```


6.6 Build a WSL Container Image

Abstract

This page explains how to build a WSL/Appx container image. WSL stands for Windows Subsystem Linux and is a zip based container format consumable by Windows 10 with enabled WSL functionality.

KIWI NG is capable of building WSL images using the **appx** utility. Make sure you have installed a package that provides this command on your build host.

Once the build host has the **appx** installed, the following image type setup is required in the XML description `config.xml`:

```
<type image="appx" metadata_path="/meta/data"/>
```

The `/meta/data` path specifies a path that provides additional information required for the **WSL-DistroLauncher**. This component consists out of a `Windows.exe` executable file and an `AppxManifest.xml` file which references other files like icons and resource configurations for the startup of the container under Windows.

Note: `/meta/data`

Except for the root filesystem tarball KIWI NG is not responsible for providing the meta data required for the **WSL-DistroLauncher**. It is expected that the given metadata path contains all the needed information. Typically this information is delivered in a package provided by the Distribution and installed on the build host

6.6.1 Setup of the WSL-DistroLauncher

The contents of the `AppxManifest.xml` will be changed by KIWI NG if a `containerconfig` section is provided in the XML description. In the context of a WSL image the following container configuration parameters are taken into account:

```
<containerconfig name="my-wsl-container">
  <history
    created_by="Organisation"
    author="Name"
    application_id="AppIdentification"
    package_version="https://docs.microsoft.com/en-us/windows/
    ↪ uwp/publish/package-version-numbering"
    launcher="WSL-DistroLauncher-exe-file"
  >Container Description Text</history>
</containerconfig>
```


All information provided here including the entire section is optional. If not provided the existing `AppxManifest.xml` stays untouched.

created_by Provides the name of a publisher organisation. An appx container needs to be signed off with a digital signature. If the image is build in the Open Build Service (OBS) this happens automatically. Outside of OBS one needs to make sure the given publisher organisation name matches the certificate used for signing.

author Provides the name of the author and maintainer of this container

application_id Provides an ID name for the container. The name must start with a letter and only allows alphanumeric characters. KIWI NG will not validate the given name string because there is no common criteria between the container architectures. KIWI NG just accepts any text.

package_version Provides the version identification for the container. KIWI NG validates this against the [Microsoft Package Version Numbering](#) rules.

launcher Provides the binary file name of the launcher `.exe` file.

Warning: There is no validation by KIWI NG if the contents of `AppxManifest.xml` are valid or complete to run the container. Users will find out at call time, not before

The following example shows how to build a WSL image based on openSUSE TW:

1. Make sure you have checked out the example image descriptions, see [Example Appliance Descriptions](#).
2. Include the `Virtualization/WSL` repository to your list:

```
$ zypper addrepo http://download.opensuse.org/repositories/  
→Virtualization:/WSL/<DIST> WSL
```

where the placeholder `<DIST>` is the preferred distribution.

3. Install **fb-util-for-appx** utility and a package that provides the **WSL-DistroLauncher** metadata. See the above note about `/meta/data`

```
$ zypper in fb-util-for-appx DISTRO_APPX_METADATA_PACKAGE
```

Note: If you are building in the Open Build Service make sure to add the packages from the zypper call above to your project config via `osc meta -e prjconf` and a line of the form `support: PACKAGE_NAME` for each package that needs to be installed on the Open Build Service worker that runs the KIWI NG build process.

4. Setup the image type:

Edit the XML description file: `kiwi-descriptions/suse/x86_64/suse-tumbleweed-JeOS/config.xml` and add the following type and containerconfig:


```
<type image="appx" metadata_path="/meta/data">
  <containerconfig name="Tumbleweed">
    <history
      created_by="SUSE"
      author="KIWI-Team"
      application_id="tumbleweed"
      package_version="2003.12.0.0"
      launcher="openSUSE-Tumbleweed.exe"
    >TW JeOS text based</history>
  </containerconfig>
</type>
```

Warning: If the configured metadata path does not exist the build will fail. Furthermore there is no validation by KIWI NG that the contents of the metadata path are valid or complete with respect to the requirements of the **WSL-DistroLauncher**

5. Build the image with KIWI NG:

```
$ sudo kiwi-ng --type appx system build \
  --description kiwi-descriptions/suse/x86_64/suse-tumbleweed-
  ↪JeOS \
  --target-dir /tmp/myimage
```

6.6.2 Testing the WSL image

For testing the image a Windows 10 system is required. As a first step the optional feature named `Microsoft-Windows-Subsystem-Linux` must be enabled. For further details on how to setup the Windows machine see the following documentation: [Windows Subsystem for Linux](#)

6.7 Building in a Self-Contained Environment

Note: Abstract

Users building images with KIWI NG face problems if they want to build an image matching one of the following criteria:

- build should happen as non root user.
- build should happen on a host system distribution for which no KIWI NG packages exists.
- build happens on an incompatible host system distribution compared to the target image distribution. For example the host system rpm database is incompatible with the image

rpm database and a dump/reload cycle is not possible between the two versions. Ideally the host system distribution is the same as the target image distribution.

This document describes how to perform the build process in a Docker container using the Dice containment build system written for KIWI NG in order to address the issues listed above.

The changes on the machine to become a build host will be reduced to the requirements of Dice and Docker.

6.7.1 Requirements

The following components need to be installed on the build system:

- Dice - a containment build system for KIWI NG.
- Docker - a container framework based on the Linux container support in the kernel.
- Docker Image - a docker build container for KIWI NG.
- optionally Vagrant - a framework to run, provision and control virtual machines and container instances. Vagrant has a very nice interface to provision a machine prior to running the actual build. It also supports docker as a provider which makes it a perfect fit for complex provisioning tasks in combination with the Docker container system.
- optionally libvirt - Toolkit to interact with the virtualization capabilities of Linux. In combination with vagrant, libvirt can be used as provider for provision and control full virtual instances running via qemu. As docker shares the host system kernel and thus any device, because KIWI NG needs to use privileged docker containers for building images, the more secure but less performant solution is to use virtual machines to run the KIWI NG build.

6.7.2 Installing and Setting up Dice

The Dice packages and sources are available at the following locations:

- Build service project: <http://download.opensuse.org/repositories/Virtualization:/Appliances:/ContainerBuilder>
- Sources: <https://github.com/OSInside/dice>

```
$ sudo zypper in ruby[VERSION]-rubygem-dice
```


6.7.3 Installing and Setting up Docker

Docker packages are usually available with the used distribution.

```
$ sudo zypper in docker
```

Make sure that the user, who is intended to build images, is a member of the `docker` group. Run the following command:

```
$ sudo useradd -G docker <builduser>
```

It is required to logout and login again to let this change become active.

Once this is done you need to setup the Docker storage backend. By default Docker uses the device mapper to manage the storage for the containers it starts. Unfortunately, this does not work if the container is supposed to build images because it runs into conflicts with tools like **kpartx** which itself maps devices using the device mapper.

Fortunately, there is a solution for Docker which allows us to use Btrfs as the storage backend. The following is only required if your host system root filesystem is not btrfs:

```
$ sudo qemu-img create /var/lib/docker-storage.btrfs 20g
$ sudo mkfs.btrfs /var/lib/docker-storage.btrfs
$ sudo mkdir -p /var/lib/docker
$ sudo mount /var/lib/docker-storage.btrfs /var/lib/docker

$ sudo vi /etc/fstab

    /var/lib/docker-storage.btrfs /var/lib/docker btrfs defaults 0 0

$ sudo vi /etc/sysconfig/docker

    DOCKER_OPTS="-s btrfs"
```

Finally start the docker service:

```
$ sudo systemctl restart docker
```

6.7.4 Installing and Setting up the Build Container

In order to build in a contained environment Docker has to start a privileged system container. Such a container must be imported before Docker can use it. The build container is provided to you as a service and build with KIWI NG in the project at <http://download.opensuse.org/repositories/Virtualization:/Appliances:/Images>. The result image is pushed to <https://hub.docker.com/r/opensuse/dice>.

When building with Dice, the container will be automatically fetched from the docker registry. However this step can also be done prior to calling **dice** as follows:


```
$ docker pull opensuse/dice:latest
```

Note: Optional step

If a custom or newer version of the Build Container should be used, it is required to update the registry. This is because Dice always fetches the latest version of the Build Container from the registry.

1. Download the .tar.bz2 file which starts with Docker-Tumbleweed

```
$ wget http://download.opensuse.org/repositories/Virtualization:/
↪Appliances:/Images/images/Docker-Tumbleweed.XXXXXXX.docker.tar.xz
```

2. Import the downloaded tarball with the command **docker:**

```
$ docker load -i Docker-Tumbleweed.XXXXXXX.docker.tar
```

3. Tag the container and push back to the registry

```
$ docker push opensuse/dice:latest
```

6.7.5 Installing and Setting up Vagrant

Note: Optional step

By default Dice shares the KIWI NG image description directory with the Docker instance. If more data from the host should be shared with the Docker instance we recommend to use Vagrant for this provision tasks.

Installing Vagrant is well documented at <https://www.vagrantup.com/docs/installation/index.html>

Access to a machine started by Vagrant is done through SSH exclusively. Because of that an initial key setup is required in the Docker image vagrant should start. The KIWI NG Docker image includes the public key of the Vagrant key pair and thus allows access. It is important to understand that the private Vagrant key is not a secure key because the private key is not protected.

However, this is not a problem because Vagrant creates a new key pair for each machine it starts. In order to allow Vagrant the initial access and the creation of a new key pair, it's required to provide access to the insecure Vagrant private key. The following commands should not be executed as root, but as the intended user to build images.

```
$ mkdir -p ~/.dice/key
$ cp -a /usr/share/doc/packages/ruby*-rubygem-dice/key ~/.dice/key
```


6.7.6 Configuring Dice

If you build in a contained environment, there is no need to have KIWI NG installed on the host system. KIWI NG is part of the container and is only called there. However, a KIWI NG image description and some metadata defining how to run the container are required as input data.

6.7.7 Selecting a Template

If you don't have a KIWI NG description select one from the templates provided at the GitHub project hosting example appliance descriptions.

```
$ git clone https://github.com/OSInside/kiwi-descriptions
```

The descriptions hosted here also provides a default `Dicefile` as part of each image description.

6.7.8 The Dicefile

The Dicefile is the configuration file for the dice buildsystem backend. All it needs to know for a plain docker based build process is the selection of the buildhost to be a Docker container. The Dicefile's found in the above mentioned appliance descriptions look all like the following:

```
Dice.configure do |config|
  config.buildhost = :DOCKER
end
```

6.7.9 Building with Dice

If you have chosen to just use the default Dice configuration as provided with the example appliance descriptions, the following example command will build the image:

```
$ cd <git-clone-result-kiwi-descriptions>

$ dice build suse/x86_64/suse-leap-15.1-JeOS
$ dice status suse/x86_64/suse-leap-15.1-JeOS
```


6.7.10 Buildsystem Backends

Dice currently supports three build system backends:

1. Host buildsystem - Dice builds on the host like if you would call KIWI NG on the host directly.
2. Vagrant Buildsystem - Dice uses Vagrant to run a virtual system which could also be a container and build the image on this machine.
3. Docker buildsystem - Dice uses Docker directly to run the build in a container

The use of the Docker buildsystem has been already explained in the above chapters. The following sections explains the pros and cons of the other two available Buildsystem Backends.

6.7.11 Building with the Host Buildsystem

Using the Host Buildsystem basically tells Dice to ssh into the specified machine with the specified user and run KIWI NG. This is also the information which needs to be provided in a Dicefile. Using the Host Buildsystem is recommended if there are dedicated build machines available to take over KIWI NG build jobs.

6.7.12 The Dicefile

```
Dice.configure do |config|
  config.buildhost = "full-qualified-dns-name-or-ip-address"
  config.ssh_user = "vagrant"
end
```

After these changes a **dice build** command will make use of the Host Buildsystem and starts the KIWI NG build process there.

Note: Provisioning of the Host Buildsystem

There is no infrastructure in place which manages the machine specified as config.buildhost. This means it is currently in the responsibility of the user to make sure the specified machine exists and is accessible via the configured user. For the future we plan to implement a Public Cloud Buildsystem which then will allow provisioning and management of a public cloud instance e.g on Amazon EC2 in order to run the build. However we are not there yet.

6.7.13 Building with the Vagrant Buildsystem

Using the Vagrant Buildsystem should be considered if one or both of the following use cases applies:

1. The build task requires additional content or logic before the build can start. Vagrant serves as provisioning system to share data from the host with the guest containers.
2. The build task should run in a completely isolated virtual machine environment. Vagrant in combination with the libvirt provider serves as both; The tool to interact with the virtualization capabilities to run and manage virtual machine instances and as provisioning system to share data from the host with the virtual machines.

6.7.14 The Dicefile

The Dicefile in the context of Vagrant needs to know the user name to access the instance. The reason for this is, in Vagrant access to the system is handled over SSH.

```
Dice.configure do |config|
  config.ssh_user = "vagrant"
end
```

6.7.15 The Vagrant setup for the Docker Provider

The following is an example for the first use case and describes how to configure Dice to use Docker in combination with Vagrant as provisioning system.

6.7.16 The Vagrantfile

The existence of a Vagrantfile tells Dice to use Vagrant as Buildsystem. Once you call **dice** to build the image it will call **vagrant** to bring up the container. In order to allow this, we have to tell Vagrant to use Docker for this task and provide parameters on how to run the container. At the same place the Dicefile exists we create the Vagrantfile with the following content:

```
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.provider "docker" do |d|
    d.image = "opensuse/dice:latest"
    d.create_args = ["-privileged=true", "-i", "-t"]
    # start the sshd in foreground to keep the container in running_
    ↪state
    d.cmd = ["/usr/sbin/sshd", "-D"]
    d.has_ssh = true
  end
end
```


After these changes a **dice build** command will make use of the Vagrant build system and offers a nice way to provision the Docker container instances prior to the actual KIWI NG build process. Vagrant will take over the task to run and manage the docker container via the `docker` tool chain.

6.7.17 The Vagrant setup for the libvirt Provider

The following sections are an example for the second use case and describes how to configure Dice to use libvirt in combination with Vagrant as provisioning and virtualization system.

6.7.18 The Vagrant Build Box

Apart from the Docker build container the Dice infrastructure also provides a virtual machine image also known as vagrant box which contains a system ready to build images with KIWI NG.

Download the Vagrant build box which starts with `Vagrant-Libvirt-Tumbleweed` from the Open BuildService and add the box to vagrant as follows:

```
$ wget http://download.opensuse.org/repositories/Virtualization:/
→Appliances:/Images/images/Vagrant-Libvirt-Tumbleweed.XXXXXXX.
→vagrant.libvirt.box

$ vagrant box add --provider libvirt --name kiwi-build-box Vagrant-
→Libvirt-Tumbleweed.XXXXXXX.vagrant.libvirt.box

$ export VAGRANT_DEFAULT_PROVIDER=libvirt
```

The command **vagrant box list** must list the box with name `kiwi-build-box` as referenced in the following Vagrantfile setup.

6.7.19 The Vagrantfile

```
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "kiwi-build-box"

  config.vm.provider "libvirt" do |lv|
    lv.memory = "1024"
  end
end
```

After these changes a **dice build** command will make use of the Vagrant build system and offers a nice way to provision fully isolated qemu instances via libvirt prior to the actual KIWI NG build process. Vagrant will take over the task to run and manage the virtual machines via the `libvirt` tool chain.

6.8 Building Images with Profiles

KIWI NG supports so-called *profiles* inside the XML image description. Profiles act as namespaces for additional settings to be applied on top of the defaults. For further details, see [Image Profiles](#).

6.8.1 Local Builds

To execute local KIWI NG builds with a specific, selected profile, add the command line flag `--profile=$PROFILE_NAME`:

```
$ sudo kiwi-ng --type iso system build \
    --profile=workstation \
    --description kiwi-descriptions/suse/x86_64/suse-leap-15.1-
↪JeOS \
    --target-dir /tmp/myimage
```

Consult the manual page of `kiwi` for further details: [SYNOPSIS](#).

6.8.2 Building with the Open Build Service

The Open Build Service (OBS) support profiles via the [multibuild](#) feature. An example project using this feature is the [openSUSE-Tumbleweed-JeOS](#) image.

To enable and use the profiles, follow these steps:

1. Add the following XML comment to your `config.xml`:

```
<!-- OBS-Profiles: @BUILD_FLAVOR@ -->
```

It must be added before the opening `<image>` element and after the `<?xml?>` element, e.g.:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- OBS-Profiles: @BUILD_FLAVOR@ -->
<image schemaversion="7.1" name="LimeJeOS-Leap-15.1">
  <!-- snip -->
</image>
```

2. Add a file `_multibuild` into your package's repository with the following contents:

```
<multibuild>
  <flavor>profile_1</flavor>
  <flavor>profile_2</flavor>
</multibuild>
```

Add a line `<flavor>$PROFILE</flavor>` for each profile that you want OBS to build.

Note, by default, OBS excludes the build **without** any profile enabled.

Running a build of a multibuild enabled repository via `osc` can be achieved via the `-M $PROFILE` flag:

```
$ osc build -M $PROFILE
```

6.9 Building in the Open Build Service

Note: Abstract

This document gives a brief overview how to build images with KIWI NG in version 9.20.5 inside of the Open Build Service. A tutorial on the Open Builds service itself can be found here: https://en.opensuse.org/openSUSE:Build_Service_Tutorial

The next generation KIWI NG is fully integrated with the Open Build Service. In order to start it's best to checkout one of the integration test image build projects from the base Testing project `Virtualization:Appliances:Images:Testing_$ARCH:$DISTRO` at:

<https://build.opensuse.org>

For example the test images for SUSE on x86 can be found [here](#).

6.9.1 Advantages of using the Open Build Service (OBS)

The Open Build Service offers multiple advantages over running KIWI NG locally:

- OBS will host the latest successful build for you without having to setup a server yourself.
- As KIWI NG is fully integrated into OBS, OBS will automatically rebuild your images if one of the included packages or one of its dependencies or KIWI NG itself get updated.
- The builds will no longer have to be executed on your own machine, but will run on OBS, thereby saving you resources. Nevertheless, if a build fails, you get a notification via email (if enabled in your user's preferences).

6.9.2 Differences Between Building Locally and on OBS

Note, there is a number of differences when building images with KIWI NG using the Open Build Service. Your image that build locally just fine, might not build without modifications.

The notable differences to running KIWI NG locally include:

- OBS will pick the KIWI NG package from the repositories configured in your project, which will most likely not be the same version that you are running locally. This is especially relevant when building images for older versions like SUSE Linux Enterprise. Therefore, include the custom appliances repository as described in the following section: *Recommendations*.

- When KIWI NG runs on OBS, OBS will extract the list of packages from `config.xml` and use it to create a build root. In contrast to a local build (where your distributions package manager will resolve the dependencies and install the packages), OBS will **not** build your image if there are multiple packages that could be chosen to satisfy the dependencies of your packages¹. This shows errors like this:

```
unresolvable: have choice for SOMEPACKAGE: SOMEPACKAGE_1_
↳SOMEPACKAGE_2
```

This can be solved by explicitly specifying one of the two packages in the project configuration via the following setting:

```
Prefer: SOMEPACKAGE_1
```

Place the above line into the project configuration, which can be accessed either via the web interface (click on the tab `Project Config` on your project's main page) or via `osc meta -e prjconf`.

Warning: We strongly encourage you to remove your repositories from `config.xml` and move them to the repository configuration in your project's settings. This usually prevents the issue of having the choice for multiple package version and results in a much smoother experience when using OBS.

- By default, OBS builds only a single build type and the default profile. If your appliance uses multiple build types, put each build type into a profile, as OBS cannot handle multiple build types.

There are two options to build multiple profiles on OBS:

1. Use the `<image>` element and add it bellow the XML declaration (`<?xml ..?>`):

```
<?xml version="1.0" encoding="utf-8"?>

<!-- OBS-Profiles: foo_profile bar_profile -->

<image schemaversion="7.1" name="openSUSE-Leap-15.1">
  <!-- image description with the profiles foo_profile and
  ↳bar_profile
</image>
```

2. Use the `multibuild` feature.

The first option is simpler to use, but has the disadvantage that your appliances are built sequentially. The `multibuild` feature allows to build each profile as a single package, thereby enabling parallel execution, but requires an additional `_multibuild` file. For the above example `config.xml` would have to be adapted as follows:

¹ This is a design decision made by OBS: as it's purpose is to build packages in a reproducible fashion it cannot make a decision which package to choose from multiple available ones. A package manager build for end-users on the other hand **must** make an a choice, as it would be otherwise hardly usable.


```
<?xml version="1.0" encoding="utf-8"?>

<!-- OBS-Profiles: @BUILD_FLAVOR@ -->

<image schemaversion="7.1" name="openSUSE-Leap-15.1">
  <!-- image description with the profiles foo_profile and bar_
  ↪profile
</image>
```

The file `_multibuild` would have the following contents:

```
<multibuild>
  <flavor>foo_profile</flavor>
  <flavor>bar_profile</flavor>
</multibuild>
```

- Subfolders in OBS projects are ignored by default by **osc** and must be explicitly added via **osc add \$FOLDER**². Bear that in mind when adding the overlay files inside the `root/` directory to your project.
- OBS ignores file permissions. Therefore `config.sh` and `images.sh` will **always** be executed through BASH (see also: *User Defined Scripts*).

6.9.3 Recommendations

Working with OBS

Although OBS is an online service, it is not necessary to test every change by uploading it. OBS will use the same process as `osc build` does, so if your image builds locally via `osc build` it should also build online on OBS.

Repository Configuration

When setting up the project, enable the `images` repository: the `images` repository's checkbox can be found at the bottom of the selection screen that appears when clicking `Add from a Distribution` in the `Repositories` tab. Or specify it manually in the project configuration (it can be accessed via `osc meta -e prj`):

```
<repository name="images">
  <arch>x86_64</arch>
</repository>
```

Furthermore, OBS requires additional repositories from which it obtains your dependent packages. These repositories can be provided in two ways:

² `osc` compresses added folders into a `cpio` archive and decompresses it before running your builds. The only downside of this is, that the contents of your overlay is not conveniently visible via the web interface.

1. Add the repositories to the project configuration on OBS and omit them from `config.xml`. Provide only the following repository inside the image description:

```
<repository type="rpm-md">
  <source path="obsrepositories:"/>
</repository>
```

This instructs OBS to inject the repositories from your project into your appliance.

Additional repositories can be added by invoking `osc meta -e prj` and adding a line of the following form as a child of `<repository name="images">`:

```
<path project="$OBS_PROJECT" repository="$REPOSITORY_NAME" />
```

The order in which you add repositories matters: if a package is present in multiple repositories, then it is taken from the **first** repository. The **last** repository is subject to path expansion: its repository paths are included as well.

Don't forget to add the repository from the `Virtualization:Appliances:Builder` project, providing the latest stable version of KIWI NG (which you are very likely using for your local builds).

The following example repository configuration³ adds the repositories from the `Virtualization:Appliances:Builder` project and those from the latest snapshot of openSUSE Tumbleweed:

```
<project name="Virtualization:Appliances:Images:openSUSE-
→Tumbleweed">
  <title>JeOS for Tumbleweed </title>
  <description>Host JeOS images for Tumbleweed</description>
  <repository name="images">
    <path project="Virtualization:Appliances:Builder"
→repository="Factory"/>
    <path project="openSUSE:Factory" repository="snapshot"/>
    <arch>x86_64</arch>
  </repository>
</project>
```

The above can be simplified further using the path expansion of the last repository to:

```
<project name="Virtualization:Appliances:Images:openSUSE-
→Tumbleweed">
  <title>JeOS for Tumbleweed </title>
  <description>Host JeOS images for Tumbleweed</description>
  <repository name="images">
    <path project="Virtualization:Appliances:Builder"
→repository="Factory"/>
    <arch>x86_64</arch>
  </repository>
</project>
```

³ Taken from the project `Virtualization:Appliances:Images:openSUSE-Tumbleweed`

Now `Virtualization:Appliances:Builder` is the last repository, which' repositories are included into the search path. As `openSUSE:Factory/snapshot` is among these, it can be omitted from the repository list.

2. Keep the repositories in your `config.xml` configuration file. If you have installed the latest stable KIWI NG as described in [Installation](#) then you should add the following repository to your projects configuration (accessible via `osc meta -e prjconf`), so that OBS will pick the latest stable KIWI NG version too:

```
<repository name="images">
  <path project="Virtualization:Appliances:Builder" repository="
  ↪$DISTRO" />
  <arch>x86_64</arch>
</repository>
```

Replace `$DISTRO` with the appropriate name for the distribution that you are currently building and optionally adjust the architecture.

We recommend to use the first method, as it integrates better into OBS. Note that your image description will then no longer build outside of OBS though. If building locally is required, use the second method.

Warning: Adding the repositories to project's configuration makes it impossible to build images for different distributions from the same project.

Since the repositories are added for every package in your project, all your image builds will share the same repositories, thereby resulting in conflicts for different distributions.

We recommend to create a separate project for each distribution. If that is impossible, you can keep all your repositories (including `Virtualization:Appliances:Builder`) in `config.xml`. That however usually requires a large number of workarounds via `Prefer:` settings in the project configuration and is thus **not** recommended.

Project Configuration

The Open Build Service will by default create the same output file as KIWI NG when run locally, but with a custom filename ending (that is unfortunately unpredictable). This has the consequence that the download URL of your image will change with every rebuild (and thus break automated scripts). OBS can create symbolic links with static names to the latest build by adding the following line to the project configuration:

```
Reptype: staticlinks
```

If build Vagrant images (see [Image Description for Vagrant](#)) add the repository-type `vagrant`. OBS creates a `boxes/` subdirectory in your download repositories, which contains JSON files for Vagrant⁴.

⁴ Vagrant uses these JSON files for automatic updates of your Vagrant boxes.

If you have added your repositories to `config.xml`, you probably see errors of the following type:

```
unresolvable: have choice for SOMEPACKAGE: SOMEPACKAGE_1 SOMEPACKAGE_
↪2
```

Instead of starting from scratch and manually adding `Prefer:` statements to the project configuration, we recommend to copy the current project configuration of the testing project `Virtualization:Appliances:Images:Testing_${ARCH}:${DISTRO}` into your own project. It provides a good starting point and can be adapted to your OBS project.

6.10 Image Types

ISO Hybrid Live Image An iso image which can be dumped on a CD/DVD or USB stick and boots off from this media without interfering with other system storage components. A useful pocket system for testing and demo and debugging purposes.

Virtual Disk Image An image representing the system disk, useful for cloud frameworks like Amazon EC2, Google Compute Engine or Microsoft Azure.

OEM Expandable Disk Image An image representing an expandable system disk. This means after deployment the system can resize itself to the new disk geometry. The resize operation is configurable as part of the image description and an installation image for CD/DVD, USB stick and Network deployment can be created in addition.

PXE root File System Image A root filesystem image which can be deployed via KIWI NG's PXE netboot infrastructure. A client configuration file on the pxe server controls how the root filesystem image should be deployed. Many different deployment strategies are possible, e.g root over NBD, AoE or NFS for diskless and diskfull clients.

Docker Container Image An archive image suitable for the docker container engine. The image can be loaded via the `docker load` command and works within the scope of the container engine

6.11 Supported Distributions

KIWI NG can build images for the distributions which are **equal** or **newer** compared to the table below. For anything older use the legacy KIWI NG version `v7.x` For more details on the legacy KIWI NG, see: [Legacy KIWI vs. KIWI Next Generation \(KIWI NG\)](#)

The most compatible environment is provided if the build host is of the same distribution than the target image. This always applies for the Open Build Service (OBS). In other cases please check the table if your target combination is known to be supported.

Host / Image	CentOS 7	Fedora 30	openSUSE Leap 15	RHEL 7	SLE 12	SLE 15	openSUSE TW	Ubuntu 19
CentOS 7	yes	no	no	yes	no	no	no	no
Fedora 30	untested	yes	no	untested	no	no	no	no
openSUSE Leap 15	untested	note:dnf	yes	untested	no	yes	no	no
RHEL 7	untested	no	no	yes	no	no	no	no
SLE 12	no	untested	untested	no	yes	no	no	no
SLE 15	untested	note:dnf	yes	no	no	yes	untested	no
openSUSE TW	untested	note:dnf	yes	untested	no	untested	yes	no
Ubuntu 19	no	no	no	no	no	no	no	yes

dnf

dnf is the package manager used on Fedora and RHEL and is the successor of yum. When KIWI NG builds images for this distributions the latest version of dnf is required to be installed on the host to build the image.

In general, our goal is to support any major distribution with KIWI NG. However for building images we rely on core tools which are not under our control. Also several design aspects of distributions like **secure boot** and working with **upstream projects** are different and not influenced by us. There are many side effects that can be annoying especially if the build host is not of the same distribution vendor than the image target.

6.12 Supported Platforms and Architectures

Images built with KIWI NG are designed for a specific use case. The author of the image description sets this with the contents in the KIWI NG XML document as well as custom scripts and services. The following list provides an abstract of the platforms where KIWI NG built images are productively used:

- Amazon EC2
- Microsoft Azure
- Google Compute Engine
- Private Data Centers based on OpenStack
- Bare metal deployments e.g Microsoft Azure Large Instance
- SAP workloads

The majority of the workloads is based on the x86 architecture. KIWI NG also supports other architectures, shown in the table below:

Architecture	Supported
x86_64	yes
ix86	yes note:distro
s390/s390x	yes note:distro
arm/aarch64	yes note:distro
ppc64	no (alpha-phase)

distro

The support status for an architecture depends on the distribution. If the distribution does not build its packages for the desired architecture, KIWI NG will not be able to build an image for it

COMMAND LINE

Note: This document provides a list of the existing KIWI Next Generation (KIWI NG) commands for version 9.20.5.

7.1 kiwi-ng

7.1.1 SYNOPSIS

```
kiwi-ng [global options] service <command> [<args>]

kiwi-ng -h | --help
kiwi-ng [--profile=<name>...]
        [--type=<build_type>]
        [--logfile=<filename>]
        [--debug]
        [--color-output]
        image <command> [<args>...]
kiwi-ng [--debug]
        [--color-output]
        result <command> [<args>...]
kiwi-ng [--profile=<name>...]
        [--shared-cache-dir=<directory>]
        [--type=<build_type>]
        [--logfile=<filename>]
        [--debug]
        [--color-output]
        system <command> [<args>...]
kiwi-ng compat <legacy_args>...
kiwi-ng -v | --version
kiwi-ng help
```


7.1.2 DESCRIPTION

KIWI NG is an imaging solution that is based on an image XML description. Such a description is represented by a directory which includes at least one `config.xml` or `.kiwi` file and may as well include other files like scripts or configuration data.

A collection of example image descriptions can be found on the github repository here: <https://github.com/OSInside/kiwi-descriptions>. Most of the descriptions provide a so called JeOS image. JeOS means Just enough Operating System. 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.

KIWI NG operates in two steps. The system build command combines both steps into one to make it easier to start with KIWI NG. 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.

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.

KIWI NG supports the creation of the following image types:

- ISO Live Systems
- Virtual Disk for e.g cloud frameworks
- OEM Expandable Disk for system deployment from ISO or the network
- File system images for deployment in a pxe boot environment

Depending on the image type a variety of different disk formats and architectures are supported.

7.1.3 GLOBAL OPTIONS

- | | |
|---|---|
| --color-output | Use Escape Sequences to print different types of information in colored output. The underlying terminal has to understand those escape characters. Error messages appear red, warning messages yellow and debugging information will be printed light grey. |
| --debug | Print debug information on the commandline. |
| --logfile=<filename> | Specify log file. the logfile contains detailed information about the process. |
| --profile=<name> | Select profile to use. The specified profile must be part of the XML description. The option can be specified multiple times to allow using a combination of profiles |
| --shared-cache-dir=<directory> | Specify an alternative shared cache directory. The directory is shared via bind mount between the build host and image root system and contains information about package repositories and their cache and meta data. The default location is set to <code>/var/cache/kiwi</code> |

--type=<build_type> Select image build type. The specified build type must be configured as part of the XML description.

--version Show program version

7.1.4 EXAMPLE

```
$ git clone https://github.com/OSInside/kiwi-descriptions
$ kiwi --type vmx system build \
  --description kiwi-descriptions/suse/x86_64/suse-leap-15.1-JeOS_
→ \
  --target-dir /tmp/myimage
```

7.1.5 COMPATIBILITY

This version of KIWI NG uses a different caller syntax compared to former versions. However there is a compatibility mode which allows to use a legacy KIWI NG commandline as follows:

```
$ kiwi compat \
  --build kiwi-descriptions/suse/x86_64/suse-leap-15.1-JeOS \
  --type vmx -d /tmp/myimage
```

7.2 kiwi-ng result list

7.2.1 SYNOPSIS

```
kiwi-ng [global options] service <command> [<args>]

kiwi-ng result list -h | --help
kiwi-ng result list --target-dir=<directory>
kiwi-ng result list help
```

7.2.2 DESCRIPTION

List build results from a previous build or create command. Please note if you build an image several times with the same target directory the build result information will be overwritten each time you build the image. Therefore the build result list is valid for the last build

7.2.3 OPTIONS

--target-dir=<directory> directory containing the kiwi build results

7.3 kiwi-ng result bundle

7.3.1 SYNOPSIS

```
kiwi-ng [global options] service <command> [<args>]

kiwi-ng result bundle -h | --help
kiwi-ng result bundle --target-dir=<directory> --id=<bundle_id> --
  ↪ bundle-dir=<directory>
  [--zsync_source=<download_location>]
kiwi-ng result bundle help
```

7.3.2 DESCRIPTION

Create result bundle from the image build results in the specified target directory. Each result image will contain the specified bundle identifier as part of its filename. Uncompressed image files will also become xz compressed and a sha sum will be created from every result image.

7.3.3 OPTIONS

--bundle-dir=<directory> directory containing the bundle results, compressed versions of image results and their sha sums

--id=<bundle_id> bundle id, could be a free form text and is appended to the image version information if present as part of the result image filename

--target-dir=<directory> directory containing the kiwi build results

--zsync_source=<download_location> Specify the download location from which the bundle file(s) can be fetched from. The information is effective if `zsync` is used to sync the bundle.

- The zsync control file is only created for those bundle files which are marked for compression because in a KIWI NG build only those are meaningful for a partial binary file download.
- It is expected that all files from a bundle are placed to the same download location

7.4 kiwi-ng system prepare

7.4.1 SYNOPSIS

```
kiwi-ng [global options] service <command> [<args>]

kiwi-ng system prepare -h | --help
kiwi-ng system prepare --description=<directory> --root=<directory>
    [--allow-existing-root]
    [--clear-cache]
    [--ignore-repos]
    [--ignore-repos-used-for-build]
    [--set-repo=<source,type,alias,priority,imageinclude,package_
↪gpgcheck>]
    [--add-repo=<source,type,alias,priority,imageinclude,package_
↪gpgcheck>...]
    [--add-package=<name>...]
    [--add-bootstrap-package=<name>...]
    [--delete-package=<name>...]
    [--signing-key=<key-file>...]
kiwi-ng system prepare help
```

7.4.2 DESCRIPTION

Create a new image root directory. The prepare step builds a new image root directory from the specified XML description. The specified root directory is the root directory of the new image root system. As the root user you can enter this system via chroot as follows:

```
$ chroot <directory> bash
```

7.4.3 OPTIONS

- add-bootstrap-package=<name>** specify package to install as part of the early kiwi bootstrap phase. The option can be specified multiple times
- add-package=<name>** specify package to add(install). The option can be specified multiple times
- add-repo=<source,type,alias,priority,imageinclude,package_gpgcheck>**
See the kiwi::system::build manual page for further details
- allow-existing-root** allow to re-use an existing image root directory
- clear-cache** delete repository cache for each of the used repositories before installing any package. This is useful if an image build should take and validate the signature of the package from the original repository source for any build. Some package managers

unconditionally trust the contents of the cache, which is ok for cache data dedicated to one build but in case of kiwi the cache is shared between multiple image builds on that host for performance reasons.

- delete-package=<name>** specify package to delete. The option can be specified multiple times
- description=<directory>** Path to the kiwi XML description. Inside of that directory there must be at least a config.xml of *.kiwi XML description.
- ignore-repos** Ignore all repository configurations from the XML description. Using that option is usually done with a sequence of `--add-repo` options otherwise there are no repositories available for the image build which would lead to an error.
- ignore-repos-used-for-build** Works the same way as `--ignore-repos` except that repository configurations which has the `imageonly` attribute set to `true` will not be ignored.
- root=<directory>** Path to create the new root system.
- set-repo=<source,type,alias,priority,imageinclude,package_gpgcheck>** See the `kiwi::system::build` manual page for further details
- signing-key=<key-file>** set the key file to be trusted and imported into the package manager database before performing any operation. This is useful if an image build should take and validate repository and package signatures during build time. This option can be specified multiple times.

7.5 kiwi-ng system update

7.5.1 SYNOPSIS

```
kiwi-ng [global options] service <command> [<args>]

kiwi-ng system update -h | --help
kiwi-ng system update --root=<directory>
    [--add-package=<name>...]
    [--delete-package=<name>...]
kiwi-ng system update help
```


7.5.2 DESCRIPTION

Update a previously prepare image root tree. The update command refreshes the contents of the root directory with potentially new versions of the packages according to the repository setup of the image XML description. In addition the update command also allows to add or remove packages from the image root tree

7.5.3 OPTIONS

--add-package=<name> specify package to add(install). The option can be specified multiple times

--delete-package=<name> specify package to delete. The option can be specified multiple times

--root=<directory> Path to the root directory of the image.

7.6 kiwi-ng system build

7.6.1 SYNOPSIS

```
kiwi-ng [global options] service <command> [<args>]

kiwi-ng system build -h | --help
kiwi-ng system build --description=<directory> --target-dir=
↳<directory>
    [--allow-existing-root]
    [--clear-cache]
    [--ignore-repos]
    [--ignore-repos-used-for-build]
    [--set-repo=<source,type,alias,priority,imageinclude,package_
↳gpgcheck>]
    [--add-repo=<source,type,alias,priority,imageinclude,package_
↳gpgcheck>...]
    [--add-package=<name>...]
    [--add-bootstrap-package=<name>...]
    [--delete-package=<name>...]
    [--signing-key=<key-file>...]
kiwi-ng system build help
```


7.6.2 DESCRIPTION

build an image in one step. The build command combines kiwi's prepare and create steps in order to build an image with just one command call. The build command creates the root directory of the image below `<target-dir>/build/image-root` and if not specified differently writes a log file `<target-dir>/build/image-root.log`. The result image files are created in the specified target-dir.

7.6.3 OPTIONS

- add-bootstrap-package=<name>** specify package to install as part of the early kiwi bootstrap phase. The option can be specified multiple times
- add-package=<name>** specify package to add(install). The option can be specified multiple times
- add-repo=<source,type,alias,priority,imageinclude,package_gpgcheck>**
Add a new repository to the existing repository setup in the XML description. This option can be specified multiple times. For details about the provided option values see the **--set-repo** information below
- allow-existing-root** Allow to use an existing root directory from an earlier build attempt. Use with caution this could cause an inconsistent root tree if the existing contents does not fit to the former image type setup
- clear-cache** delete repository cache for each of the used repositories before installing any package. This is useful if an image build should take and validate the signature of the package from the original repository source for any build. Some package managers unconditionally trust the contents of the cache, which is ok for cache data dedicated to one build but in case of kiwi the cache is shared between multiple image builds on that host for performance reasons.
- delete-package=<name>** specify package to delete. The option can be specified multiple times
- description=<directory>** Path to the XML description. This is a directory containing at least one `_config.xml_` or `_*kiwi_` XML file.
- ignore-repos** Ignore all repository configurations from the XML description. Using that option is usually done with a sequence of `--add-repo` options otherwise there are no repositories available for the image build which would lead to an error.
- ignore-repos-used-for-build** Works the same way as `--ignore-repos` except that repository configurations which has the `imageonly` attribute set to `true` will not be ignored.

--set-repo=<source,type,alias,priority,imageinclude,package_gpgcheck>

Overwrite the first repository entry in the XML description with the provided information:

- **source**

source url, pointing to a package repository which must be in a format supported by the selected package manager. See the `URI_TYPES` section for details about the supported source locators.

- **type**

repository type, could be one of `rpm-md`, `rpm-dir` or `yast2`.

- **alias**

An alias name for the repository. If not specified kiwi calculates an alias name as result from a sha sum. The sha sum is used to uniquely identify the repository, but not very expressive. We recommend to set an expressive and unique alias name.

- **priority**

A number indicating the repository priority. How the value is evaluated depends on the selected package manager. Please refer to the package manager documentation for details about the supported priority ranges and their meaning.

- **imageinclude**

Set to either **true** or **false** to specify if this repository should be part of the system image repository setup or not.

- **package_gpgcheck**

Set to either **true** or **false** to specify if this repository should validate the package signatures.

--signing-key=<key-file> set the key file to be trusted and imported into the package manager database before performing any operation. This is useful if an image build should take and validate repository and package signatures during build time. This option can be specified multiple times

--target-dir=<directory> Path to store the build results.

7.6.4 URI_TYPES

- **http:// | https:// | ftp://**

remote repository delivered via http or ftp protocol.

- **obs://**

Open Buildservice repository. The source data is translated into an http url pointing to <http://download.opensuse.org>.

- **ibs://**

Internal Open Buildservice repository. The source data is translated into an http url pointing to download.suse.de.

- **iso://**

Local iso file. kiwi loop mounts the file and uses the mount point as temporary directory source type

- **dir://**

Local directory

7.7 kiwi-ng system create

7.7.1 SYNOPSIS

```
kiwi-ng [global options] service <command> [<args>]

kiwi-ng system create -h | --help
kiwi-ng system create --root=<directory> --target-dir=<directory>
    [--signing-key=<key-file>...]
kiwi-ng system create help
```

7.7.2 DESCRIPTION

Create an image from a previously prepared image root directory. The kiwi create call is usually issued after a kiwi prepare command and builds the requested image type in the specified target directory

7.7.3 OPTIONS

- root=<directory>** Path to the image root directory. This directory is usually created by the kiwi prepare command. If a directory is used which was not created by kiwi's prepare command, it's important to know that kiwi stores image build metadata below the image/ directory which needs to be present in order to let the create command operate correctly.
- target-dir=<directory>** Path to store the build results.
- signing-key=<key-file>** set the key file to be trusted and imported into the package manager database before performing any operation. This is useful if an image build should take and validate repository and package signatures during build time. In create step this option only affects the boot image. This option can be specified multiple times

7.8 kiwi-ng image resize

7.8.1 SYNOPSIS

```
kiwi-ng [global options] service <command> [<args>]

kiwi-ng image resize -h | --help
kiwi-ng image resize --target-dir=<directory> --size=<size>
    [--root=<directory>]
kiwi-ng image resize help
```

7.8.2 DESCRIPTION

For disk based images, allow to resize the image to a new disk geometry. The additional space is free and not in use by the image. In order to make use of the additional free space a repartition process is required like it is provided by kiwi's oem boot code. Therefore the resize operation is useful for oem image builds most of the time.

7.8.3 OPTIONS

- root=<directory>** The path to the root directory, if not specified kiwi searches the root directory in build/image-root below the specified target directory
- size=<size>** New size of the image. The value is either a size in bytes or can be specified with m=MB or g=GB. Example: 20g
- target-dir=<directory>** Directory containing the kiwi build results

7.9 kiwi-ng image info

7.9.1 SYNOPSIS

```
kiwi-ng [global options] service <command> [<args>]

kiwi-ng image info -h | --help
kiwi-ng image info --description=<directory>
    [--resolve-package-list]
    [--ignore-repos]
    [--add-repo=<source,type,alias,priority>...]
kiwi-ng image info help
```

7.9.2 DESCRIPTION

Provides information about the specified image description. If no specific info option is provided the command just lists basic information about the image which could also be directly obtained by reading the image XML description file. Specifying an extension option like `resolve-package-list` will cause a dependency resolver to run over the list of packages and thus provides more detailed information about the image description.

7.9.3 OPTIONS

- add-repo=<source,type,alias,priority>** Add repository with given source, type, alias and priority.
- description=<directory>** The description must be a directory containing a kiwi XML description and optional metadata files.
- ignore-repos** Ignore all repository configurations from the XML description. Using that option is usually done with a sequence of `--add-repo` options otherwise there are no repositories available for the processing the requested image information which could lead to an error.
- resolve-package-list** Solve package dependencies and return a list of all packages including their attributes e.g size, shasum, and more.

CONTRIBUTING

Note: Abstract

This document describes the development process of KIWI NG and how you can be part of it. This description applies to version 9.20.5.

8.1 Using KIWI NG in a Python Project

Note: Abstract

KIWI NG is provided as python module under the **kiwi** namespace. It is available for the python 3 version. The following description applies for KIWI NG version 9.20.5.

KIWI NG can also function as a module for other Python projects. The following example demonstrates how to read an existing image description, add a new repository definition and export the modified description on stdout.

```
import sys
import logging

from kiwi.xml_description import XMLDescription
from kiwi.xml_state import XMLState

description = XMLDescription('path/to/kiwi/XML/config.xml')

xml_data = description.load()

xml_state = XMLState(
    xml_data=xml_data, profiles=[], build_type='iso'
)

xml_state.add_repository(
    repo_source='http://repo',
```

(continues on next page)

(continued from previous page)

```
repo_type='rpm-md',
repo_alias='myrepo',
repo_prio=99
)

xml_data.export(
    outfile=sys.stdout, level=0
)
```

All classes are written in a way to care for a single responsibility in order to allow for re-use on other use cases. Therefore it is possible to use KIWI NG outside of the main image building scope to manage e.g the setup of loop devices, filesystems, partitions, etc. . .

8.2 Plugin Architecture

Each command provided by KIWI NG is written as a task plugin under the **kiwi.tasks** namespace. As a developer you can extend KIWI NG with custom task plugins if the following conventions are taken into account:

8.2.1 Naming Conventions

Task Plugin File Name The file name of a task plugin must follow the pattern `<service>_<command>.py`. This allows to invoke the task with **kiwi-ng service command ...**

Task Plugin Option Handling KIWI NG uses the docopt module to handle options. Each task plugin must use docopt to allow option handling.

Task Plugin Class The implementation of the plugin must be a class that matches the naming convention: `<Service><Command>Task`. The class must inherit from the `CliTask` base class. On startup of the plugin, KIWI NG expects an implementation of the `process` method.

Task Plugin Entry Point Registration of the plugin must be done in `setup.py` using the `entry_points` concept from Python's `setuptools`.

```
'entry_points': {
    'kiwi.tasks': [
        'service_command=kiwi.tasks.service_command'
    ]
}
```


8.2.2 Example Plugin

Note: The following example assumes an existing Python project which was set up according to the Python project rules and standards.

1. Create the task plugin directory `kiwi/tasks`
2. Create the entry point in `setup.py`.

Assuming we want to create the service named **relax** providing the command **justdoit** this would be the following entry point definition in `setup.py`:

```
'entry_points': {
    'kiwi.tasks': [
        'relax_justdoit=kiwi.tasks.relax_justdoit'
    ]
}
```

3. Create the plugin code in the file `kiwi/tasks/relax_justdoit.py` with the following content:

```
"""
usage: kiwi-ng relax justdoit -h | --help
       kiwi-ng relax justdoit --now

commands:
    justdoit
        time to relax

options:
    --now
        right now. For more details about docopt
        see: http://docopt.org

"""
# These imports requires kiwi to be part of your environment
# It can be either installed from pip into a virtual development
# environment or from the distribution package manager
from kiwi.tasks.base import CliTask
from kiwi.help import Help

class RelaxJustDoItTask(CliTask):
    def process(self):
        self.manual = Help()
        if self.command_args.get('help') is True:
            # The following will invoke man to show the man page
            # for the requested command. Thus for the call to
            # succeed a manual page needs to be written and
            # installed by the plugin
            return self.manual.show('kiwi::relax::justdoit')
```

(continues on next page)

(continued from previous page)

```
print(  
    'https://genius.com/Frankie-goes-to-hollywood-relax-  
→lyrics'  
)
```

4. Test the plugin

```
$ ./setup.py develop  
$ kiwi-ng relax justdoit --now
```

8.3 Extending KIWI NG with Custom Operations

Note: Abstract

Users building images with KIWI NG need to implement their own infrastructure if the image description does not provide a way to embed custom information which is outside of the scope of the general schema as it is provided by KIWI NG today.

This document describes how to create an extension plugin for the KIWI NG schema to add and validate additional information in the KIWI NG image description.

Such a schema extension can be used in an additional KIWI NG task plugin to provide a new subcommand for KIWI NG. As of today there is no other plugin interface except for providing additional KIWI NG commands implemented.

Depending on the demand for custom plugins, the interface to hook in code into other parts of the KIWI NG processing needs to be extended.

This description applies for version 9.20.5.

8.3.1 The <extension> Section

The main KIWI NG schema supports an extension section which allows to specify any XML structure and attributes as long as they are connected to a namespace. According to this any custom XML structure can be implemented like the following example shows:

```
<image>  
  ...  
  <extension xmlns:my_plugin="http://www.my_plugin.com">  
    <my_plugin:my_feature>  
      <my_plugin:title name="cool stuff"/>  
    </my_plugin:my_feature>  
  </extension>  
</image>
```


- Any toplevel namespace must exist only once
- Multiple different toplevel namespaces are allowed, e.g my_plugin_a, my_plugin_b

8.3.2 RELAX NG Schema for the Extension

If an extension section is found, KIWI NG looks up its namespace and asks the main XML catalog for the schema file to validate the extension data. The schema file must be a RELAX NG schema in the .rng format. We recommend to place the schema as /usr/share/xml/kiwi/my_plugin.rng

For the above example the RELAX NG Schema in the compressed format my_plugin.rnc would look like this:

```
namespace my_plugin = "http://www.my_plugin.com"

start =
    k.my_feature

div {
    k.my_feature.attlist = empty
    k.my_feature =
        element my_plugin:my_feature {
            k.my_feature.attlist &
            k.title
        }
}

div {
    k.title.name.attribute =
        attribute name { text }
    k.title.attlist = k.title.name.attribute
    k.title =
        element my_plugin:title {
            k.title.attlist
        }
}
```

In order to convert this schema to the .rng format just call:

```
$ trang -I rnc -O rng my_plugin.rnc /usr/share/xml/kiwi/my_plugin.
→rng
```


8.3.3 Extension Schema in XML catalog

As mentioned above the mapping from the extension namespace to the correct RELAX NG schema file is handled by a XML catalog file. The XML catalog for the example use here looks like this:

```
<?xml version="1.0"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <system
    systemId="http://www.my_plugin.com"
    uri="file:///usr/share/xml/kiwi/my_plugin.rng"/>
</catalog>
```

For resolving the catalog KIWI NG uses the **xmlcatalog** command and the main XML catalog from the system which is `/etc/xml/catalog`.

Note: It depends on the distribution and its version how the main catalog gets informed about the existence of the KIWI NG extension catalog file. Please consult the distribution manual about adding XML catalogs.

If the following command provides the information to the correct RELAX NG schema file you are ready for a first test:

```
$ xmlcatalog /etc/xml/catalog http://www.my_plugin.com
```

8.3.4 Using the Extension

In order to test your extension place the example extension section from the beginning of this document into one of your image description's `config.xml` file

The following example will read the name attribute from the title section of the `my_feature` root element and prints it:

```
import logging

from kiwi.xml_description import XMLDescription

description = XMLDescription('path/to/kiwi/XML/config.xml')
description.load()

my_plugin = description.get_extension_xml_data('my_plugin')

print(my_plugin.getroot()[0].get('name'))
```


8.4 The Basics

The core appliance builder is developed in Python and follows the test driven development rules.

If you want to implement a bigger feature, consider opening an issue on GitHub first to discuss the changes. Or join the discussion in the #kiwi channel on [Riot.im](https://riot.im).

8.5 Fork the upstream repository

1. On GitHub, navigate to: <https://github.com/OSInside/kiwi>
2. In the top-right corner of the page, click **Fork**.

8.6 Create a local clone of the forked repository

```
$ git clone https://github.com/YOUR-USERNAME/kiwi
$ git remote add upstream https://github.com/OSInside/kiwi.git
```

8.7 Install Required Operating System Packages

KIWI NG requires the following additional packages which are not provided by **pip**:

XML processing libraries `libxml2` and `libxslt` (for `lxml`)

Python header files, GCC compiler and glibc-devel header files Required for python modules that hooks into shared library context

Spell Checking library Provided by the `enchant` library

ShellCheck [ShellCheck](https://github.com/koalaman/shellcheck) script linter.

ISO creation program One of `xorriso` (preferred) or `genisoimage`.

LaTeX documentation build environment A full LaTeX installation is required to build the PDF documentation¹.

The above mentioned system packages will be installed by calling the `install_devel_packages.sh` helper script from the checked out Git repository as follows:

¹ Sphinx requires a plethora of additional LaTeX packages. Unfortunately there is currently no comprehensive list available. On Ubuntu/Debian installing `texlive-latex-extra` should be sufficient. For Fedora, consult the package list from `.gitlab-ci.yml`.


```
$ sudo helper/install_devel_packages.sh
```

Note: The helper script checks for the package managers `zypper` and `dnf` and associates a distribution with it. If you use a distribution that does not use one of those package managers the script will not install any packages and exit with an error message. In this case we recommend to take a look at the package list encoded in the script and adapt to your distribution and package manager as needed.

8.8 Create a Python Virtual Development Environment

The following commands initializes and activates a development environment for Python 3:

```
$ tox -e devel
$ source .tox/3/bin/activate
```

The commands above automatically creates the application script called **kiwi-ng**, which allows you to run KIWI NG from the Python sources inside the virtual environment:

```
$ kiwi-ng --help
```

Warning: The virtualenv's `$PATH` will not be taken into account when calling KIWI NG via **sudo**! Use the absolute path to the KIWI NG executable to run an actual build using your local changes:

```
$ sudo $PWD/.tox/3/bin/kiwi-ng system build ...
```

To leave the development mode, run:

```
$ deactivate
```

To resume your work, **cd** into your local Git repository and call:

```
$ source .tox/3/bin/activate
```

8.9 Running the Unit Tests

We use **tox** to run the unit tests. Tox sets up its own virtualenvs inside the `.tox` directory for multiple Python versions and should thus **not** be invoked from inside your development virtualenv.

Before submitting your changes via a pull request, ensure that all tests pass and that the code has the required test coverage via the command:


```
$ tox
```

We also include `pytest-xdist` in the development `virtualenv` which allows to run the unit tests in parallel. It is turned off by default but can be enabled via:

```
$ tox "-n NUMBER_OF_PROCESSES"
```

where you can insert an arbitrary number as `NUMBER_OF_PROCESSES` (or a shell command like `$(nproc)`). Note that the double quotes around `-n NUMBER_OF_PROCESSES` are required (otherwise `tox` will consume this command line flag instead of forwarding it to `pytest`).

The previous call would run the unit tests for different Python versions, check the source code for errors and build the documentation.

If you want to see the available targets, use the option `-l` to let `tox` print a list of them:

```
$ tox -l
```

To only run a special target, use the `-e` option. The following example runs the test cases for the Python 3.6 interpreter only:

```
$ tox -e unit_py3_6
```

8.10 Create a Branch for each Feature or Bugfix

Code changes should be done in an extra Git branch. This allows for creating GitHub pull requests in a clean way. See also: [Collaborating with issues and pull requests](#)

```
$ git checkout -b my-topic-branch
```

Make and commit your changes.

Note: You can make multiple commits which is generally useful to give your changes a clear structure and to allow us to better review your work.

Note: Your work is important and must be signed to ensure the integrity of the repository and the code. Thus we recommend to setup a signing key as documented in [Signing Git Patches](#).

```
$ git commit -S -a
```

Run the tests and code style checks. All of these are also performed by the [Travis CI](#) and [GitLab CI](#) integration test systems when a pull request is created.


```
$ tox
```

Once everything is done, push your local branch to your forked repository and create a pull request into the upstream repository.

```
$ git push origin my-topic-branch
```

Thank you much for contributing to KIWI NG. Your time and work effort is very much appreciated!

8.11 Coding Style

KIWI NG follows the general PEP8 guidelines with the following exceptions:

- We do not use free functions at all. Even utility functions must be part of a class, but should be either prefixed with the `@classmethod` or `@staticmethod` decorators (whichever is more appropriate).
- Do not set module and class level variables, put these into the classes' `__init__` method.
- The names of constants are not written in all capital letters.

8.11.1 Documentation

KIWI NG uses [Sphinx](#) for the API and user documentation.

In order to build the HTML documentation call:

```
tox -e doc
```

or to build the full documentation (including a PDF generated by LaTeX³):

```
tox -e packagedoc
```

Document all your classes, methods, their parameters and their types using the standard [reStructuredText](#) syntax as supported by Sphinx, an example class is documented as follows:

```
class Example:
    """
    **Example class**

    :param str param: A parameter
    :param bool : Source file name to compress
    :param list supported_zipper: List of supported compression_
    ↪tools
```

(continues on next page)

³ Requires a full LaTeX installation.

(continued from previous page)

```
:attr Optional[str] attr: A class attribute
"""
def __init__(self, param, param_w_default=False):
    self.attr = param if param_w_default else None

def method(self, param):
    """
    A method that takes a parameter.

    :param list param: a parameter
    :return: whether param is very long
    :rtype: bool
    """
    return len(param) > 50
```

Try to stick to the following guidelines when documenting source code:

- Classes should be documented directly in their main docstring and not in `__init__`.
- Document **every** function parameter and every public attribute including their types.
- Only public methods should be documented, private methods don't have to, unless they are complex and it is not easy to grasp what they do (which should be avoided anyway).

Please also document any user-facing changes that you implementing (e.g. adding a new build type) in the user documentation, which can be found in `doc/source`. General documentation should be put into the `working_with_kiwi/` subfolder, whereas documentation about more specialized topics would belong into the `building/` subfolder.

Adhere to a line limit of 75 characters when writing the user facing documentation².

² Configure your editor to automatically break lines and/or reformat paragraphs. For Emacs you can use `M-x set-fill-column RET 75` and `M-x auto-fill-mode RET` for automatic filling of paragraphs in conjunction with `M-x fill-paragraph` (usually bound to `M-q`) to reformat a paragraph to adhere to the current column width. For editing reStructuredText we recommend `rst-mode` (built-in to Emacs since version 23.1). Vim users can set the text width via `:tw 75` and then use the commands `gwip` or `gg`.

8.12 Additional Information

The following sections provides further information about the repository integrity, version, package and documentation management.

8.12.1 Signing Git Patches

To ensure the integrity of the repository and the code base, patches sent for inclusion should be signed with a GPG key.

To prepare Git to sign commits, follow these instructions:

1. Create a key suitable for signing (it is not recommended to use existing keys to not mix it with your email environment):

```
$ gpg2 --expert --full-gen-key
```

2. Either choose a RSA key for signing (option (4)) or an ECC key for signing (option (10)). For a RSA key choose a key size of 4096 bits and for a ECC key choose Curve 25519 (option (1)). Enter a reasonable validity period (we recommend 2 to 5 years). Complete the key generation by entering your name and email address.
3. Add the key ID to your git configuration, by running the following **git config** commands:

```
$ git config --local user.signingkey $YOUR_SIGN_KEY_ID
$ git config --local commit.gpgSign true
```

Omitting the flag `--local` will make these settings global for all repositories (they will be added to `~/.gitconfig`). You can find your signkey's ID via:

```
$ gpg2 --list-keys --keyid-format long $YOUR_EMAIL
pub  rsa4096/AABBCCDDEEFF0011 2019-04-26 [S] [expires: 2021-04-
→16]
AAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBB
uid  [ultimate] YOU <$YOUR_EMAIL>
```

The key's ID in this case would be `AABBCCDDEEFF0011`. Note that your signkey will have only a `[S]` after the creation date, not a `[SC]` (then you are looking at your ordinary GPG key that can also encrypt).

8.12.2 Bumping the Version

The KIWI NG project follows the [Semantic Versioning](#) scheme. We use the **bumpversion** tool for consistent versioning.

Follow these instructions to bump the major, minor, or patch part of the KIWI NG version. Ensure that your repository is clean (i.e. no modified and unknown files exist) beforehand running **bumpversion**.

- For backwards-compatible bug fixes:

```
$ bumpversion patch
```

- For additional functionality in a backwards-compatible manner. When changed, the patch level is reset to zero:

```
$ bumpversion minor
```

- For incompatible API changes. When changed, the patch and minor levels are reset to zero:

```
$ bumpversion major
```

8.12.3 Creating a RPM Package

We provide a template for a RPM spec file in `package/python-kiwi-spec-template` alongside with a `rpmlint` configuration file and an automatically updated `python-kiwi.changes`.

To create the necessary files to build a RPM package via `rpmbuild`, run:

```
$ make build
```

The sources are collected in the `dist/` directory. These can be directly build it with **rpmbuild**, **fedpkg**, or submitted to the Open Build Service using **osc**.

IMAGE DESCRIPTION XML SCHEMA

Note: This API documentation covers KIWI NG 9.20.5

10.1 kiwi Package

10.1.1 Submodules

10.1.2 `kiwi.app` Module

class `kiwi.app.App`

Bases: `object`

Implements creation of task instances

Each task class implements a process method which is called when constructing an instance of App

10.1.3 `kiwi.cli` Module

class `kiwi.cli.Cli`

Bases: `object`

Implements the main command line interface

An instance of the Cli class builds the entry point for the application and implements methods to load further command plugins which itself provides their own command line interface

get_command()

Extract selected command name

Returns command name

Return type `str`

get_command_args()

Extract argument dict for selected command

Returns

Contains dictionary of command arguments

```
{
    '--command-option': 'value'
}
```

Return type `dict`**get_global_args()**

Extract argument dict for global arguments

Returns

Contains dictionary of global arguments

```
{
    '--global-option': 'value'
}
```

Return type `dict`**get_servicename()**

Extract service name from argument parse result

Returns service name**Return type** `str`**invoke_kiwicompat(*compat_args*)**

Execute kiwicompat with provided legacy KIWI command line arguments

Example:

```
invoke_kiwicompat(
    '--build', 'description', '--type', 'vmx',
    '-d', 'destination'
)
```

Parameters **compat_args** (*list*) – legacy kiwi command arguments**load_command()**

Loads task class plugin according to service and command name

Returns loaded task module**Return type** `object`**show_and_exit_on_help_request()**

Execute man to show the selected manual page

10.1.4 kiwi.command Module

class `kiwi.command.Command`

Bases: `object`

Implements command invocation

An instance of `Command` provides methods to invoke external commands in blocking and non blocking mode. Control of stdout and stderr is given to the caller

static call (*command*, *custom_env=None*)

Execute a program and return an io file handle pair back. stdout and stderr are both on different channels. The caller must read from the output file handles in order to actually run the command. This can be done using the `CommandIterator` from `command_process`

Example:

```
process = Command.call(['ls', '-l'])
```

Parameters

- **command** (*list*) – command and arguments
- **custom_env** (*list*) – custom os.environ

Returns

Contains process results in command type

```
command(  
    output='string', output_available=bool,  
    error='string', error_available=bool,  
    process=subprocess  
)
```

Return type `namedtuple`

static run (*command*, *custom_env=None*, *raise_on_error=True*,
 stderr_to_stdout=False)

Execute a program and block the caller. The return value is a hash containing the stdout, stderr and return code information. Unless `raise_on_error` is set to false an exception is thrown if the command exits with an error code not equal to zero

Example:

```
result = Command.run(['ls', '-l'])
```

Parameters

- **command** (*list*) – command and arguments
- **custom_env** (*list*) – custom os.environ

- **raise_on_error** (*bool*) – control error behaviour
- **stderr_to_stdout** (*bool*) – redirects stderr to stdout

Returns

Contains call results in command type

```
command(output='string', error='string', ↵  
↵returncode=int)
```

Return type namedtuple

`kiwi.command.command_type`
alias of `kiwi.command.command`

10.1.5 kiwi.command_process Module

class `kiwi.command_process.CommandIterator` (*command*)
Bases: `object`

Implements an Iterator for Instances of Command

Parameters **command** (*subprocess*) – instance of subprocess

get_error_code ()

Provide return value from processed command

Returns errorcode

Return type `int`

get_error_output ()

Provide data which was sent to the stderr channel

Returns stderr data

Return type `str`

get_pid ()

Provide process ID of command while running

Returns pid

Return type `int`

kill ()

Send kill signal SIGTERM to command process

class `kiwi.command_process.CommandProcess` (*command*,
log_topic='system')
Bases: `object`

Implements processing of non blocking Command calls

Provides methods to iterate over non blocking instances of the Command class with and without progress information

Parameters

- **command** (*subprocess*) – instance of subprocess
- **log_topic** (*string*) – topic string for logging

create_match_method (*method*)

create a matcher function pointer which calls the given method as method(item_to_match, data) on dereference

Parameters **method** (*function*) – function reference**Returns** function pointer**Return type** *object***poll** ()

Iterate over process, raise on error and log output

poll_and_watch ()

Iterate over process don't raise on error and log stdout and stderr

poll_show_progress (*items_to_complete*, *match_method*)

Iterate over process and show progress in percent raise on error and log output

Parameters

- **items_to_complete** (*list*) – all items
- **match_method** (*function*) – method matching item

returncode ()

10.1.6 kiwi.defaults Module

class `kiwi.defaults.Defaults`

Bases: *object*

Implements default values

Provides static methods for default values and state information

get (*key*)

Implements get method for profile elements

Parameters **key** (*string*) – profile keyname**Returns** key value**Return type** *str***static** **get_archive_image_types** ()

Provides list of supported archive image types

Returns archive names**Return type** *list*

static get_bios_image_name()

Provides bios core boot binary name

Returns name

Return type str

static get_bios_module_directory_name()

Provides x86 BIOS directory name which stores the pc binaries

Returns directory name

Return type str

static get_boot_image_description_path()

Provides the path to find custom kiwi boot descriptions

Returns directory path

Return type str

static get_boot_image_strip_file()

Provides the file path to bootloader strip metadata. This file contains information about the files and directories automatically striped out from the kiwi initrd

Returns file path

Return type str

static get_buildservice_env_name()

Provides the base name of the environment file in a buildservice worker

Returns file basename

Return type str

static get_common_functions_file()

Provides the file path to config functions metadata.

This file contains bash functions used for system configuration or in the boot code from the kiwi initrd

Returns file path

Return type str

static get_container_base_image_tag()

Provides the tag used to identify base layers during the build of derived images.

Returns tag

Return type str

static get_container_compression()

Provides default container compression algorithm

Returns name

Return type str

static get_container_image_types()

Provides list of supported container image types

Returns container names

Return type `list`

static get_custom_rpm_bootstrap_macro_name()

Returns the rpm bootstrap macro file name created in the custom rpm macros path

Returns filename

Return type `str`

static get_custom_rpm_image_macro_name()

Returns the rpm image macro file name created in the custom rpm macros path

Returns filename

Return type `str`

static get_custom_rpm_macros_path()

Returns the custom macros directory for the rpm database.

Returns path name

Return type `str`

static get_default_boot_mbytes()

Provides default boot partition size in mbytes

Returns mbsize value

Return type `int`

static get_default_boot_timeout_seconds()

Provides default boot timeout in seconds

Returns seconds

Return type `int`

static get_default_container_created_by()

Provides the default ‘created by’ history entry for containers.

Returns the specific kiwi version used for the build

Return type `str`

static get_default_container_name()

Provides the default container name.

Returns name

Return type `str`

static get_default_container_subcommand()

Provides the default container subcommand.

Returns command as a list of arguments

Return type `list`

static `get_default_container_tag()`

Provides the default container tag.

Returns `tag`

Return type `str`

static `get_default_disk_start_sector()`

Provides the default initial disk sector for the first disk partition.

Returns `sector value`

Return type `int`

static `get_default_efi_boot_mbytes()`

Provides default EFI partition size in mbytes

Returns `mbsize value`

Return type `int`

static `get_default_efi_partition_table_type()`

Provides the default partition table type for efi firmwares.

Returns `partition table type name`

Return type `str`

static `get_default_firmware(arch)`

Provides default firmware for specified architecture

Parameters `arch` (*string*) – platform.machine

Returns `firmware name`

Return type `str`

static `get_default_inode_size()`

Provides default size of inodes in bytes. This is only relevant for inode based filesystems

Returns `bytesize value`

Return type `int`

static `get_default_legacy_bios_mbytes()`

Provides default size of bios_grub partition in mbytes

Returns `mbsize value`

Return type `int`

static `get_default_live_iso_root_filesystem()`

Provides default live iso root filesystem type

Returns `filesystem name`

Return type `str`

static get_default_live_iso_type()

Provides default live iso union type

Returns live iso type

Return type `str`

static get_default_packager_tool(*package_manager*)

Provides the packager tool according to the package manager

Parameters **package_manager** (*string*) – package manger name

Returns packager tool binary name

Return type `str`

static get_default_prep_mbytes()

Provides default size of prep partition in mbytes

Returns mbsize value

Return type `int`

static get_default_uri_type()

Provides default URI type

Absolute path specifications used in the context of an URI will apply the specified default mime type

Returns URI mime type

Return type `str`

static get_default_video_mode()

Provides 800x600 default video mode as hex value for the kernel

Returns vesa video kernel hex value

Return type `str`

static get_default_volume_group_name()

Provides default LVM volume group name

Returns name

Return type `str`

static get_disk_format_types()

Provides supported disk format types

Returns disk types

Return type `list`

static get_disk_image_types()

Provides supported disk image types

Returns disk image type names

Return type `list`

static get_dracut_conf_name()

Provides file path of dracut config file to be used with KIWI

Returns file path name

Return type `str`

static get_ec2_capable_firmware_names()

Provides list of EC2 capable firmware names. These are those for which kiwi supports the creation of disk images bootable within the Amazon EC2 public cloud

Returns firmware names

Return type `list`

static get_efi_capable_firmware_names()

Provides list of EFI capable firmware names. These are those for which kiwi supports the creation of an EFI bootable disk image

Returns firmware names

Return type `list`

static get_efi_image_name(*arch*)

Provides architecture specific EFI boot binary name

Parameters **arch** (*string*) – platform.machine

Returns name

Return type `str`

static get_efi_module_directory_name(*arch*)

Provides architecture specific EFI directory name which stores the EFI binaries for the desired architecture.

Parameters **arch** (*string*) – platform.machine

Returns directory name

Return type `str`

get_exclude_list_for_non_physical_devices()

Provides the list of folders that are not associated with a physical device. KIWI returns the basename of the folders typically used as mountpoint for those devices.

Returns list of file and directory names

Return type `list`

static get_exclude_list_for_root_data_sync()

Provides the list of files or folders that are created by KIWI for its own purposes. Those files should be not be included in the resulting image.

Returns list of file and directory names

Return type `list`

static get_failsafe_kernel_options()

Provides failsafe boot kernel options

Returns

list of kernel options

```
['option=value', 'option']
```

Return type `list`

static get_filesystem_image_types()

Provides list of supported filesystem image types

Returns filesystem names

Return type `list`

static get_firmware_types()

Provides supported architecture specific firmware types

Returns firmware types per architecture

Return type `dict`

static get_grub_basic_modules(*multiboot*)

Provides list of basic grub modules

Parameters **multiboot** (*bool*) – grub multiboot mode

Returns list of module names

Return type `list`

static get_grub_bios_core_loader(*root_path*)

Provides grub bios image

Searches distribution specific locations to find the core bios image below the given root path

Parameters **root_path** (*string*) – image root path

Returns file path or None

Return type `str`

static get_grub_bios_modules(*multiboot=False*)

Provides list of grub bios modules

Parameters **multiboot** (*bool*) – grub multiboot mode

Returns list of module names

Return type `list`

static get_grub_boot_directory_name(*lookup_path*)

Provides grub2 data directory name in boot/ directory

Depending on the distribution the grub2 boot path could be either boot/grub2 or boot/grub. The method will decide for the correct base directory name according to the name pattern of the installed grub2 tools

Returns directory basename

Return type `str`

static `get_grub_efi_font_directory (root_path)`

Provides distribution specific EFI font directory used with grub.

Parameters `root_path (string)` – image root path

Returns file path or None

Return type `str`

static `get_grub_efi_modules (multiboot=False)`

Provides list of grub efi modules

Parameters `multiboot (bool)` – grub multiboot mode

Returns list of module names

Return type `list`

static `get_grub_ofw_modules ()`

Provides list of grub ofw modules (ppc)

Returns list of module names

Return type `list`

static `get_grub_path (root_path, filename, raise_on_error=True)`

Provides grub path to given search file

Depending on the distribution grub could be installed below a grub2 or grub directory. grub could also reside in /usr/lib as well as in /usr/share. Therefore this information needs to be dynamically looked up

Parameters

- **root_path (string)** – root path to start the lookup from
- **filename (string)** – filename to search
- **raise_on_error (bool)** – raise on not found, defaults to True

The method returns the path to the given grub search file. By default it raises a `KiwiBootLoaderGrubDataError` exception if the file could not be found in any of the search locations. If `raise_on_error` is set to False and no file could be found the function returns None

Returns filepath

Return type `str`

static `get_imported_root_image (root_dir)`

Provides the path to an imported root system image

If the image description specified a `derived_from` attribute the file from this attribute is copied into the `root_dir` using the name as provided by this method

Parameters `root_dir` (*string*) – image root directory

Returns file path name

Return type *str*

static `get_install_volume_id()`

Provides default value for ISO volume ID for install media

Returns name

Return type *str*

static `get_iso_boot_path()`

Provides arch specific relative path to boot files on kiwi iso filesystems

Returns relative path name

Return type *str*

static `get_iso_tool_category()`

Provides default iso tool category

Returns name

Return type *str*

static `get_isolinux_bios_grub_loader()`

Return name of eltorito grub image used as isolinux loader in BIOS mode if isolinux.bin should not be used

Returns file base name

Return type *str*

static `get_live_dracut_module_from_flag(flag_name)`

Provides `flag_name` to dracut module name map

Depending on the value of the `flag` attribute in the KIWI image description a specific dracut module needs to be selected

Returns dracut module name

Return type *str*

static `get_live_image_types()`

Provides supported live image types

Returns live image type names

Return type *list*

static `get_live_iso_persistent_boot_options(persistent_filesystem=None)`

Provides list of boot options passed to the dracut kiwi-live module to setup persistent writing

Returns list of boot options

Return type `list`

static `get_luks_key_length()`

Provides key length to use for random luks keys

static `get_lvm_overhead_mbytes()`

Provides empiric LVM overhead size in mbytes

Returns mbsize value

Return type `int`

static `get_min_partition_mbytes()`

Provides default minimum partition size in mbytes

Returns mbsize value

Return type `int`

static `get_min_volume_mbytes()`

Provides default minimum LVM volume size in mbytes

Returns mbsize value

Return type `int`

static `get_network_image_types()`

Provides supported pxe image types

Returns pxe image type names

Return type `list`

static `get_obs_download_server_url()`

Provides the default download server url hosting the public open builds service repositories

Returns url path

Return type `str`

static `get_oci_archive_tool()`

Provides the default OCI archive tool name.

Returns name

Return type `str`

static `get_preparer()`

Provides ISO preparer name

Returns name

Return type `str`

`get_profile_file()`

Return name of profile file for given root directory

static `get_publisher()`

Provides ISO publisher name

Returns name

Return type `str`

static `get_recovery_spare_mbytes()`

Provides spare size of recovery partition in mbytes

Returns mbsize value

Return type `int`

static `get_s390_disk_block_size()`

Provides the default block size for s390 storage disks

Returns blocksize value

Return type `int`

static `get_s390_disk_type()`

Provides the default disk type for s390 storage disks

Returns type name

Return type `str`

static `get_schema_file()`

Provides file path to kiwi RelaxNG schema

Returns file path

Return type `str`

static `get_shared_cache_location()`

Provides the shared cache location

This is a directory which shares data from the image buildsystem host with the image root system. The location is returned as an absolute path stripped off by the leading `'/'`. This is because the path is transparently used on the host `/<cache-dir>` and inside of the image `imageroot/<cache-dir>`

Returns directory path

Return type `str`

static `get_shim_loader(root_path)`

Provides shim loader file path

Searches distribution specific locations to find shim.efi below the given root path

Parameters `root_path` (*string*) – image root path

Returns file path or None

Return type `str`

static `get_shim_vendor_directory(root_path)`

Provides shim vendor directory

Searches distribution specific locations to find shim.efi below the given root path and return the directory name to the file found

Parameters `root_path` (*string*) – image root path

Returns directory path or None

Return type `str`

static `get_signed_grub_loader` (*root_path*)

Provides shim signed grub loader file path

Searches distribution specific locations to find grub.efi below the given root path

Parameters `root_path` (*string*) – image root path

Returns file path or None

Return type `str`

static `get_snapper_config_template_file` ()

Provides the default configuration template file for snapper

Returns file

Return type `str`

static `get_solvable_location` ()

Provides the directory to store SAT solvables for repositories. The solvable files are used to perform package dependency and metadata resolution

Returns directory path

Return type `str`

static `get_swapspace_mbytes` ()

Provides swapsize in MB

static `get_syslinux_modules` ()

Returns list of syslinux modules to include on ISO images that boots via isolinux

Returns base file names

Return type `list`

static `get_syslinux_search_paths` ()

syslinux is packaged differently between distributions. This method returns a list of directories to search for syslinux data

Returns directory names

Return type `list`

static `get_unsigned_grub_loader` (*root_path*)

Provides unsigned grub efi loader file path

Searches distribution specific locations to find grub.efi below the given root path

Parameters `root_path` (*string*) – image root path

Returns file path or None

Return type `str`

static get_vagrant_config_virtualbox_guest_additions()

Provides the default value for vagrantconfig.virtualbox_guest_additions_present

Returns whether guest additions are expected to be present in the vagrant box

Return type `bool`

static get_video_mode_map()

Provides video mode map

Assign a tuple to each kernel vesa hex id for each of the supported bootloaders

Returns

video type map

```
{ 'kernel_hex_mode': video_type(grub2='mode', ↵  
↵isolinux='mode') }
```

Return type `dict`

static get_volume_id()

Provides default value for ISO volume ID

Returns name

Return type `str`

static get_xsl_stylesheet_file()

Provides the file path to the KIWI XSLT style sheets

Returns file path

Return type `str`

static get_xz_compression_options()

Provides compression options for the xz compressor

Returns

Contains list of options

```
[ '--option=value' ]
```

Return type `list`

static is_buildservice_worker()

Checks if build host is an open buildservice machine

The presence of /.buildenv on the build host indicates we are building inside of the open buildservice

Returns True if obs worker, else False

Return type `bool`

static `is_x86_arch`(*arch*)

Checks if machine architecture is x86 based

Any arch that matches 32bit and 64bit x86 architecture causes the method to return True. Anything else will cause the method to return False

Return type `bool`

static `project_file`(*filename*)

Provides the python module base directory search path

The method uses the `resource_filename` method to identify files and directories from the application

Parameters `filename` (*string*) – relative project file

Returns absolute file path name

Return type `str`

`to_profile`(*profile*)

Implements method to add list of profile keys and their values to the specified instance of a Profile class

Parameters `profile` (*object*) – Profile instance

10.1.7 kiwi.exceptions Module

exception `kiwi.exceptions.KiwiArchiveSetupError`(*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if an unsupported image archive type is used.

exception `kiwi.exceptions.KiwiArchiveTarError`(*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if impossible to determine which tar command version is installed on the underlying system

exception `kiwi.exceptions.KiwiBootImageDumpError`(*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if an instance of `BootImage*` can not be serialized on as file via pickle dump

exception `kiwi.exceptions.KiwiBootImageSetupError`(*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if an unsupported initrd system type is used.

exception `kiwi.exceptions.KiwiBootLoaderConfigSetupError`(*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if a configuration for an unsupported bootloader is requested.

exception `kiwi.exceptions.KiwiBootLoaderGrubDataError` (*message*)
Bases: `kiwi.exceptions.KiwiError`

Exception raised if no grub installation was found.

exception `kiwi.exceptions.KiwiBootLoaderGrubFontError` (*message*)
Bases: `kiwi.exceptions.KiwiError`

Exception raised if no grub unicode font was found.

exception `kiwi.exceptions.KiwiBootLoaderGrubInstallError` (*message*)
Bases: `kiwi.exceptions.KiwiError`

Exception raised if grub install to master boot record has failed.

exception `kiwi.exceptions.KiwiBootLoaderGrubModulesError` (*message*)
Bases: `kiwi.exceptions.KiwiError`

Exception raised if the synchronisation of modules from the grub installation to the boot space has failed.

exception `kiwi.exceptions.KiwiBootLoaderGrubPlatformError` (*message*)
Bases: `kiwi.exceptions.KiwiError`

Exception raised if an attempt was made to use grub on an unsupported platform.

exception `kiwi.exceptions.KiwiBootLoaderGrubSecureBootError` (*message*)
Bases: `kiwi.exceptions.KiwiError`

Exception raised if the Microsoft signed shim loader or grub2 loader could not be found in the image root system

exception `kiwi.exceptions.KiwiBootLoaderInstallSetupError` (*message*)
Bases: `kiwi.exceptions.KiwiError`

Exception raised if an installation for an unsupported bootloader is requested.

exception `kiwi.exceptions.KiwiBootLoaderTargetError` (*message*)
Bases: `kiwi.exceptions.KiwiError`

Exception raised if the target to read the bootloader path from is not a disk or an iso image.

exception `kiwi.exceptions.KiwiBootLoaderZiplInstallError` (*message*)
Bases: `kiwi.exceptions.KiwiError`

Exception raised if the installation of zipl has failed.

exception `kiwi.exceptions.KiwiBootLoaderZiplPlatformError` (*message*)
Bases: `kiwi.exceptions.KiwiError`

Exception raised if a configuration for an unsupported zipl architecture is requested.

exception `kiwi.exceptions.KiwiBootLoaderZiplSetupError` (*message*)
Bases: `kiwi.exceptions.KiwiError`

Exception raised if the data set to configure the zipl bootloader is incomplete.

exception `kiwi.exceptions.KiwiBootStrapPhaseFailed` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the bootstrap phase of the system prepare command has failed.

exception `kiwi.exceptions.KiwiBuildahError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised on inconsistent buildah class calls

exception `kiwi.exceptions.KiwiBundleError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the system bundle command has failed.

exception `kiwi.exceptions.KiwiCommandCapabilitiesError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception is raised when some the CommandCapabilities methods fails, usually meaning there is some issue trying to parse some command output.

exception `kiwi.exceptions.KiwiCommandError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if an external command called via a Command instance has returned with an exit code != 0 or could not be called at all.

exception `kiwi.exceptions.KiwiCommandNotFound` (*message*)

Bases: `kiwi.exceptions.KiwiCommandError`

Exception raised if any executable command cannot be found in the environment PATH variable.

exception `kiwi.exceptions.KiwiCommandNotLoaded` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if a kiwi command task module could not be loaded.

exception `kiwi.exceptions.KiwiCompatError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the given kiwi compatibility command line could not be understood by the compat option parser.

exception `kiwi.exceptions.KiwiCompressionFormatUnknown` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the compression format of the data could not be detected.

exception `kiwi.exceptions.KiwiConfigFileNotFound` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if no kiwi XML description was found.

exception `kiwi.exceptions.KiwiContainerBuilderError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception is raised when something fails during a container image build procedure.

exception `kiwi.exceptions.KiwiContainerImageSetupError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if an attempt to create a container instance for an unsupported container type is performed.

exception `kiwi.exceptions.KiwiContainerSetupError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if an error in the creation of the container archive happened.

exception `kiwi.exceptions.KiwiDataStructureError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the XML description failed to parse the data structure.

exception `kiwi.exceptions.KiwiDebootstrapError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if not enough user data to call debootstrap were provided or the debootstrap has failed.

exception `kiwi.exceptions.KiwiDecodingError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception is raised on decoding literals failure

exception `kiwi.exceptions.KiwiDescriptionConflict` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if both, a description file and `xml_content` is provided

exception `kiwi.exceptions.KiwiDescriptionInvalid` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the XML description failed to validate the XML schema.

exception `kiwi.exceptions.KiwiDeviceProviderError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if a storage provide is asked for its managed device but no such device exists.

exception `kiwi.exceptions.KiwiDiskBootImageError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if a kiwi boot image does not provide the requested data, e.g kernel, or hypervisor files.

exception `kiwi.exceptions.KiwiDiskFormatSetupError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if an attempt was made to create a disk format instance of an unsupported disk format.

exception `kiwi.exceptions.KiwiDiskGeometryError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the disk geometry (partition table) could not be read or evaluated against their expected geometry and capabilities.

exception `kiwi.exceptions.KiwiDistributionNameError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the distribution name could not be found. The information is extracted from the boot attribute of the XML description. If no boot attribute is present or does not match the naming conventions the exception is raised.

exception `kiwi.exceptions.KiwiError` (*message*)

Bases: `Exception`

Base class to handle all known exceptions

Specific exceptions are implemented as sub classes of KiwiError

Attributes

Parameters `message` (*string*) – Exception message text

exception `kiwi.exceptions.KiwiExtensionError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if an extension section of the same namespace is used multiple times as toplevel section within the extension section. Each extension must have a single toplevel entry point qualified by its namespace

exception `kiwi.exceptions.KiwiFileAccessError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if accessing a file or its metadata failed

exception `kiwi.exceptions.KiwiFileNotFound` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the requested file could not be found.

exception `kiwi.exceptions.KiwiFileSystemSetupError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if an attempt was made to build an unsupported or unspecified filesystem.

exception `kiwi.exceptions.KiwiFileSystemSyncError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the data sync from the system into the loop mounted filesystem image failed.

exception `kiwi.exceptions.KiwiFormatSetupError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the requested disk format could not be created.

exception `kiwi.exceptions.KiwiHelpNoCommandGiven` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the request for the help page is executed without a command to show the help for.

exception `kiwi.exceptions.KiwiImageResizeError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the request to resize a disk image failed. Reasons could be a missing raw disk reference or a wrong size specification.

exception `kiwi.exceptions.KiwiImportDescriptionError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the XML description data and scripts could not be imported into the root of the image.

exception `kiwi.exceptions.KiwiInstallBootImageError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the required files to boot an installation image could not be found, e.g kernel or hypervisor.

exception `kiwi.exceptions.KiwiInstallMediaError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if a request for an installation media is made but the system image type is not an oem type.

exception `kiwi.exceptions.KiwiInstallPhaseFailed` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the install phase of a system prepare command has failed.

exception `kiwi.exceptions.KiwiIsoLoaderError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if no isolinux loader file could be found.

exception `kiwi.exceptions.KiwiIsoMetadataError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if an inconsistency in the ISO header was found such like invalid eltorito specification or a broken path table.

exception `kiwi.exceptions.KiwiIsoToolError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if an iso helper tool such as isoinfo could not be found on the build system.

exception `kiwi.exceptions.KiwiKernelLookupError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the search for the kernel image file failed

exception `kiwi.exceptions.KiwiLiveBootImageError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if an attempt was made to use an unsupported live iso type.

exception `kiwi.exceptions.KiwiLoadCommandUndefined` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if no command is specified for a given service on the commandline.

exception `kiwi.exceptions.KiwiLogFileSetupFailed` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the log file could not be created.

exception `kiwi.exceptions.KiwiLoopSetupError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if not enough user data to create a loop device is specified.

exception `kiwi.exceptions.KiwiLuksSetupError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if not enough user data is provided to setup the luks encryption on the given device.

exception `kiwi.exceptions.KiwiMappedDeviceError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the device to become mapped does not exist.

exception `kiwi.exceptions.KiwiMountKernelFileSystemsError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if a kernel filesystem such as proc or sys could not be mounted.

exception `kiwi.exceptions.KiwiMountSharedDirectoryError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the host <-> image shared directory could not be mounted.

exception `kiwi.exceptions.KiwiNotImplementedError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if a functionality is not yet implemented.

exception `kiwi.exceptions.KiwiOCIArchiveToolError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the requested OCI archive tool is not supported

exception `kiwi.exceptions.KiwiPackageManagerSetupError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if an attempt was made to create a package manager instance for an unsupported package manager.

exception `kiwi.exceptions.KiwiPackagesDeletePhaseFailed` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the packages deletion phase in system prepare fails.

exception `kiwi.exceptions.KiwiPartitionerGptFlagError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if an attempt was made to set an unknown partition flag for an entry in the GPT table.

exception `kiwi.exceptions.KiwiPartitionerMsDosFlagError` (*message*)
Bases: `kiwi.exceptions.KiwiError`

Exception raised if an attempt was made to set an unknown partition flag for an entry in the MSDOS table.

exception `kiwi.exceptions.KiwiPartitionerSetupError` (*message*)
Bases: `kiwi.exceptions.KiwiError`

Exception raised if an attempt was made to create an instance of a partitioner for an unsupported partitioner.

exception `kiwi.exceptions.KiwiPrivilegesError` (*message*)
Bases: `kiwi.exceptions.KiwiError`

Exception raised if root privileges are required but not granted.

exception `kiwi.exceptions.KiwiProfileNotFound` (*message*)
Bases: `kiwi.exceptions.KiwiError`

Exception raised if a specified profile does not exist in the XML configuration.

exception `kiwi.exceptions.KiwiPxeBootImageError` (*message*)
Bases: `kiwi.exceptions.KiwiError`

Exception raised if a required boot file e.g the kernel could not be found in the process of building a pxe image.

exception `kiwi.exceptions.KiwiRaidSetupError` (*message*)
Bases: `kiwi.exceptions.KiwiError`

Exception raised if invalid or not enough user data is provided to create a raid array on the specified storage device.

exception `kiwi.exceptions.KiwiRepositorySetupError` (*message*)
Bases: `kiwi.exceptions.KiwiError`

Exception raised if an attempt was made to create an instance of a repository for an unsupported package manager.

exception `kiwi.exceptions.KiwiRequestError` (*message*)
Bases: `kiwi.exceptions.KiwiError`

Exception raised if a package request could not be processed by the corresponding package manager instance.

exception `kiwi.exceptions.KiwiRequestedTypeError` (*message*)
Bases: `kiwi.exceptions.KiwiError`

Exception raised if an attempt was made to build an image for an unsupported image type.

exception `kiwi.exceptions.KiwiResizeRawDiskError` (*message*)
Bases: `kiwi.exceptions.KiwiError`

Exception raised if an attempt was made to resize the image disk to a smaller size than the current one. Simply shrinking a disk image file is not possible without data corruption because the partitions were setup to use the entire disk geometry as it fits into the file. A successful shrinking operation would require the filesystems and the partition table to be reduced which is not done by the provided simple storage resize method. In addition without the user overwriting the disk size in the XML setup, kiwi will calculate the minimum required size in order to store the data. Thus in almost all cases it will not be possible to store the data in a smaller disk.

exception `kiwi.exceptions.KiwiResultError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the image build result pickle information could not be created or loaded.

exception `kiwi.exceptions.KiwiRootDirExists` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the specified image root directory already exists and should not be re-used.

exception `kiwi.exceptions.KiwiRootImportError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception is raised when something fails during the root import procedure.

exception `kiwi.exceptions.KiwiRootInitCreationError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the initialization of a new image root directory has failed.

exception `kiwi.exceptions.KiwiRpmDirNotRemoteError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the provided rpm-dir repository is not local

exception `kiwi.exceptions.KiwiRuntimeConfigFormatError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the expected format in the yaml KIWI runtime config file does not match

exception `kiwi.exceptions.KiwiRuntimeError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if a runtime check has failed.

exception `kiwi.exceptions.KiwiSatSolverJobError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if a sat solver job can not be done, e.g because the requested package or collection does not exist in the registered repository metadata

exception `kiwi.exceptions.KiwiSatSolverJobProblems` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the sat solver operations returned with solver problems e.g package conflicts

exception `kiwi.exceptions.KiwiSatSolverPluginError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the python solv module failed to load. The solv module is provided by SUSE's rpm package python-solv and provides a python binding to the libsolv C library

exception `kiwi.exceptions.KiwiSchemaImportError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the schema file could not be read by lxml.RelaxNG.

exception `kiwi.exceptions.KiwiScriptFailed` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if a user script returned with an exit code != 0.

exception `kiwi.exceptions.KiwiSetupIntermediateConfigError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the setup of the temporary image system configuration for the duration of the build process has failed.

exception `kiwi.exceptions.KiwiSizeError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception is raised when the conversion from a given size in string format to a number.

exception `kiwi.exceptions.KiwiSolverRepositorySetupError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the repository type is not supported for the creation of a SAT solvable

exception `kiwi.exceptions.KiwiSystemDeletePackagesFailed` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the deletion of a package has failed in the corresponding package manager instance.

exception `kiwi.exceptions.KiwiSystemInstallPackagesFailed` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the installation of a package has failed in the corresponding package manager instance.

exception `kiwi.exceptions.KiwiSystemUpdateFailed` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the package upgrade has failed in the corresponding package manager instance.

exception `kiwi.exceptions.KiwiTargetDirectoryNotFound` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the specified target directory to store the image results was not found.

exception `kiwi.exceptions.KiwiTemplateError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the substitution of variables in a configuration file template has failed.

exception `kiwi.exceptions.KiwiTypeNotFound` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if no build type was found in the XML description.

exception `kiwi.exceptions.KiwiUnknownServiceName` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if an unknown service name was provided on the commandline.

exception `kiwi.exceptions.KiwiUriOpenError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the urllib urlopen request has failed

exception `kiwi.exceptions.KiwiUriStyleUnknown` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if an unsupported URI style was used in the source definition of a repository.

exception `kiwi.exceptions.KiwiUriTypeUnknown` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the protocol type of an URI is unknown in the source definition of a repository.

exception `kiwi.exceptions.KiwiValidationError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the XML validation against the schema has failed.

exception `kiwi.exceptions.KiwiVhdTagError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the GUID tag is not provided in the expected format.

exception `kiwi.exceptions.KiwiVolumeGroupConflict` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the requested LVM volume group already is in use on the build system.

exception `kiwi.exceptions.KiwiVolumeManagerSetupError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the preconditions for volume mangement support are not met or an attempt was made to create an instance of a volume manager for an unsupported volume management system.

exception `kiwi.exceptions.KiwiVolumeRootIDError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if the root volume can not be found. This concept currently exists only for the btrfs subvolume system.

10.1.8 `kiwi.firmware` Module

class `kiwi.firmware.FirmWare` (*xml_state*)

Bases: `object`

Implements firmware specific methods

According to the selected firmware some parameters in a disk image changes. This class provides methods to provide firmware dependant information

- **param object `xml_state`** instance of `XMLState`

`bios_mode()`

Check if BIOS mode is requested

Returns True or False

Return type `bool`

`ec2_mode()`

Check if EC2 mode is requested

Returns True or False

Return type `bool`

`efi_mode()`

Check if EFI mode is requested

Returns The requested EFI mode or None if no EFI mode requested

Return type `str`

`get_efi_partition_size()`

Size of EFI partition. Returns 0 if no such partition is needed

Returns mbsize value

Return type `int`

`get_legacy_bios_partition_size()`

Size of legacy bios_grub partition if legacy BIOS mode is required. Returns 0 if no such partition is needed

Returns mbsize value

Return type `int`

`get_partition_table_type()`

Provides partition table type according to architecture and firmware

Returns partition table name

Return type `str`

get_prep_partition_size()

Size of Prep partition if OFW mode is requested. Returns 0 if no such partition is needed

Returns mbsize value

Return type `int`

legacy_bios_mode()

Check if the legacy boot from BIOS systems should be activated

Returns True or False

Return type `bool`

ofw_mode()

Check if OFW mode is requested

Returns True or False

Return type `bool`

opal_mode()

Check if Opal mode is requested

Returns True or False

Return type `bool`

10.1.9 kiwi.help Module

class `kiwi.help.Help`

Bases: `object`

Implements man page help for kiwi commands

Each kiwi command implements their own manual page, which is shown if the positional argument 'help' is passed to the command.

show (*command=None*)

Call man to show the command specific manual page

All kiwi commands store their manual page in the section '8' of the man system. The calling process is replaced by the man process

Parameters **command** (*string*) – man page name

10.1.10 kiwi.kiwi Module

`kiwi.kiwi.extras(help_, version, options, doc)`

Overwritten method from docopt

Shows our own usage message for `-h|--help`

Parameters

- **help** (*bool*) – indicate to show help
- **version** (*string*) – version string
- **options** (*list*) –

list of option tuples

`[option(name='name', value='value')]`

Parameters doc (*string*) – docopt doc string

`kiwi.kiwi.main()`

kiwi - main application entry point

Initializes a global log object and handles all errors of the application. Every known error is inherited from `KiwiError`, everything else is passed down until the generic `Exception` which is handled as unexpected error including the python backtrace

`kiwi.kiwi.usage(command_usage)`

Instead of the docopt way to show the usage information we provide a kiwi specific usage information. The usage data now always consists out of:

1. the generic call `kiwi-ng [global options] service <command> [<args>]`
2. the command specific usage defined by the docopt string short form by default, long form with `-h|--help`
3. the global options

Parameters command_usage (*string*) – usage data

10.1.11 kiwi.logger Module

class `kiwi.logger.Logger(name)`

Bases: `logging.Logger`

Extended logging facility based on Python logging

Parameters name (*string*) – name of the logger

getLogLevel()

Return currently used log level

Returns log level number

Return type `int`

get_logfile()

Return file path name of logfile

Returns file path

Return type `str`

progress(*current*, *total*, *prefix*, *bar_length*=40)

Custom progress log information. progress information is intentionally only logged to stdout and will bypass any handlers. We don't want this information to show up in the log file

Parameters

- **current** (`int`) – current item
- **total** (`int`) – total number of items
- **prefix** (`string`) – prefix name
- **bar_length** (`int`) – length of progress bar

setLogLevel(*level*)

Set custom log level for all console handlers

Parameters **level** (`int`) – log level number

set_color_format()

Set color format for all console handlers

set_logfile(*filename*)

Set logfile handler

Parameters **filename** (`string`) – logfile file path

10.1.12 kiwi.logger_color_formatter Module

class `kiwi.logger_color_formatter.ColorFormatter`(*args,
**kwargs)

Bases: `logging.Formatter`

Extended standard logging Formatter

Extended format supporting text with color metadata

Example:

```
ColorFormatter(message_format, '%H:%M:%S')
```

format(*record*)

Creates a logging Formatter with support for color messages

Parameters **record** (`tuple`) – logging message record

Returns result from `format_message`

Return type `str`

class `kiwi.logger_color_formatter.ColorMessage`

Bases: `object`

Implements color messages for Python logging facility

Has to implement the `format_message` method to serve as message formatter

format_message (*level, message*)

Message formatter with support for embedded color sequences

The Message is allowed to contain the following color metadata:

- `$RESET`, reset to no color mode
- `$BOLD`, bold
- `$COLOR`, color the following text
- `$LIGHTCOLOR`, light color the following text

The color of the message depends on the level and is defined in the `ColorMessage` constructor

Parameters

- **level** (*int*) – color level number
- **message** (*string*) – text

Returns color message with escape sequences

Return type `str`

10.1.13 `kiwi.logger_filter` Module

class `kiwi.logger_filter.DebugFilter` (*name=""*)

Bases: `logging.Filter`

Extended standard debug logging Filter

filter (*record*)

Only messages with record level `DEBUG` can pass for messages with another level an extra handler is used

Parameters **record** (*tuple*) – logging message record

Returns `record.name`

Return type `str`

class `kiwi.logger_filter.ErrorFilter` (*name=""*)

Bases: `logging.Filter`

Extended standard error logging Filter

filter (*record*)

Only messages with record level DEBUG can pass for messages with another level an extra handler is used

Parameters **record** (*tuple*) – logging message record

Returns record.name

Return type str

class kiwi.logger_filter.**InfoFilter** (*name=""*)

Bases: logging.Filter

Extended standard logging Filter

filter (*record*)

Only messages with record level INFO and WARNING can pass for messages with another level an extra handler is used

Parameters **record** (*tuple*) – logging message record

Returns record.name

Return type str

class kiwi.logger_filter.**LoggerSchedulerFilter** (*name=""*)

Bases: logging.Filter

Extended standard logging Filter

filter (*record*)

Messages from apscheduler scheduler instances are filtered out They conflict with console progress information

Parameters **record** (*tuple*) – logging message record

Returns record.name

Return type str

class kiwi.logger_filter.**WarningFilter** (*name=""*)

Bases: logging.Filter

Extended standard warning logging Filter

filter (*record*)

Only messages with record level WARNING can pass for messages with another level an extra handler is used

Parameters **record** (*tuple*) – logging message record

Returns record.name

Return type str

10.1.14 `kiwi.mount_manager` Module

class `kiwi.mount_manager.MountManager` (*device*, *mountpoint=None*)

Bases: `object`

Implements methods for mounting, unmounting and mount checking

If a `MountManager` instance is used to mount a device the caller must care for the time when `umount` needs to be called. The class does not automatically release the mounted device, which is intentional

- **param string device** device node name
- **param string mountpoint** mountpoint directory name

bind_mount ()

Bind mount the device to the mountpoint

is_mounted ()

Check if mounted

Returns True or False

Return type `bool`

mount (*options=None*)

Standard mount the device to the mountpoint

Parameters **options** (*list*) – mount options

umount ()

Umount by the mountpoint directory

If the resource is busy the call will return False

Returns True or False

Return type `bool`

umount_lazy ()

Umount by the mountpoint directory in lazy mode

Release the mount in any case, however the time when the mounted resource is released by the kernel depends on when the resource enters the non busy state

10.1.15 `kiwi.path` Module

class `kiwi.path.Path`

Bases: `object`

Directory path helpers

static access (*path*, *mode*, ***kwargs*)

Check whether path can be accessed with the given mode.

Parameters

- **path** (*str*) – The path that should be checked for access.
- **mode** (*int*) – Which access mode should be checked. This value must be a bit-wise or of one or more of the following constants:
`os.F_OK` (note that this one is zero), `os.X_OK`, `os.R_OK` and `os.W_OK`
- **kwargs** – further keyword arguments are forwarded to `os.access()`

Returns Boolean value whether this access mode is allowed

Return type `bool`

Raises

- **ValueError** – if the supplied mode is invalid
- **`kiwi.exceptions.KiwiFileNotFound`** – if the path does not exist or is not accessible by the current user

static create (*path*)

Create path and all sub directories to target

Parameters **path** (*string*) – path name

static move_to_root (*root, elements*)

Change the given path elements to a new root directory

Parameters

- **root** (*str*) – the root path to trim
- **elements** (*list*) – list of path names

Returns changed elements

Return type `list`

static rebase_to_root (*root, elements*)

Include the root prefix for the given paths elements

Parameters

- **root** (*str*) – the new root path
- **elements** (*list*) – list of path names

Returns changed elements

Return type `list`

static remove (*path*)

Delete empty path, causes an error if target is not empty

Parameters **path** (*string*) – path name

static remove_hierarchy (*path*)

Recursively remove an empty path and its sub directories ignore non empty or protected paths and leave them untouched

Parameters `path` (*string*) – path name

static `sort_by_hierarchy` (*path_list*)

Sort given list of path names by their hierarchy in the tree

Example:

```
result = Path.sort_by_hierarchy(['/var/lib', '/var'])
```

Parameters `path_list` (*list*) – list of path names

Returns hierarchy sorted `path_list`

Return type *list*

static `which` (*filename*, *alternative_lookup_paths=None*, *custom_env=None*,
access_mode=None, *root_dir=None*)

Lookup file name in PATH

Parameters

- **filename** (*string*) – file base name
- **alternative_lookup_paths** (*list*) – list of additional lookup paths
- **custom_env** (*list*) – a custom `os.environ`
- **access_mode** (*int*) – one of the `os` access modes or a combination of them (`os.R_OK`, `os.W_OK` and `os.X_OK`). If the provided access mode does not match the file is considered not existing
- **root_dir** (*str*) – the root path to look at

Returns absolute path to file or `None`

Return type *str*

static `wipe` (*path*)

Delete path and all contents

Parameters `path` (*string*) – path name

10.1.16 kiwi.privileges Module

class `kiwi.privileges.Privileges`

Bases: *object*

Implements check for root privileges

static `check_for_root_permissions` ()

Check if we are effectively root on the system. If not an exception is thrown

Returns True or raise an Exception

Return type *bool*

10.1.17 kiwi.runtime_checker Module

class `kiwi.runtime_checker.RuntimeChecker` (*xml_state*)

Bases: `object`

Implements build consistency checks at runtime

The schema of an image description covers structure and syntax of the provided data. The `RuntimeChecker` provides methods to perform further semantic checks which allows to recognize potential build or boot problems early.

- **param object `xml_state`** Instance of `XMLState`

`check_appx_naming_conventions_valid()`

When building wsl images there are some naming conventions that must be fulfilled to run the container on Microsoft Windows

`check_architecture_supports_iso_firmware_setup()`

For creating ISO images a different bootloader setup is performed depending on the configured firmware. If the firmware is set to bios, isolinux is used and that limits the architecture to x86 only. In any other case the appliance configured bootloader is used. This check examines if the host architecture is supported with the configured firmware on request of an ISO image.

`check_boot_description_exists()`

If a kiwi initrd is used, a lookup to the specified boot description is done and fails early if it does not exist

`check_consistent_kernel_in_boot_and_system_image()`

If a kiwi initrd is used, the kernel used to build the kiwi initrd and the kernel used in the system image must be the same in order to avoid an inconsistent boot setup

`check_container_tool_chain_installed()`

When creating container images the specific tools are used in order to import and export OCI or Docker compatible images. This check searches for those tools to be installed in the build system and fails if it can't find them

`check_dracut_module_for_disk_oem_in_package_list()`

OEM images if configured to use dracut as initrd system requires the KIWI provided `dracut-kiwi-oem-repart` module to be installed at the time dracut is called. Thus this runtime check examines if the required package is part of the package list in the image description.

`check_dracut_module_for_disk_overlay_in_package_list()`

Disk images configured to use a root filesystem overlay requires the KIWI provided `kiwi-overlay` dracut module to be installed at the time dracut is called. Thus this runtime check examines if the required package is part of the package list in the image description.

`check_dracut_module_for_live_iso_in_package_list()`

Live ISO images uses a dracut initrd to boot and requires the KIWI provided `kiwi-live` dracut module to be installed at the time dracut is called. Thus this runtime

check examines if the required package is part of the package list in the image description.

check_dracut_module_for_oem_install_in_package_list()

OEM images if configured to use dracut as initrd system and configured with one of the installiso, installstick or installpxe attributes requires the KIWI provided dracut-kiwi-oem-dump module to be installed at the time dracut is called. Thus this runtime check examines if the required package is part of the package list in the image description.

check_efi_mode_for_disk_overlay_correctly_setup()

Disk images configured to use a root filesystem overlay only supports the standard EFI mode and not secure boot. That's because the shim setup performs changes to the root filesystem which can not be applied during the bootloader setup at build time because at that point the root filesystem is a read-only squashfs source.

check_image_include_repos_publicly_resolvable()

Verify that all repos marked with the imageinclude attribute can be resolved into a http based web URL

check_mediacheck_installed()

If the image description enables the mediacheck attribute the required tools to run this check must be installed on the image build host

check_minimal_required_preferences()

Kiwi requires some of the elements of the preferences element to be present at least in one of the preferences section. This runtime check validates <version> and <packagemanager> are provided.

check_repositories_configured()

Verify that that there are repositories configured

check_target_directory_not_in_shared_cache(target_dir)

The target directory must be outside of the kiwi shared cache directory in order to avoid busy mounts because kiwi bind mounts the cache directory into the image root tree to access host caching information

Parameters *target_dir* (*string*) – path name

check_volume_label_used_with_lvm()

The optional volume label in a systemdisk setup is only effective if the LVM, logical volume manager system is used. In any other case where the filesystem itself offers volume management capabilities there are no extra filesystem labels which can be applied per volume

check_volume_setup_defines_multiple_fullsize_volumes()

The volume size specification 'all' makes this volume to take the rest space available on the system. It's only allowed to specify one all size volume

check_volume_setup_defines_reserved_labels()

check_volume_setup_has_no_root_definition()

The root volume in a systemdisk setup is handled in a special way. It is not allowed to setup a custom name or mountpoint for the root volume. Therefore the size of

the root volume can be setup via the @root volume name. This check looks up the volume setup and searches if there is a configuration for the '/' mountpoint which would cause the image build to fail

check_xen_uniquely_setup_as_server_or_guest()

If the image is classified to be used as Xen image, it can be either a Xen Server(dom0) or a Xen guest. The image configuration is checked if the information uniquely identifies the image as such

10.1.18 kiwi.runtime_config Module

class `kiwi.runtime_config.RuntimeConfig`

Bases: `object`

Implements reading of runtime configuration file:

1. `~/.config/kiwi/config.yml`
2. `/etc/kiwi.yml`

The KIWI runtime configuration file is a yaml formatted file containing information to control the behavior of the tools used by KIWI.

get_bundle_compression (*default=True*)

Return boolean value to express if the image bundle should contain XZ compressed image results or not.

bundle:

- `compress: true/false`

If compression of image build results is activated the size of the bundle is smaller and the download speed increases. However the image must be uncompressed before use

If no compression is explicitly configured, the provided default value applies

Parameters `default` (*bool*) – Default value

Returns True or False

Return type `bool`

get_container_compression ()

Return compression algorithm to use for compression of container images

container:

- `compress: xz/none`

if no or invalid configuration data is provided, the default compression algorithm from the Defaults class is returned

Returns A name

Return type `str`

get_disabled_runtime_checks()

Returns disabled runtime checks. Checks can be disabled with:

runtime_checks:

- disable: check_container_tool_chain_installed

if the provided string does not match any RuntimeChecker method it is just ignored.

get_iso_tool_category()

Return tool category which should be used to build iso images

iso:

- tool_category: cdrtoolslxorriso

if no or invalid configuration exists the default tool category from the Defaults class is returned

Returns A name

Return type `str`

get_max_size_constraint()

Returns the maximum allowed size of the built image. The value is returned in bytes and it is specified in build_constraints element with the max_size attribute. The value can be specified in bytes or it can be specified with m=MB or g=GB.

build_constraints:

- max_size: 700m

if no configuration exists None is returned

Returns byte value or None

Return type `int`

get_obs_download_server_url()

Return URL of buildservice download server in:

obs:

- download_url: ...

if no configuration exists the downloadserver from the Defaults class is returned

Returns URL type data

Return type `str`

get_oci_archive_tool()

Return OCI archive tool which should be used on creation of container archives for OCI compliant images, e.g docker

oci:

- archive_tool: umoci

if no configuration exists the default tool from the Defaults class is returned

Returns A name

Return type `str`

get_xz_options()

Return list of XZ compression options in:

xz:

- options: ...

if no configuration exists None is returned

Returns

Contains list of options

```
[ '--option=value' ]
```

Return type `list`

is_obs_public()

Check if the buildservice configuration is public or private in:

obs:

- public: true/false

if no configuration exists we assume to be public

Returns True or False

Return type `bool`

10.1.19 `kiwi.version` Module

Global version information used in kiwi and the package

10.1.20 `kiwi.xml_description` Module

```
class kiwi.xml_description.XMLDescription(description=None,  
                                           derived_from=None,  
                                           xml_content=None)
```

Bases: `object`

Implements data management for the XML description

- XSLT Style Sheet processing to apply on this version of kiwi
- Schema Validation based on RelaxNG schema
- Loading XML data into internal data structures

Attributes

Parameters

- **description** (*string*) – path to XML description file
- **derived_from** (*string*) – path to base XML description file
- **xml_content** (*string*) – XML description data as content string

get_extension_xml_data (*namespace_name*)

Return the xml etree parse result for the specified extension namespace

Parameters **namespace_name** (*string*) – name of the extension namespace

Returns result of etree.parse

Return type *object*

load ()

Read XML description, pass it along to the XSLT processor, validate it against the schema and finally pass it to the autogenerated(generateDS) parser.

Returns instance of XML toplevel domain (image)

Return type *object*

10.1.21 kiwi.xml_state Module

class `kiwi.xml_state.XMLState` (*xml_data*, *profiles=None*,
build_type=None)

Bases: *object*

Implements methods to get stateful information from the XML data

Parameters

- **xml_data** (*object*) – parse result from XMLDescription.load()
- **profiles** (*list*) – list of used profiles
- **build_type** (*object*) – build <type> section reference

add_container_config_label (*label_name*, *value*)

Adds a new label in the containerconfig section, if a label with the same name is already defined in containerconfig it gets overwritten by this method.

Parameters

- **label_name** (*str*) – the string representing the label name
- **value** (*str*) – the value of the label

add_repository (*repo_source*, *repo_type*, *repo_alias*,
repo_prio, *repo_imageinclude=False*,
repo_package_gpgcheck=None)

Add a new repository section at the end of the list

Parameters

- **repo_source** (*string*) – repository URI

- **repo_type** (*string*) – type name defined by schema
- **repo_alias** (*string*) – alias name
- **repo_prio** (*string*) – priority number, package manager specific
- **repo_imageinclude** (*boolean*) – setup repository inside of the image
- **repo_package_gpgcheck** (*boolean*) – enable/disable package gpg checks

copy_bootdelete_packages (*target_state*)

Copy packages marked as bootdelete to the packages type=delete section in the target xml state

Parameters **target_state** (*object*) – XMLState instance

copy_bootincluded_archives (*target_state*)

Copy archives marked as bootinclude to the packages type=bootstrap section in the target xml state

Parameters **target_state** (*object*) – XMLState instance

copy_bootincluded_packages (*target_state*)

Copy packages marked as bootinclude to the packages type=image (or type=bootstrap if no type=image was found) section in the target xml state. The package will also be removed from the packages type=delete section in the target xml state if present there

Parameters **target_state** (*object*) – XMLState instance

copy_build_type_attributes (*attribute_names*, *target_state*)

Copy specified attributes from this build type section to the target xml state build type section

Parameters

- **attribute_names** (*list*) – type section attributes
- **target_state** (*object*) – XMLState instance

copy_displayname (*target_state*)

Copy image displayname from this xml state to the target xml state

Parameters **target_state** (*object*) – XMLState instance

copy_drivers_sections (*target_state*)

Copy drivers sections from this xml state to the target xml state

Parameters **target_state** (*object*) – XMLState instance

copy_machine_section (*target_state*)

Copy machine sections from this xml state to the target xml state

Parameters **target_state** (*object*) – XMLState instance

copy_name (*target_state*)

Copy image name from this xml state to the target xml state

Parameters **target_state** (*object*) – XMLState instance

copy_oemconfig_section (*target_state*)

Copy oemconfig sections from this xml state to the target xml state

Parameters **target_state** (*object*) – XMLState instance

copy_preferences_subsections (*section_names*, *target_state*)

Copy subsections of the preferences sections, matching given section names, from this xml state to the target xml state

Parameters

- **section_names** (*list*) – preferences subsection names
- **target_state** (*object*) – XMLState instance

copy_repository_sections (*target_state*, *wipe=False*)

Copy repository sections from this xml state to the target xml state

Parameters

- **target_state** (*object*) – XMLState instance
- **wipe** (*bool*) – delete all repos in target prior to copy

copy_strip_sections (*target_state*)

Copy strip sections from this xml state to the target xml state

Parameters **target_state** (*object*) – XMLState instance

copy_systemdisk_section (*target_state*)

Copy systemdisk sections from this xml state to the target xml state

Parameters **target_state** (*object*) – XMLState instance

delete_repository_sections ()

Delete all repository sections matching configured profiles

delete_repository_sections_used_for_build ()

Delete all repository sections used to build the image matching configured profiles

get_bootstrap_archives ()

List of archive names from the type="bootstrap" packages section(s)

Returns archive names

Return type *list*

get_bootstrap_collection_type ()

Collection type for packages sections matching type="bootstrap"

Returns collection type name

Return type *str*

get_bootstrap_collections()

List of collection names from the packages sections matching type="bootstrap"

Returns collection names

Return type `list`

get_bootstrap_packages(*plus_packages=None*)

List of package names from the type="bootstrap" packages section(s)

The list gets the selected package manager appended if there is a request to install packages inside of the image via a chroot operation

Parameters **plus_packages** (`list`) – list of additional packages

Returns package names

Return type `list`

get_bootstrap_packages_sections()

List of packages sections matching type="bootstrap"

Returns list of <packages> section reference(s)

Return type `list`

get_bootstrap_products()

List of product names from the packages sections matching type="bootstrap"

Returns product names

Return type `list`

get_build_type_containerconfig_section()

First containerconfig section from the build type section

Returns <containerconfig> section reference

Return type `xml_parse::containerconfig`

get_build_type_format_options()

Disk format options returned as a dictionary

Returns format options

Return type `dict`

get_build_type_machine_section()

First machine section from the build type section

Returns <machine> section reference

Return type `xml_parse::machine`

get_build_type_name()

Default build type name

Returns Content of image attribute from build type

Return type `str`

get_build_type_oemconfig_section()

First oemconfig section from the build type section

Returns <oemconfig> section reference

Return type xml_parse::oemconfig

get_build_type_size(include_unpartitioned=False)

Size information from the build type section. If no unit is set the value is treated as mbytes

Parameters **include_unpartitioned** (*bool*) – sets if the unpartitioned area should be included in the computed size or not

Returns mbytes

Return type int

get_build_type_spare_part_fs_attributes()

Build type specific list of filesystem attributes applied to the spare partition.

Returns list of strings or empty list

Return type list

get_build_type_spare_part_size()

Size information for the spare_part size from the build type. If no unit is set the value is treated as mbytes

Returns mbytes

Return type int

get_build_type_system_disk_section()

First system disk section from the build type section

Returns <systemdisk> section reference

Return type xml_parse::systemdisk

get_build_type_unpartitioned_bytes()

Size of the unpartitioned area for image in megabytes

Returns mbytes

Return type int

get_build_type_vagrant_config_section()

First vagrantconfig section from the build type section

Returns <vagrantconfig> section reference

Return type xml_parse::vagrantconfig

get_build_type_vmconfig_entries()

List of vmconfig-entry section values from the first machine section in the build type section

Returns <vmconfig_entry> section reference(s)

Return type `list`

get_build_type_vmdisk_section()

First vmdisk section from the first machine section in the build type section

Returns <vmdisk> section reference

Return type `xml_parse::vmdisk`

get_build_type_vmdvd_section()

First vmdvd section from the first machine section in the build type section

Returns <vmdvd> section reference

Return type `xml_parse::vmdvd`

get_build_type_vmnic_entries()

vmnic section(s) from the first machine section in the build type section

Returns list of <vmnic> section reference(s)

Return type `list`

get_collection_type (*section_type='image'*)

Collection type from packages sections matching given section type.

If no collection type is specified the default collection type is set to: onlyRequired

Parameters **section_type** (*string*) – type name from packages section

Returns collection type name

Return type `str`

get_collections (*section_type='image'*)

List of collection names from the packages sections matching type=section_type and type=build_type

Returns collection names

Return type `list`

get_container_config()

Dictionary of containerconfig information

Takes attributes and subsection data from the selected <containerconfig> section and stores it in a dictionary

get_derived_from_image_uri()

Uri object of derived image if configured

Specific image types can be based on a master image. This method returns the location of this image when configured in the XML description

Returns Instance of Uri

Return type `object`

get_description_section()

The description section

Returns description_type tuple providing the elements author contact and specification

Return type tuple

get_disk_start_sector()

First disk sector number to be used by the first disk partition.

Returns number

Return type int

get_distribution_name_from_boot_attribute()

Extract the distribution name from the boot attribute of the build type section.

If no boot attribute is configured or the contents does not match the kiwi defined naming schema for boot image descriptions, an exception is thrown

Returns lowercase distribution name

Return type str

get_drivers_list()

List of driver names from all drivers sections matching configured profiles

Returns driver names

Return type list

get_fs_create_option_list()

List of root filesystem creation options

The list contains elements with the information from the fscreateoptions attribute string that got split into its substring components

Returns list with create options

Return type list

get_fs_mount_option_list()

List of root filesystem mount options

The list contains one element with the information from the fsmountoptions attribute. The value there is passed along to the -o mount option

Returns max one element list with mount option string

Return type list

get_image_packages_sections()

List of packages sections matching type="image"

Returns list of <packages> section reference(s)

Return type list

get_image_version()

Image version from preferences section.

Multiple occurrences of version in preferences sections are not forbidden, however only the first version found defines the final image version

Returns Content of <version> section

Return type `str`

get_initrd_system()

Name of initrd system to use

Depending on the image type a specific initrd system is either pre selected or free of choice according to the XML type setup

Returns dracut, kiwi or None

Return type `str`

get_locale()

Gets list of locale names if configured. Takes the first locale setup from the existing preferences sections into account.

Returns List of names or None

Return type `list|None`

get_oemconfig_oem_multipath_scan()

State value to activate multipath maps. Returns a boolean value if specified or None

Returns Content of <oem-multipath-scan> section value

Return type `bool`

get_oemconfig_swap_mbytes()

Return swapsize in MB if requested or None

Operates on the value of oem-swap and if set to true returns the given size or the default value.

Returns Content of <oem-swapsize> section value or default

Return type `int`

get_package_manager()

Get configured package manager from selected preferences section

Returns Content of the <packagemanager> section

Return type `str`

get_package_sections(*packages_sections*)

List of package sections from the given packages sections. Each list element contains a tuple with the <package> section reference and the <packages> section this package belongs to

If a package entry specifies an architecture, it is only taken if the host architecture matches the configured architecture

Parameters **packages_sections** (*list*) – <packages>

Returns

Contains list of package_type tuples

```
[package_type(packages_section=object, package_
↪section=object) ]
```

Return type *list*

get_packages_sections (*section_types*)

List of packages sections matching given section type(s)

Parameters **section_types** (*list*) – type name(s) from packages sections

Returns list of <packages> section reference(s)

Return type *list*

get_preferences_sections ()

All preferences sections for the selected profiles

Returns list of <preferences> section reference(s)

Return type *list*

get_products (*section_type='image'*)

List of product names from the packages sections matching type=section_type and type=build_type

Parameters **section_type** (*string*) – type name from packages section

Returns product names

Return type *list*

get_repository_sections ()

List of all repository sections matching configured profiles

Returns <repository> section reference(s)

Return type *list*

get_repository_sections_used_for_build ()

List of all repositorys sections used to build the image and matching configured profiles.

Returns <repository> section reference(s)

Return type *list*

get_repository_sections_used_in_image ()

List of all repositorys sections to be configured in the resulting image matching configured profiles.

Returns <repository> section reference(s)

Return type `list`

get_root_partition_uuid()

Return preserved PARTUUID

get_rpm_check_signatures()

Gets the rpm-check-signatures configuration flag. Returns False if not present.

Returns True or False

Return type `bool`

get_rpm_excludedocs()

Gets the rpm-excludedocs configuration flag. Returns False if not present.

Returns True or False

Return type `bool`

get_rpm_locale()

Gets list of locale names to filter out by rpm if rpm-locale-filtering is switched on the the list always contains: [POSIX, C, C.UTF-8] and is extended by the optionally configured locale

Returns List of names or None

Return type `list|None`

get_rpm_locale_filtering()

Gets the rpm-locale-filtering configuration flag. Returns False if not present.

Returns True or False

Return type `bool`

get_strip_files_to_delete()

Items to delete from strip section

Returns item names

Return type `list`

get_strip_libraries_to_keep()

Libraries to keep from strip section

Returns librarie names

Return type `list`

get_strip_list(*section_type*)

List of strip names matching the given section type and profiles

Parameters **section_type** (*string*) – type name from packages section

Returns strip names

Return type `list`

get_strip_tools_to_keep()

Tools to keep from strip section

Returns tool names

Return type `list`

get_system_archives()

List of archive names from the packages sections matching type="image" and type=build_type

Returns archive names

Return type `list`

get_system_collection_type()

Collection type for packages sections matching type="image"

Returns collection type name

Return type `str`

get_system_collections()

List of collection names from the packages sections matching type="image"

Returns collection names

Return type `list`

get_system_ignore_packages()

List of ignore package names from the packages sections matching type="image" and type=build_type

Returns package names

Return type `list`

get_system_packages()

List of package names from the packages sections matching type="image" and type=build_type

Returns package names

Return type `list`

get_system_products()

List of product names from the packages sections matching type="image"

Returns product names

Return type `list`

get_to_become_deleted_packages(*force=True*)

List of package names from the type="delete" or type="uninstall" packages section(s)

Parameters **force** (*bool*) – return "delete" type if True, "uninstall" type otherwise

Returns package names

Return type `list`

get_user_groups (*user_name*)

List of group names matching specified user

Each entry in the list is the name of a group that the specified user belongs to. The first item in the list is the login or primary group. The list will be empty if no groups are specified in the description file.

Returns groups data for the given user

Return type `list`

get_users ()

List of configured users.

Each entry in the list is a single `xml_parse::user` instance.

Returns list of <user> section reference(s)

Return type `list`

get_users_sections ()

All users sections for the selected profiles

Returns list of <users> section reference(s)

Return type `list`

get_vagrant_config_virtualbox_guest_additions ()

Attribute `virtualbox_guest_additions_present` from the first `vagrantconfig` section.

Returns `<vagrantconfig virtualbox_guest_additions_present=>value`

Return type `bool`

get_volume_group_name ()

Volume group name from selected <systemdisk> section

Returns volume group name

Return type `str`

get_volume_management ()

Provides information which volume management system is used

Returns name of volume manager

Return type `str`

get_volumes ()

List of configured `systemdisk` volumes.

Each entry in the list is a tuple with the following information

- name: name of the volume

- size: size of the volume
- realpath: system path to lookup volume data. If no mountpoint is set the volume name is used as data path.
- mountpoint: volume mount point and volume data path
- fullsize: takes all space True/False
- attributes: list of volume attributes handled via chatter

Returns

Contains list of volume_type tuples

```
[
    volume_type(
        name=volume_name,
        size=volume_size,
        realpath=path,
        mountpoint=path,
        fullsize=True,
        label=volume_label,
        attributes=['no-copy-on-write']
    )
]
```

Return type `list`

is_xen_guest()

Check if build type setup specifies a Xen Guest (domX) The check is based on the architecture, the firmware and xen_loader configuration values:

- We only support Xen setup on the x86_64 architecture
- Firmware pointing to ec2 means the image is targeted to run in Amazon EC2 which is a Xen guest
- Machine setup with a xen_loader attribute also indicates a Xen guest target

Returns True or False

Return type `bool`

is_xen_server()

Check if build type domain setup specifies a Xen Server (dom0)

Returns True or False

Return type `bool`

package_matches_host_architecture(package)

Tests if the given package section is applicable for the current host architecture. If no architecture is specified within the section it is considered as a match returning True.

Note: The XML section pointer must provide an arch attribute

Parameters **section** – XML section object

Returns True or False

Return type `bool`

profile_matches_host_architecture (*profile*)

Tests if the given profile section is applicable for the current host architecture. If no architecture is specified within the section it is considered as a match returning True.

Note: The XML section pointer must provide an arch attribute

Parameters **section** – XML section object

Returns True or False

Return type `bool`

set_container_config_tag (*tag*)

Set new tag name in containerconfig section

In order to set a new tag value an existing containerconfig and tag setup is required

Parameters **tag** (*string*) – tag name

set_derived_from_image_uri (*uri*)

Set derived_from attribute to a new value

In order to set a new value the derived_from attribute must be already present in the image configuration

Parameters **uri** (*string*) – URI

set_repository (*repo_source*, *repo_type*, *repo_alias*,
repo_prio, *repo_imageinclude=False*,
repo_package_gpgcheck=None)

Overwrite repository data of the first repository

Parameters

- **repo_source** (*string*) – repository URI
- **repo_type** (*string*) – type name defined by schema
- **repo_alias** (*string*) – alias name
- **repo_prio** (*string*) – priority number, package manager specific
- **repo_imageinclude** (*boolean*) – setup repository inside of the image
- **repo_package_gpgcheck** (*boolean*) – enable/disable package gpg checks

set_root_partition_uuid (*uuid*)

Store PARTUUID provided in uuid as state information

Parameters **uuid** (*string*) – PARTUUID

10.1.22 Module Contents

10.2 kiwi.archive Package

10.2.1 Submodules

10.2.2 kiwi.archive.cpio Module

class `kiwi.archive.cpio.ArchiveCpio` (*filename*)

Bases: `object`

Extraction/Creation of cpio archives

Parameters **filename** (*string*) – filename to use for archive extraction or creation

create (*source_dir*, *exclude=None*)

Create cpio archive

Parameters

- **source_dir** (*string*) – data source directory
- **exclude** (*list*) – list of excluded items

extract (*dest_dir*)

Extract cpio archive contents

Parameters **dest_dir** (*string*) – target data directory

10.2.3 kiwi.archive.tar Module

class `kiwi.archive.tar.ArchiveTar` (*filename*, *create_from_file_list=True*, *file_list=None*)

Bases: `object`

Extraction/Creation of tar archives

The tarfile python module is not used by that class, since it does not provide support for some relevant features in comparison to the GNU tar command (e.g. numeric-owner). Moreover tarfile lacks support for xz compression under Python v2.7.

Parameters

- **filename** (*string*) – filename to use for archive extraction or creation
- **create_from_file_list** (*bool*) – use file list not entire directory to create the archive

- **file_list** (*list*) – list of files and directorie names to archive

append_files (*source_dir*, *files_to_append*, *options=None*)

Append files to an already existing uncompressed tar archive

Parameters

- **source_dir** (*string*) – data source directory
- **files_to_append** (*list*) – list of items to append
- **options** (*list*) – custom options

create (*source_dir*, *exclude=None*, *options=None*)

Create uncompressed tar archive

Parameters

- **source_dir** (*string*) – data source directory
- **exclude** (*list*) – list of excluded items
- **options** (*list*) – custom creation options

create_gnu_gzip_compressed (*source_dir*, *exclude=None*)

Create gzip compressed tar archive

Parameters

- **source_dir** (*string*) – data source directory
- **exclude** (*list*) – list of excluded items

create_xz_compressed (*source_dir*, *exclude=None*, *options=None*,
xz_options=None)

Create XZ compressed tar archive

Parameters

- **source_dir** (*string*) – data source directory
- **exclude** (*list*) – list of excluded items
- **options** (*list*) – custom tar creation options
- **xz_options** (*list*) – custom xz compression options

extract (*dest_dir*)

Extract tar archive contents

Parameters **dest_dir** (*string*) – target data directory

10.2.4 Module Contents

10.3 kiwi.boot.image Package

10.3.1 Submodules

10.3.2 kiwi.boot.image.base Module

```
class kiwi.boot.image.base.BootImageBase (xml_state,      target_dir,  
                                           root_dir=None,    sign-  
                                           ing_keys=None)
```

Bases: `object`

Base class for boot image(initrd) task

Parameters

- **xml_state** (*object*) – Instance of XMLState
- **target_dir** (*string*) – target dir to store the initrd
- **root_dir** (*string*) – system image root directory
- **signing_keys** (*list*) – list of package signing keys

```
create_initrd (mbrid=None, basename=None, install_initrd=False)
```

Implements creation of the initrd

Parameters

- **mbrid** (*object*) – instance of ImageIdentifier
- **basename** (*string*) – base initrd file name
- **install_initrd** (*bool*) – installation media initrd

Implementation in specialized boot image class

```
disable_cleanup ()
```

Deactivate cleanup(deletion) of boot root directory

```
dump (filename)
```

Pickle dump this instance to a file. If the object dump is requested the destructor code will also be disabled in order to preserve the generated data

Parameters **filename** (*string*) – file path name

```
enable_cleanup ()
```

Activate cleanup(deletion) of boot root directory

```
get_boot_description_directory ()
```

Provide path to the boot image XML description

Returns path name

Return type `str`

get_boot_names()

Provides kernel and initrd names for the boot image

Returns

Contains boot_names_type tuple

```
boot_names_type(  
    kernel_name='INSTALLED_KERNEL',  
    initrd_name='DRACUT_OUTPUT_NAME'  
)
```

Return type tuple

import_system_description_elements()

Copy information from the system image relevant to create the boot image to the boot image state XML description

include_file (*filename*, *install_media=False*)

Include file to boot image

For kiwi boot images this is done by adding package or archive definitions with the bootinclude attribute. Thus for kiwi boot images the method is a noop

Parameters

- **filename** (*string*) – file path name
- **install_media** (*bool*) – include also for installation media
initrd

include_module (*module*, *install_media=False*)

Include module to boot image

For kiwi boot no modules configuration is required. Thus in such a case this method is a noop.

Parameters

- **module** (*string*) – module to include
- **install_media** (*bool*) – include the module for install initrds

is_prepared()

Check if initrd system is prepared.

Returns True or False

Return type bool

load_boot_xml_description()

Load the boot image description referenced by the system image description boot attribute

omit_module (*module*, *install_media=False*)

Omit module to boot image

For kiwi boot no modules configuration is required. Thus in such a case this method is a noop.

Parameters

- **module** (*string*) – module to omit
- **install_media** (*bool*) – omit the module for install initrds

post_init()

Post initialization method

Implementation in specialized boot image class

prepare()

Prepare new root system to create initrd from. Implementation is only needed if there is no other root system available

Implementation in specialized boot image class

write_system_config_file (*config*, *config_file=None*)

Writes relevant boot image configuration into configuration file that will be part of the system image.

This is used to configure any further boot image rebuilds after deployment. For instance, initrds recreated on kernel update.

For kiwi boot no specific configuration is required for initrds recreation, thus this method is a noop in that case.

Parameters

- **config** (*dict*) – dictionary including configuration parameters
- **config_file** (*string*) – configuration file to write

10.3.3 kiwi.boot.image.dracut Module

```
class kiwi.boot.image.dracut.BootImageDracut (xml_state,  
                                             target_dir,  
                                             root_dir=None,  
                                             sign-  
                                             ing_keys=None)
```

Bases: *kiwi.boot.image.base.BootImageBase*

Implements creation of dracut boot(initrd) images.

create_initrd (*mbrid=None*, *basename=None*, *install_initrd=False*)

Create kiwi .profile environment to be included in dracut initrd. Call dracut as chroot operation to create the initrd and move the result into the image build target directory

Parameters

- **mbrid** (*object*) – unused

- **basename** (*string*) – base initrd file name
- **install_initrd** (*bool*) – installation media initrd

include_file (*filename*, *install_media=False*)

Include file to dracut boot image

Parameters **filename** (*string*) – file path name

include_module (*module*, *install_media=False*)

Include module to dracut boot image

Parameters

- **module** (*string*) – module to include
- **install_media** (*bool*) – include the module for install initrds

omit_module (*module*, *install_media=False*)

Omit module to dracut boot image

Parameters

- **module** (*string*) – module to omit
- **install_media** (*bool*) – omit the module for install initrds

post_init ()

Post initialization method

Initialize empty list of dracut caller options

prepare ()

Prepare dracut caller environment

- Setup machine_id(s) to be generic and rebuild by dracut on boot

write_system_config_file (*config*, *config_file=None*)

Writes modules configuration into a dracut configuration file.

Parameters

- **config** (*dict*) – a dictionary containing the modules to add and omit
- **conf_file** (*string*) – configuration file to write

10.3.4 kiwi.boot.image.builtin_kiwi Module

```
class kiwi.boot.image.builtin_kiwi.BootImageKiwi (xml_state,  
                                                target_dir,  
                                                root_dir=None,  
                                                sign-  
                                                ing_keys=None)
```

Bases: *kiwi.boot.image.base.BootImageBase*

Implements preparation and creation of kiwi boot(initrd) images

The kiwi initrd is a customized first boot initrd which allows to control the first boot an appliance. The kiwi initrd replaces itself after first boot by the result of dracut.

create_initrd (*mbrid=None, basename=None, install_initrd=False*)

Create initrd from prepared boot system tree and compress the result

Parameters

- **mbrid** (*object*) – instance of ImageIdentifier
- **basename** (*string*) – base initrd file name
- **install_initrd** (*bool*) – installation media initrd

post_init ()

Post initialization method

Creates custom directory to prepare the boot image root filesystem which is a separate image to create the initrd from

prepare ()

Prepare new root system suitable to create a kiwi initrd from it

10.3.5 Module Contents

class `kiwi.boot.image.BootImage`

Bases: `object`

BootImage Factory

Parameters

- **xml_state** (*object*) – Instance of XMLState
- **target_dir** (*string*) – target dir to store the initrd
- **root_dir** (*string*) – system image root directory
- **signing_keys** (*list*) – list of package signing keys

10.4 kiwi.bootloader.config Package

10.4.1 Submodules

10.4.2 `kiwi.bootloader.config.base` Module

class `kiwi.bootloader.config.base.BootLoaderConfigBase` (*xml_state, root_dir, boot_dir=None, custom_args=None*)

Bases: `object`

Base class for bootloader configuration

Parameters

- **xml_state** (*object*) – instance of XMLState
- **root_dir** (*string*) – root directory path name
- **custom_args** (*dict*) – custom bootloader arguments dictionary

create_efi_path (*in_sub_dir*='boot/efi')

Create standard EFI boot directory structure

Parameters **in_sub_dir** (*string*) – toplevel directory

Returns Full qualified EFI boot path

Return type *str*

failsafe_boot_entry_requested ()

Check if a failsafe boot entry is requested

Returns True or False

Return type *bool*

get_boot_cmdline (*uuid*=None)

Boot commandline arguments passed to the kernel

Parameters **uuid** (*string*) – boot device UUID

Returns kernel boot arguments

Return type *str*

get_boot_path (*target*='disk')

Bootloader lookup path on boot device

If the bootloader reads the data it needs to boot, it does that from the configured boot device. Depending if that device is an extra boot partition or the root partition or or based on a non standard filesystem like a btrfs snapshot, the path name varies

Parameters **target** (*string*) – target name: disk|iso

Returns path name

Return type *str*

get_boot_theme ()

Bootloader Theme name

Returns theme name

Return type *str*

get_boot_timeout_seconds ()

Bootloader timeout in seconds

If no timeout is specified the default timeout applies

Returns timeout seconds

Return type `int`

get_continue_on_timeout()

Check if the boot should continue after boot timeout or not

Returns True or False

Return type `bool`

get_gfxmode(target)

Graphics mode according to bootloader target

Bootloaders which support a graphics mode can be configured to run graphics in a specific resolution and colors. There is no standard for this setup which causes kiwi to create a mapping from the kernel vesa mode number to the corresponding bootloader graphics mode setup

Parameters **target** (*string*) – bootloader name

Returns boot graphics mode

Return type `str`

get_install_image_boot_default(loader=None)

Provide the default boot menu entry identifier for install images

The install image can be configured to provide more than one boot menu entry. Menu entries configured are:

- [0] Boot From Hard Disk
- [1] Install
- [2] Failsafe Install

The installboot attribute controls which of these are used by default. If not specified the boot from hard disk entry will be the default. Depending on the specified loader type either an entry number or name will be returned.

Parameters **loader** (*string*) – bootloader name

Returns menu name or id

Return type `str`

get_menu_entry_install_title()

Prefixed menu entry title for install images

If no displayname is specified in the image description, the menu title is constructed from the image name

Returns title text

Return type `str`

get_menu_entry_title(plain=False)

Prefixed menu entry title

If no `displayname` is specified in the image description, the menu title is constructed from the image name and build type

Parameters `plain` (*bool*) – indicate to add built type into title text

Returns title text

Return type *str*

post_init (*custom_args*)

Post initialization method

Store custom arguments by default

Parameters `custom_args` (*dict*) – custom bootloader arguments

quote_title (*name*)

Quote special characters in the title name

Not all characters can be displayed correctly in the bootloader environment. Therefore a quoting is required

Parameters `name` (*string*) – title name

Returns quoted text

Return type *str*

setup_disk_boot_images (*boot_uuid*, *lookup_path=None*)

Create bootloader images for disk boot

Some bootloaders requires to build a boot image the bootloader can load from a specific offset address or from a standardized path on a filesystem.

Parameters

- **boot_uuid** (*string*) – boot device UUID
- **lookup_path** (*string*) – custom module lookup path

Implementation in specialized bootloader class required

setup_disk_image_config (*boot_uuid*, *root_uuid*, *hypervisor*, *kernel*, *initrd*, *boot_options={}*)

Create boot config file to boot from disk.

Parameters

- **boot_uuid** (*string*) – boot device UUID
- **root_uuid** (*string*) – root device UUID
- **hypervisor** (*string*) – hypervisor name
- **kernel** (*string*) – kernel name
- **initrd** (*string*) – initrd name
- **boot_options** (*dict*) – custom options dictionary required to setup the bootloader. The scope of the options covers all information needed to setup and configure the bootloader and gets effective

in the individual implementation. `boot_options` should not be mixed up with commandline options used at boot time. This information is provided from the `get_*_cmdline` methods. The contents of the dictionary can vary between bootloaders or even not be needed

Implementation in specialized bootloader class required

setup_install_boot_images (*mbrid*, *lookup_path=None*)

Create bootloader images for ISO boot an install media

Parameters

- **mbrid** (*string*) – mbrid file name on boot device
- **lookup_path** (*string*) – custom module lookup path

Implementation in specialized bootloader class required

setup_install_image_config (*mbrid*, *hypervisor*, *kernel*, *initrd*)

Create boot config file to boot from install media in EFI mode.

Parameters

- **mbrid** (*string*) – mbrid file name on boot device
- **hypervisor** (*string*) – hypervisor name
- **kernel** (*string*) – kernel name
- **initrd** (*string*) – initrd name

Implementation in specialized bootloader class required

setup_live_boot_images (*mbrid*, *lookup_path=None*)

Create bootloader images for ISO boot a live ISO image

Parameters

- **mbrid** (*string*) – mbrid file name on boot device
- **lookup_path** (*string*) – custom module lookup path

Implementation in specialized bootloader class required

setup_live_image_config (*mbrid*, *hypervisor*, *kernel*, *initrd*)

Create boot config file to boot live ISO image in EFI mode.

Parameters

- **mbrid** (*string*) – mbrid file name on boot device
- **hypervisor** (*string*) – hypervisor name
- **kernel** (*string*) – kernel name
- **initrd** (*string*) – initrd name

Implementation in specialized bootloader class required

setup_sysconfig_bootloader()

Create or update etc/sysconfig/bootloader by parameters required according to the bootloader setup

Implementation in specialized bootloader class required

write()

Write config data to config file.

Implementation in specialized bootloader class required

write_meta_data (*root_uuid=None, boot_options=""*)

Write bootloader setup meta data files

Parameters

- **root_uuid** (*string*) – root device UUID
- **boot_options** (*string*) – kernel options as string

Implementation in specialized bootloader class optional

10.4.3 kiwi.bootloader.config.grub2 Module

```
class kiwi.bootloader.config.grub2.BootLoaderConfigGrub2 (xml_state,  
                                                         root_dir,  
                                                         boot_dir=None,  
                                                         cus-  
                                                         tom_args=None)
```

Bases: *kiwi.bootloader.config.base.BootLoaderConfigBase*

grub2 bootloader configuration.**post_init** (*custom_args*)

grub2 post initialization method

Parameters **custom_args** (*dict*) – Contains grub config arguments

```
{'grub_directory_name': 'grub|grub2'}
```

setup_disk_boot_images (*boot_uuid, lookup_path=None*)

Create/Provide grub2 boot images and metadata

In order to boot from the disk grub2 modules, images and theme data needs to be created and provided at the correct place in the filesystem

Parameters

- **boot_uuid** (*string*) – boot device UUID
- **lookup_path** (*string*) – custom module lookup path

setup_disk_image_config (*boot_uuid=None, root_uuid=None, hyper-*
visor=None, kernel=None, initrd=None,
boot_options={})

Create grub2 config file to boot from disk using grub2-mkconfig

Parameters

- **boot_uuid**(*string*) – unused
- **root_uuid**(*string*) – unused
- **hypervisor**(*string*) – unused
- **kernel**(*string*) – unused
- **initrd**(*string*) – unused
- **boot_options**(*dict*) –

options dictionary that has to contain the root and boot device and optional volume configuration. KIWI has to mount the system prior to run grub2-mkconfig.

```
{
    'root_device': string,
    'boot_device': string,
    'efi_device': string,
    'system_volumes': volume_manager_instance.get_volumes()
}
```

setup_install_boot_images(*mbrid*, *lookup_path=None*)

Create/Provide grub2 boot images and metadata

In order to boot from the ISO grub2 modules, images and theme data needs to be created and provided at the correct place on the iso filesystem

Parameters

- **mbrid**(*string*) – mbrid file name on boot device
- **lookup_path**(*string*) – custom module lookup path

setup_install_image_config(*mbrid*, *hypervisor='xen.gz'*, *kernel='linux'*, *initrd='initrd'*)

Create grub2 config file to boot from an ISO install image

Parameters

- **mbrid**(*string*) – mbrid file name on boot device
- **hypervisor**(*string*) – hypervisor name
- **kernel**(*string*) – kernel name
- **initrd**(*string*) – initrd name

setup_live_boot_images(*mbrid*, *lookup_path=None*)

Create/Provide grub2 boot images and metadata

Calls setup_install_boot_images because no different action required

setup_live_image_config(*mbrid*, *hypervisor='xen.gz'*, *kernel='linux'*, *initrd='initrd'*)

Create grub2 config file to boot a live media ISO image

Parameters

- **mbrid** (*string*) – mbrid file name on boot device
- **hypervisor** (*string*) – hypervisor name
- **kernel** (*string*) – kernel name
- **initrd** (*string*) – initrd name

write ()

Write bootloader configuration

- writes grub.cfg template by KIWI if template system is used
- copies grub config file to alternative boot path for EFI systems in fallback mode
- creates an embedded fat efi image for EFI ISO boot

write_meta_data (*root_uuid=None, boot_options=""*)

Write bootloader setup meta data files

- cmdline arguments initialization
- etc/default/grub setup file
- etc/sysconfig/bootloader

Parameters

- **root_uuid** (*string*) – root device UUID
- **boot_options** (*string*) – kernel options as string
- **iso_boot** (*bool*) – indicate target is an ISO

10.4.4 kiwi.bootloader.config.isolinux Module

class `kiwi.bootloader.config.isolinux.BootLoaderConfigIsoLinux` (*xml_state, root_dir, boot_dir=None, custom_args=None*)

Bases: `kiwi.bootloader.config.base.BootLoaderConfigBase`

isolinux bootloader configuration.

post_init (*custom_args*)

isolinux post initialization method

Parameters **custom_args** (*dict*) – custom isolinux config arguments

setup_install_boot_images (*mbrid, lookup_path=None*)

Provide isolinux boot metadata

No extra boot images must be created for isolinux

Parameters

- **mbrid**(*string*) – unused
- **lookup_path**(*string*) – unused

setup_install_image_config(*mbrid*, *hypervisor*='xen.gz', *kernel*='linux', *initrd*='initrd')

Create isolinux.cfg in memory from a template suitable to boot from an ISO image in BIOS boot mode

Parameters

- **mbrid**(*string*) – mbrid file name on boot device
- **hypervisor**(*string*) – hypervisor name
- **kernel**(*string*) – kernel name
- **initrd**(*string*) – initrd name

setup_live_boot_images(*mbrid*, *lookup_path*=None)

Provide isolinux boot metadata

No extra boot images must be created for isolinux

Parameters

- **mbrid**(*string*) – unused
- **lookup_path**(*string*) – unused

setup_live_image_config(*mbrid*, *hypervisor*='xen.gz', *kernel*='linux', *initrd*='initrd')

Create isolinux.cfg in memory from a template suitable to boot a live system from an ISO image in BIOS boot mode

Parameters

- **mbrid**(*string*) – mbrid file name on boot device
- **hypervisor**(*string*) – hypervisor name
- **kernel**(*string*) – kernel name
- **initrd**(*string*) – initrd name

write()

Write isolinux.cfg and isolinux.msg file

10.4.5 kiwi.bootloader.config.zipl Module

```
class kiwi.bootloader.config.zipl.BootLoaderConfigZipl(xml_state,  
                                                       root_dir,  
                                                       boot_dir=None,  
                                                       custom_args=None)
```

Bases: *kiwi.bootloader.config.base.BootLoaderConfigBase*

zipl bootloader configuration.

post_init (*custom_args*)
zipl post initialization method

Parameters **custom_args** (*dict*) – Contains zipl config arguments

```
{'targetbase': 'device_name'}
```

setup_disk_boot_images (*boot_uuid*, *lookup_path=None*)

On s390 no bootloader images needs to be created

Thus this method does nothing

Parameters

- **boot_uuid** (*string*) – boot device UUID
- **lookup_path** (*string*) – custom module lookup path

setup_disk_image_config (*boot_uuid=None*, *root_uuid=None*, *hypervisor=None*, *kernel='image'*, *initrd='initrd'*, *boot_options={}*)

Create the zipl config in memory from a template suitable to boot from a disk image.

Parameters

- **boot_uuid** (*string*) – unused
- **root_uuid** (*string*) – unused
- **hypervisor** (*string*) – unused
- **kernel** (*string*) – kernel name
- **initrd** (*string*) – initrd name
- **boot_options** (*dict*) – unused

write ()
Write zipl config file

10.4.6 Module Contents

class `kiwi.bootloader.config.BootLoaderConfig`

Bases: `object`

BootLoaderConfig factory

Parameters

- **name** (*string*) – bootloader name
- **xml_state** (*object*) – instance of XMLState
- **root_dir** (*string*) – root directory path name
- **custom_args** (*dict*) – custom bootloader config arguments dictionary

10.5 kiwi.bootloader.install Package

10.5.1 Submodules

10.5.2 kiwi.bootloader.install.base Module

```
class kiwi.bootloader.install.base.BootLoaderInstallBase(root_dir,  
                                                         de-  
                                                         vice_provider,  
                                                         cus-  
                                                         tom_args=None)
```

Bases: `object`

Base class for bootloader installation on device

Parameters

- **root_dir** (*string*) – root directory path name
- **device_provider** (*object*) – instance of DeviceProvider
- **custom_args** (*dict*) – custom arguments dictionary

install()

Install bootloader on self.device

Implementation in specialized bootloader install class required

install_required()

Check if bootloader needs to be installed

Implementation in specialized bootloader install class required

post_init (*custom_args*)

Post initialization method

Store custom arguments by default

Parameters **custom_args** (*dict*) – custom bootloader arguments

10.5.3 kiwi.bootloader.install.grub2 Module

```
class kiwi.bootloader.install.grub2.BootLoaderInstallGrub2(root_dir,  
                                                            de-  
                                                            vice_provider,  
                                                            cus-  
                                                            tom_args=None)
```

Bases: `kiwi.bootloader.install.base.BootLoaderInstallBase`

grub2 bootloader installation

install()

Install bootloader on disk device

install_required()

Check if grub2 has to be installed

Take architecture and firmware setup into account to check if bootloader code in a boot record is required

Returns True or False

Return type bool

post_init (*custom_args*)

grub2 post initialization method

Parameters **custom_args** (*dict*) – Contains custom grub2 bootloader arguments

```
{
    'target_removable': bool,
    'system_volumes': list_of_volumes,
    'firmware': FirmWare_instance,
    'efi_device': string,
    'boot_device': string,
    'root_device': string
}
```

10.5.4 kiwi.bootloader.install.zipl Module

class kiwi.bootloader.install.zipl.**BootLoaderInstallZipl** (*root_dir*,
de-
vice_provider,
cus-
tom_args=None)

Bases: *kiwi.bootloader.install.base.BootLoaderInstallBase*

zipl bootloader installation

install()

Install bootloader on self.device

install_required()

Check if zipl has to be installed

Always required

Returns True

Return type bool

post_init (*custom_args*)

zipl post initialization method

Parameters **custom_args** (*dict*) – Contains custom zipl bootloader arguments


```
{'boot_device': string}
```

10.5.5 Module Contents

class `kiwi.bootloader.install.BootLoaderInstall`

Bases: `object`

BootLoaderInstall Factory

Parameters

- **name** (*string*) – bootloader name
- **root_dir** (*string*) – root directory path name
- **device_provider** (*object*) – instance of `DeviceProvider`
- **custom_args** (*dict*) – custom arguments dictionary

10.6 kiwi.bootloader Package

10.6.1 Module Contents

10.7 kiwi.bootloader.template Package

10.7.1 Submodules

10.7.2 kiwi.bootloader.template.grub2 Module

class `kiwi.bootloader.template.grub2.BootLoaderTemplateGrub2`

Bases: `object`

grub2 configuraton file templates

get_install_template (*failsafe=True, hybrid=True, terminal='gfxterm',
with_timeout=True*)

Bootloader configuration template for install media

Parameters

- **failsafe** (*bool*) – with failsafe true/false
- **hybrid** (*bool*) – with hybrid true/false
- **terminal** (*string*) – output terminal name

Returns instance of `Template`

Return type `Template`

get_iso_template (*failsafe=True, hybrid=True, terminal='gfxterm', check-iso=False*)

Bootloader configuration template for live ISO media

Parameters

- **failsafe** (*bool*) – with failsafe true/false
- **hybrid** (*bool*) – with hybrid true/false
- **terminal** (*string*) – output terminal name

Returns instance of Template

Return type Template

get_multiboot_install_template (*failsafe=True, terminal='gfxterm', with_timeout=True*)

Bootloader configuration template for install media with hypervisor, e.g Xen dom0

Parameters

- **failsafe** (*bool*) – with failsafe true/false
- **terminal** (*string*) – output terminal name

Returns instance of Template

Return type Template

get_multiboot_iso_template (*failsafe=True, terminal='gfxterm', check-iso=False*)

Bootloader configuration template for live ISO media with hypervisor, e.g Xen dom0

Parameters

- **failsafe** (*bool*) – with failsafe true/false
- **terminal** (*string*) – output terminal name

Returns instance of Template

Return type Template

10.7.3 kiwi.bootloader.template.isolinux Module

class kiwi.bootloader.template.isolinux.BootLoaderTemplateIsoLinux

Bases: *object*

isolinux configuraton file templates

get_install_message_template ()

Bootloader template for text message file in install mode. isolinux displays this as menu if no graphics mode can be initialized

Returns instance of Template

Return type Template

get_install_template (*failsafe=True, with_theme=True, terminal=None, with_timeout=True*)

Bootloader configuration template for install media

Parameters

- **failsafe** (*bool*) – with failsafe true/false
- **with_theme** (*bool*) – with graphics theme true/false

Returns instance of `Template`

Return type `Template`

get_message_template ()

Bootloader template for text message file. `isolinux` displays this as menu if no graphics mode can be initialized

Returns instance of `Template`

Return type `Template`

get_multiboot_install_template (*failsafe=True, with_theme=True, terminal=None, with_timeout=True*)

Bootloader configuration template for install media with hypervisor, e.g Xen dom0

Parameters

- **failsafe** (*bool*) – with failsafe true/false
- **with_theme** (*bool*) – with graphics theme true/false

Returns instance of `Template`

Return type `Template`

get_multiboot_template (*failsafe=True, with_theme=True, terminal=None, checkiso=False*)

Bootloader configuration template for live media with hypervisor, e.g Xen dom0

Parameters

- **failsafe** (*bool*) – with failsafe true/false
- **with_theme** (*bool*) – with graphics theme true/false

Returns instance of `Template`

Return type `Template`

get_template (*failsafe=True, with_theme=True, terminal=None, checkiso=False*)

Bootloader configuration template for live media

Parameters

- **failsafe** (*bool*) – with failsafe true/false
- **with_theme** (*bool*) – with graphics theme true/false

Returns instance of `Template`

Return type Template

10.7.4 kiwi.bootloader.template.zipl Module

class kiwi.bootloader.template.zipl.BootLoaderTemplateZipl

Bases: `object`

zipl configuraton file templates

get_template (*failsafe=True, targettype=None*)

Bootloader configuration template for disk boot

Parameters **failsafe** (*bool*) – with failsafe true/false

Returns instance of Template

Return type Template

10.7.5 Module Contents

10.8 kiwi.boot Package

10.8.1 Module Contents

10.9 kiwi.builder Package

10.9.1 Submodules

10.9.2 kiwi.builder.archive Module

class kiwi.builder.archive.ArchiveBuilder (*xml_state, target_dir, root_dir, custom_args=None*)

Bases: `object`

Root archive image builder

Parameters

- **xml_state** (*object*) – Instance of XMLState
- **target_dir** (*str*) – target directory path name
- **root_dir** (*str*) – root directory path name
- **custom_args** (*dict*) – Custom processing arguments defined as hash keys: * **xz_options**: string of XZ compression parameters

create()

Create a root archive tarball

Build a simple XZ compressed root tarball from the image root tree

Image types which triggers this builder are:

- `image="tbz"`

Returns result

Return type instance of `Result`

10.9.3 kiwi.builder.container Module

```
class kiwi.builder.container.ContainerBuilder(xml_state,  
                                              target_dir,  
                                              root_dir,      cus-  
                                              tom_args=None)
```

Bases: `object`

Container image builder

Parameters

- **xml_state** (*object*) – Instance of `XMLState`
- **target_dir** (*str*) – target directory path name
- **root_dir** (*str*) – root directory path name
- **custom_args** (*dict*) – Custom processing arguments defined as hash keys: * `xz_options`: string of XZ compression parameters

create()

Builds a container image which is usually a data archive including container specific metadata.

Image types which triggers this builder are:

- `image="docker"`
- `image="oci"`
- `image="appx"`

Returns result

Return type instance of `Result`

10.9.4 kiwi.builder.disk Module

class `kiwi.builder.disk.DiskBuilder` (*xml_state*, *target_dir*, *root_dir*,
custom_args=None)

Bases: `object`

Disk image builder

Parameters

- **xml_state** (*object*) – Instance of `XMLState`
- **target_dir** (*str*) – Target directory path name
- **root_dir** (*str*) – Root directory path name
- **custom_args** (*dict*) – Custom processing arguments defined as hash keys: * **signing_keys**: list of package signing keys * **xz_options**: string of XZ compression parameters

append_unpartitioned_space ()

Extends the raw disk if an unpartitioned area is specified

create ()

Build a bootable disk image and optional installation image The installation image is a bootable hybrid ISO image which embeds the disk image and an image installer

Image types which triggers this builder are:

- **image**="oem"
- **image**="vmx"

Returns `result`

Return type instance of `Result`

create_disk ()

Build a bootable raw disk image

Raises

- **KiwiInstallMediaError** – if install media is required and image type is not oem
- **KiwiVolumeManagerSetupError** – root overlay at the same time volumes are defined is not supported

Returns `result`

Return type instance of `Result`

create_disk_format (*result_instance*)

Create a bootable disk format from a previously created raw disk image

Parameters **result_instance** (*object*) – instance of `Result`

Returns updated `result_instance`

Return type instance of `Result`

create_install_media (*result_instance*)

Build an installation image. The installation image is a bootable hybrid ISO image which embeds the raw disk image and an image installer

Parameters **result_instance** (*object*) – instance of `Result`

Returns updated `result_instance` with installation media

Return type instance of `Result`

10.9.5 kiwi.builder.filesystem Module

```
class kiwi.builder.filesystem.FileSystemBuilder(xml_state,  
                                              target_dir,  
                                              root_dir)
```

Bases: `object`

Filesystem image builder

Parameters

- **label** (*str*) – filesystem label
- **root_uuid** (*str*) – UUID of the created filesystem (on block device only)
- **root_dir** (*str*) – root directory path name
- **target_dir** (*str*) – target directory path name
- **requested_image_type** (*str*) – configured image type
- **requested_filesystem** (*str*) – requested filesystem name
- **system_setup** (*object*) – instance of `SystemSetup`
- **filename** (*str*) – file name of the filesystem image
- **blocksize** (*int*) – configured disk blocksize
- **filesystem_setup** (*object*) – instance of `FileSystemSetup`
- **filesystems_no_device_node** (*object*) – List of filesystems which are created from a data tree and do not require a block device e.g loop
- **filesystem_custom_parameters** (*dict*) – Configured custom filesystem mount and creation arguments
- **result** (*object*) – instance of `Result`

create ()

Build a mountable filesystem image

Image types which triggers this builder are:

- image="ext2"
- image="ext3"
- image="ext4"
- image="btrfs"
- image="xfs"

Returns result

Return type instance of Result

10.9.6 kiwi.builder.install Module

```
class kiwi.builder.install.InstallImageBuilder(xml_state,  
                                              root_dir,  
                                              target_dir,  
                                              boot_image_task,  
                                              cus-  
                                              tom_args=None)
```

Bases: `object`

Installation image builder

Parameters

- **xml_state** (*object*) – instance of XMLState
- **root_dir** (*str*) – system image root directory
- **target_dir** (*str*) – target directory path name
- **boot_image_task** (*object*) – instance of BootImage
- **custom_args** (*dict*) – Custom processing arguments defined as hash keys: * xz_options: string of XZ compression parameters

create_install_iso()

Create an install ISO from the disk_image as hybrid ISO bootable via legacy BIOS, EFI and as disk from Stick

Image types which triggers this builder are:

- installiso="true|false"
- installstick="true|false"

create_install_pxe_archive()

Create an oem install tar archive suitable for installing a disk image via the network using the PXE boot protocol. The archive contains:

- The raw system image xz compressed
- The raw system image checksum metadata file

- The append file template for the boot server
- The system image initrd for kexec
- The install initrd
- The kernel

Image types which triggers this builder are:

- `installpxe="true|false"`

10.9.7 `kiwi.builder.live` Module

```
class kiwi.builder.live.LiveImageBuilder(xml_state, target_dir, root_dir, custom_args=None)
```

Bases: `object`

Live image builder

Parameters

- **`xml_state`** (*object*) – instance of `XMLState`
- **`target_dir`** (*str*) – target directory path name
- **`root_dir`** (*str*) – root directory path name
- **`custom_args`** (*dict*) – Custom processing arguments

`create()`

Build a bootable hybrid live ISO image

Image types which triggers this builder are:

- `image="iso"`

Raises `KiwiLiveBootImageError` – if no kernel or hipervisor is found in boot image tree

Returns result

Return type instance of `Result`

10.9.8 `kiwi.builder.pxe` Module

```
class kiwi.builder.pxe.PxeBuilder(xml_state, target_dir, root_dir, custom_args=None)
```

Bases: `object`

Filesystem based PXE image builder.

Parameters

- **`xml_state`** (*object*) – instance of `XMLState`

- **target_dir** (*str*) – target directory path name
- **root_dir** (*str*) – system image root directory
- **custom_args** (*dict*) – Custom processing arguments defined as hash keys: * signing_keys: list of package signing keys * xz_options: string of XZ compression parameters

create ()

Build a pxe image set consisting out of a boot image(initrd) plus its appropriate kernel files and the root filesystem image with a checksum. The result can be used within the kiwi PXE boot infrastructure

Image types which triggers this builder are:

- image="pxe"

Raises *KiwiPxeBootImageError* – if no kernel or hipervisor is found in boot image tree

Returns result

Return type instance of *Result*

10.9.9 Module Contents

class *kiwi.builder.ImageBuilder*

Bases: *object*

image builder factory

10.10 kiwi.container Package

10.10.1 Submodules

10.10.2 *kiwi.container.oci* Module

class *kiwi.container.oci.ContainerImageOCI* (*root_dir*, *transport*,
custom_args=None)

Bases: *object*

Create oci container from a root directory

Parameters

- **root_dir** (*string*) – root directory path name
- **custom_args** (*dict*) –

Custom processing arguments defined as hash keys:

Example


```
{
  'container_name': 'name',
  'container_tag': '1.0',
  'additional_tags': ['current', 'foobar'],
  'entry_command': ['/bin/bash', '-x'],
  'entry_subcommand': ['ls', '-l'],
  'maintainer': 'tux',
  'user': 'root',
  'workingdir': '/root',
  'expose_ports': ['80', '42'],
  'volumes': ['/var/log', '/tmp'],
  'environment': {'PATH': '/bin'},
  'labels': {'name': 'value'},
  'history': {
    'created_by': 'some explanation here',
    'comment': 'some comment here',
    'author': 'tux'
  }
}
```

create (*filename*, *base_image*)

Create compressed oci system container tar archive

Parameters

- **filename** (*string*) – archive file name
- **base_image** (*string*) – archive used as a base image

10.10.3 Module Contents

class `kiwi.container.ContainerImage`

Bases: `object`

Container Image factory

Parameters

- **name** (*string*) – container system name
- **root_dir** (*string*) – root directory path name
- **custom_args** (*dict*) – custom arguments

10.11 kiwi.container.setup Package

10.11.1 Submodules

10.11.2 kiwi.container.setup.base Module

```
class kiwi.container.setup.base.ContainerSetupBase(root_dir,  
                                                    cus-  
                                                    tom_args=None)
```

Bases: `object`

Base class for setting up the root system to create a container image from for e.g docker. The methods here are generic to linux systems following the FHS standard and modern enough e.g based on systemd

Attributes

- **root_dir** root directory path name
- **custom_args** dict of custom arguments

deactivate_bootloader_setup()

Container bootloader setup

Tell the system there is no bootloader configuration it needs to care for. A container does not boot

deactivate_root_filesystem_check()

Container filesystem check setup

The root filesystem of a container could be an overlay or a mapped device. In any case it should not be checked for consistency as this is should be done by the container infrastructure

deactivate_systemd_service(*name*)

Container system services setup

Init systems among others also controls services which starts at boot time. A container does not really boot. Thus some services needs to be deactivated

Parameters **name** (*string*) – systemd service name

get_container_name()

Container name

Returns name

Return type `str`

post_init(*custom_args*)

Post initialization method

Implementation in specialized container setup class

Parameters **custom_args** (*list*) – unused

setup()

Setup container metadata

Implementation in specialized bootloader class required

setup_root_console()

Container console setup

/dev/console should be allowed to login by root

setup_static_device_nodes()

Container device node setup

Without subsystems like udev running in a container it is required to provide a set of device nodes to let the system in the container function correctly. This is done by syncing the host system nodes to the container. That this will also create device nodes which are not necessarily present in the container later is a known limitation of this method and considered harmless

10.11.3 `kiwi.container.setup.docker` Module

```
class kiwi.container.setup.docker.ContainerSetupDocker(root_dir,  
                                                    cus-  
                                                    tom_args=None)
```

Bases: `kiwi.container.setup.oci.ContainerSetupOCI`

Docker container setup

10.11.4 Module Contents

```
class kiwi.container.setup.ContainerSetup
```

Bases: `object`

container setup factory

10.12 `kiwi.filesystem` Package

10.12.1 Submodules

10.12.2 `kiwi.filesystem.base` Module

```
class kiwi.filesystem.base.FileSystemBase(device_provider,  
                                          root_dir=None,      cus-  
                                          tom_args=None)
```

Bases: `object`

Implements base class for filesystem interface

Parameters

- **device_provider** (*object*) – Instance of a class based on `DeviceProvider` required for filesystems which needs a block device for creation. In most cases the `DeviceProvider` is a `LoopDevice`
- **root_dir** (*string*) – root directory path name
- **custom_args** (*dict*) – custom filesystem arguments

create_on_device (*label=None*)

Create filesystem on block device

Implement in specialized filesystem class for filesystems which requires a block device for creation, e.g ext4.

Parameters **label** (*string*) – label name

create_on_file (*filename, label=None, exclude=None*)

Create filesystem from root data tree

Implement in specialized filesystem class for filesystems which requires a data tree for creation, e.g squashfs.

Parameters

- **filename** (*string*) – result file path name
- **label** (*string*) – label name
- **exclude** (*list*) – list of exclude dirs/files

post_init (*custom_args*)

Post initialization method

Store dictionary of custom arguments if not empty. This overrides the default custom argument hash

Parameters **custom_args** (*dict*) – custom arguments

```
{
    'create_options': ['option'],
    'mount_options': ['option'],
    'meta_data': {
        'key': 'value'
    }
}
```

sync_data (*exclude=None*)

Copy root data tree into filesystem

Parameters **exclude** (*list*) – list of exclude dirs/files

10.12.3 `kiwi.filesystem.btrfs` Module

```
class kiwi.filesystem.btrfs.FileSystemBtrfs (device_provider,  
                                           root_dir=None,  
                                           custom_args=None)
```

Bases: `kiwi.filesystem.base.FileSystemBase`

Implements creation of btrfs filesystem

```
create_on_device (label=None)
```

Create btrfs filesystem on block device

Parameters `label` (*string*) – label name

10.12.4 `kiwi.filesystem.clicfs` Module

```
class kiwi.filesystem.clicfs.FileSystemClicFs (device_provider,  
                                              root_dir=None,  
                                              cus-  
                                              tom_args=None)
```

Bases: `kiwi.filesystem.base.FileSystemBase`

Implements creation of clicfs filesystem

```
create_on_file (filename, label=None, exclude=None)
```

Create clicfs filesystem from data tree

There is no label which could be set for clicfs thus this parameter is not used

There is no option to exclude data from clicfs thus this parameter is not used

Parameters

- **filename** (*string*) – result file path name
- **label** (*string*) – unused
- **exclude** (*list*) – unused

```
post_init (custom_args=None)
```

Post initialization method

Initialize temporary container_dir directory to store clicfs embedded filesystem

Parameters `custom_args` (*dict*) – unused

10.12.5 `kiwi.filesystem.ext2` Module

```
class kiwi.filesystem.ext2.FileSystemExt2 (device_provider,  
                                           root_dir=None,      cus-  
                                           tom_args=None)
```

Bases: `kiwi.filesystem.base.FileSystemBase`

Implements creation of ext2 filesystem

create_on_device (*label=None*)

Create ext2 filesystem on block device

Parameters **label** (*string*) – label name

10.12.6 `kiwi.filesystem.ext3` Module

```
class kiwi.filesystem.ext3.FileSystemExt3 (device_provider,  
                                           root_dir=None,      cus-  
                                           tom_args=None)
```

Bases: `kiwi.filesystem.base.FileSystemBase`

Implements creation of ext3 filesystem

create_on_device (*label=None*)

Create ext3 filesystem on block device

Parameters **label** (*string*) – label name

10.12.7 `kiwi.filesystem.ext4` Module

```
class kiwi.filesystem.ext4.FileSystemExt4 (device_provider,  
                                           root_dir=None,      cus-  
                                           tom_args=None)
```

Bases: `kiwi.filesystem.base.FileSystemBase`

Implements creation of ext4 filesystem

create_on_device (*label=None*)

Create ext4 filesystem on block device

Parameters **label** (*string*) – label name

10.12.8 `kiwi.filesystem.fat16` Module

```
class kiwi.filesystem.fat16.FileSystemFat16 (device_provider,  
                                           root_dir=None,  
                                           custom_args=None)
```

Bases: `kiwi.filesystem.base.FileSystemBase`

Implements creation of fat16 filesystem

create_on_device (*label=None*)
Create fat16 filesystem on block device

Parameters **label** (*string*) – label name

10.12.9 `kiwi.filesystem.fat32` Module

```
class kiwi.filesystem.fat32.FileSystemFat32 (device_provider,  
                                           root_dir=None,  
                                           custom_args=None)
```

Bases: `kiwi.filesystem.base.FileSystemBase`

Implements creation of fat16 filesystem

create_on_device (*label=None*)
Create fat32 filesystem on block device

Parameters **label** (*string*) – label name

10.12.10 `kiwi.filesystem.isofs` Module

```
class kiwi.filesystem.isofs.FileSystemIsoFs (device_provider,  
                                           root_dir=None,  
                                           custom_args=None)
```

Bases: `kiwi.filesystem.base.FileSystemBase`

Implements creation of iso filesystem

create_on_file (*filename, label=None, exclude=None*)
Create iso filesystem from data tree

There is no label which could be set for iso filesystem thus this parameter is not used

Parameters

- **filename** (*string*) – result file path name
- **label** (*string*) – unused
- **exclude** (*string*) – unused

10.12.11 `kiwi.filesystem.setup` Module

class `kiwi.filesystem.setup.FileSystemSetup` (*xml_state*, *root_dir*)

Bases: `object`

Implement filesystem setup methods

Methods from this class provides information from the root directory required before building a filesystem image

Parameters

- **xml_state** (*object*) – Instance of XMLState
- **root_dir** (*string*) – root directory path

get_size_mbytes (*filesystem=None*)

Precalculate the requires size in mbytes to store all data from the root directory in the requested filesystem. Return the configured value if present, if not return the calculated result

Parameters **filesystem** (*string*) – name

Returns mbytes

Return type `int`

10.12.12 `kiwi.filesystem.squashfs` Module

class `kiwi.filesystem.squashfs.FileSystemSquashFs` (*device_provider*,
root_dir=None,
custom_args=None)

Bases: `kiwi.filesystem.base.FileSystemBase`

Implements creation of squashfs filesystem

create_on_file (*filename*, *label=None*, *exclude=None*)

Create squashfs filesystem from data tree

There is no label which could be set for squashfs thus this parameter is not used

Parameters

- **filename** (*string*) – result file path name
- **label** (*string*) – unused
- **exclude** (*list*) – list of exclude dirs/files

10.12.13 kiwi.filesystem.xfs Module

```
class kiwi.filesystem.xfs.FileSystemXfs (device_provider,  
                                         root_dir=None,           cus-  
                                         tom_args=None)
```

Bases: *kiwi.filesystem.base.FileSystemBase*

Implements creation of xfs filesystem

create_on_device (*label=None*)

Create xfs filesystem on block device

Parameters **label** (*string*) – label name

10.12.14 Module Contents

```
class kiwi.filesystem.FileSystem
```

Bases: *object*

FileSystem factory

Parameters

- **name** (*string*) – filesystem name
- **device_provider** (*object*) – Instance of DeviceProvider
- **root_dir** (*string*) – root directory path name
- **custom_args** (*dict*) – dict of custom filesystem arguments

10.13 kiwi.iso_tools Package

10.13.1 Submodules

10.13.2 kiwi.iso_tools.base Module

```
class kiwi.iso_tools.base.IsoToolsBase (source_dir)
```

Bases: *object*

Base Class for Parameter API for iso creation tools

Parameters

- **source_dir** (*string*) – data source dir, usually root_dir
- **boot_path** (*str*) – architecture specific boot path on the ISO
- **iso_parameters** (*str*) – list of ISO creation parameters
- **iso_loaders** (*str*) – list of ISO loaders to embed

add_efi_loader_parameters ()

Add ISO creation parameters to embed the EFI loader

Implementation in specialized tool class

create_iso (*filename*, *hidden_files=None*)

Create iso file

Implementation in specialized tool class

Parameters

- **filename** (*str*) – unused
- **hidden_files** (*list*) – unused

get_tool_name ()

Return caller name for iso creation tool

Implementation in specialized tool class

Returns tool name

Return type *str*

has_iso_hybrid_capability ()

Indicate if the iso tool has the capability to embed a partition table into the iso such that it can be used as both; an iso and a disk

Implementation in specialized tool class

init_iso_creation_parameters (*custom_args=None*)

Create a set of standard parameters for the main isolinux loader

Implementation in specialized tool class

Parameters **custom_args** (*list*) – unused

list_iso (*isofile*)

List contents of an ISO image

Parameters **isofile** (*str*) – unused

static setup_media_loader_directory (*lookup_path*, *media_path*,
boot_theme)

10.13.3 kiwi.iso_tools.cdrtools Module

class `kiwi.iso_tools.cdrtools.IsoToolsCdrTools` (*source_dir*)

Bases: `kiwi.iso_tools.base.IsoToolsBase`

cdrkit/cdrtools wrapper class

Implementation of Parameter API for iso creation tools using the cdrkit/cdrtools projects. Addressed here are the option compatible tools mkisofs and genisoimage

add_efi_loader_parameters()

Add ISO creation parameters to embed the EFI loader

In order to boot the ISO from EFI, the EFI binary is added as alternative loader to the ISO creation parameter list. The EFI binary must be included into a fat filesystem in order to become recognized by the firmware. For details about this file refer to `_create_embedded_fat_efi_image()` from `bootloader/config/grub2.py`

create_iso(filename, hidden_files=None)

Creates the iso file with the given filename using cdrtools

Parameters

- **filename** (*str*) – output filename
- **hidden_files** (*list*) – list of hidden files

get_tool_name()

There are tools by J.Schilling and tools from the community Depending on what is installed a decision needs to be made. mkisofs is preferred over genisoimage

Raises *KiwiIsoToolError* – if no iso creation tool is found

Returns tool name

Return type *str*

has_iso_hybrid_capability()

Indicate if the iso tool has the capability to embed a partition table into the iso such that it can be used as both; an iso and a disk

Returns True or False

Return type *bool*

init_iso_creation_parameters(custom_args=None)

Create a set of standard parameters

Parameters **custom_args** (*list*) – custom ISO creation args

list_iso(isofile)

List contents of an ISO image

Parameters **isofile** (*str*) – path to the ISO file

Returns formatted isoinfo result

Return type *dict*

10.13.4 kiwi.iso_tools.iso Module

class `kiwi.iso_tools.iso.Iso` (*source_dir*)

Bases: `object`

Implements helper methods around the creation of ISO filesystems

Parameters

- **header_id** (*str*) – static identifier string for self written headers
- **header_end_name** (*str*) – file name to store the header_id to
- **header_end_file** (*str*) – full file path for the header_end_name file
- **boot_path** (*str*) – architecture specific boot path on the ISO

create_header_end_block (*isofile*)

Find offset address of file containing the header_id and replace it by a list of 2k blocks in range 0 - offset + 1 This is the required preparation to support hybrid ISO images, meaning to let isohybrid work correctly

Parameters **isofile** (*string*) – path to the ISO file

Raises `KiwiIsoLoaderError` – if the header_id file is not found

Returns 512 byte blocks offset address

Return type `int`

create_header_end_marker ()

Prepare iso file to become a hybrid iso image.

To do this the offset address of the end of the first iso block is required. To lookup this address a reference(marker) file named 'header_end' is created and will show up as last file in the block.

static create_hybrid (*offset, mbrid, isofile, efi_mode=False*)

Create hybrid ISO

A hybrid ISO embeds both, an isolinux signature as well as a disk signature. kiwi always adds an msdos and a GPT table for the disk signatures

Parameters

- **offset** (*str*) – hex offset
- **mbrid** (*str*) – boot record id
- **isofile** (*str*) – path to the ISO file
- **efi_mode** (*bool*) – sets the iso to support efi firmware or not

static fix_boot_catalog (*isofile*)

Fixup inconsistencies in boot catalog

Make sure all catalog entries are in correct order and provide complete metadata information e.g catalog name

Parameters **isofile** (*str*) – path to the ISO file

static relocate_boot_catalog (*isofile*)

Move ISO boot catalog to the standardized place

Check location of the boot catalog and move it to the place where all BIOS and firmware implementations expects it

Parameters **isofile** (*str*) – path to the ISO file

static set_media_tag (*isofile*)

Include checksum tag in the ISO so it can be verified with the mediacheck program.

Parameters **isofile** (*str*) – path to the ISO file

setup_isolinux_boot_path ()

Write the base boot path into the isolinux loader binary

Raises **KiwiIsoLoaderError** – if loader/isolinux.bin is not found

10.13.5 Module Contents

class `kiwi.iso_tools.IsoTools`

Bases: `object`

IsoTools factory

10.14 kiwi.package_manager Package

10.14.1 Submodules

10.14.2 `kiwi.package_manager.base` Module

class `kiwi.package_manager.base.PackageManagerBase` (*repository*,

cus-

tom_args=None)

Bases: `object`

Implements base class for installation/deletion of packages and collections using a package manager

Parameters

- **repository** (*object*) – instance of `Repository`
- **root_dir** (*str*) – root directory path name
- **package_requests** (*list*) – list of packages to install or delete
- **collection_requests** (*list*) – list of collections to install
- **product_requests** (*list*) – list of products to install

cleanup_requests ()

Cleanup request queues

database_consistent ()

OBSOLETE: Will be removed 2019-06-05

dump_reload_package_database (*version=45*)

OBSOLETE: Will be removed 2019-06-05

has_failed (*returncode*)

Evaluate given result return code

Any returncode != 0 is considered an error unless overwritten in specialized package manager class

Parameters **returncode** (*int*) – return code number

Returns True|False

Return type boolean

match_package_deleted (*package_list, log_line*)

Match expression to indicate a package has been deleted

Implementation in specialized package manager class

Parameters

- **package_list** (*list*) – unused
- **log_line** (*str*) – unused

match_package_installed (*package_list, log_line*)

Match expression to indicate a package has been installed

Implementation in specialized package manager class

Parameters

- **package_list** (*list*) – unused
- **log_line** (*str*) – unused

post_init (*custom_args=None*)

Post initialization method

Implementation in specialized package manager class

Parameters **custom_args** (*list*) – unused

post_process_install_requests_bootstrap ()

Process extra code required after bootstrapping

Implementation in specialized package manager class

process_delete_requests (*force=False*)

Process package delete requests (chroot)

Implementation in specialized package manager class

Parameters **force** (*bool*) – unused

process_install_requests ()

Process package install requests for image phase (chroot)

Implementation in specialized package manager class

process_install_requests_bootstrap ()

Process package install requests for bootstrap phase (no chroot)

Implementation in specialized package manager class

process_only_required ()

Setup package processing only for required packages

Implementation in specialized package manager class

process_plus_recommended ()

Setup package processing to also include recommended dependencies

Implementation in specialized package manager class

request_collection (*name*)

Queue a package collection

Implementation in specialized package manager class

Parameters **name** (*str*) – unused

request_package (*name*)

Queue a package request

Implementation in specialized package manager class

Parameters **name** (*str*) – unused

request_package_exclusion (*name*)

Queue a package exclusion(skip) request

Implementation in specialized package manager class

Parameters **name** (*str*) – unused

request_package_lock (*name*)

Queue a package exclusion(skip) request

OBSOLETE: Will be removed 2019-06-05

Kept for API compatibility Method calls: request_package_exclusion

request_product (*name*)

Queue a product request

Implementation in specialized package manager class

Parameters **name** (*str*) – unused

update ()

Process package update requests (chroot)

Implementation in specialized package manager class

10.14.3 `kiwi.package_manager.dnf` Module

class `kiwi.package_manager.dnf.PackageManagerDnf` (*repository*,
custom_args=None)

Bases: `kiwi.package_manager.base.PackageManagerBase`

Implements base class for installation/deletion of packages and collections using dnf

Parameters

- **dnf_args** (*doct*) – dnf arguments from repository runtime configuration
- **command_env** (*dict*) – dnf command environment from repository runtime configuration

match_package_deleted (*package_name*, *dnf_output*)

Match expression to indicate a package has been deleted

Parameters

- **package_list** (*list*) – list of all packages
- **log_line** (*str*) – dnf status line

Returns match or None if there isn't any match

Return type match object, None

match_package_installed (*package_name*, *dnf_output*)

Match expression to indicate a package has been installed

This match for the package to be installed in the output of the dnf command is not 100% accurate. There might be false positives due to sub package names starting with the same base package name

Parameters

- **package_list** (*list*) – list of all packages
- **log_line** (*str*) – dnf status line

Returns match or None if there isn't any match

Return type match object, None

post_init (*custom_args=None*)

Post initialization method

Parameters **custom_args** (*list*) – custom dnf arguments

post_process_install_requests_bootstrap()

Move the rpm database to the place as it is expected by the rpm package installed during bootstrap phase

process_delete_requests (*force=False*)

Process package delete requests (chroot)

Parameters **force** (*bool*) – force deletion: true/false

Raises *KiwiRequestError* – if none of the packages to delete is installed.

Returns process results in command type

Return type namedtuple

process_install_requests()

Process package install requests for image phase (chroot)

Returns process results in command type

Return type namedtuple

process_install_requests_bootstrap()

Process package install requests for bootstrap phase (no chroot)

Returns process results in command type

Return type namedtuple

process_only_required()

Setup package processing only for required packages

process_plus_recommended()

Setup package processing to also include recommended dependencies.

request_collection (*name*)

Queue a collection request

Parameters **name** (*str*) – dnf group name

request_package (*name*)

Queue a package request

Parameters **name** (*str*) – package name

request_package_exclusion (*name*)

Queue a package exclusion(skip) request

Parameters **name** (*str*) – package name

request_product (*name*)

Queue a product request

There is no product definition in the fedora repo data

Parameters **name** (*str*) – unused

update ()

Process package update requests (chroot)

Returns process results in command type

Return type namedtuple

10.14.4 kiwi.package_manager.zypper Module

class `kiwi.package_manager.zypper.PackageManagerZypper` (*repository*,
cus-
tom_args=None)

Bases: `kiwi.package_manager.base.PackageManagerBase`

Implements base class for installation/deletion of packages and collections using zypper

Parameters

- **zypper_args** (*list*) – zypper arguments from repository runtime configuration
- **command_env** (*dict*) – zypper command environment from repository runtime configuration

has_failed (*returncode*)

Evaluate given result return code

In zypper any return code == 0 or >= 100 is considered success. Any return code different from 0 and < 100 is treated as an error we care for. Return codes >= 100 indicates an issue like ‘new kernel needs reboot of the system’ or similar which we don’t care in the scope of image building

Parameters **returncode** (*int*) – return code number

Returns True/False

Return type boolean

match_package_deleted (*package_name*, *zypper_output*)

Match expression to indicate a package has been deleted

Parameters

- **package_list** (*list*) – list of all packages
- **log_line** (*str*) – zypper status line

Returns match or None if there isn’t any match

Return type match object, `None`

match_package_installed (*package_name*, *zypper_output*)

Match expression to indicate a package has been installed

This match for the package to be installed in the output of the zypper command is not 100% accurate. There might be false positives due to sub package names starting with the same base package name

Parameters

- **package_list** (*list*) – list of all packages
- **log_line** (*str*) – zypper status line

Returns match or None if there isn't any match

Return type match object, *None*

post_init (*custom_args=None*)

Post initialization method

Store custom zypper arguments

Parameters **custom_args** (*list*) – custom zypper arguments

post_process_install_requests_bootstrap ()

Move the rpm database to the place as it is expected by the rpm package installed during bootstrap phase

process_delete_requests (*force=False*)

Process package delete requests (chroot)

Parameters **force** (*bool*) – force deletion: true/false

Raises *KiwiRequestError* – if none of the packages to delete is installed

Returns process results in command type

Return type namedtuple

process_install_requests ()

Process package install requests for image phase (chroot)

Returns process results in command type

Return type namedtuple

process_install_requests_bootstrap ()

Process package install requests for bootstrap phase (no chroot)

Returns process results in command type

Return type namedtuple

process_only_required ()

Setup package processing only for required packages

process_plus_recommended ()

Setup package processing to also include recommended dependencies.

request_collection (*name*)

Queue a collection request

Parameters **name** (*str*) – zypper pattern name

request_package (*name*)

Queue a package request

Parameters **name** (*str*) – package name

request_package_exclusion (*name*)

Queue a package exclusion(skip) request

Parameters **name** (*str*) – package name

request_product (*name*)

Queue a product request

Parameters **name** (*str*) – zypper product name

update ()

Process package update requests (chroot)

Returns process results in command type

Return type namedtuple

10.14.5 Module Contents

class `kiwi.package_manager.PackageManager`

Bases: `object`

Package manager factory

Parameters

- **repository** (*object*) – instance of `Repository`
- **package_manager** (*str*) – package manager name
- **custom_args** (*list*) – custom package manager arguments list

Raises `KiwiPackageManagerSetupError` – if the requested package manager type is not supported

Returns package manager

Return type `PackageManagerBase` subclass

10.15 kiwi.partitionner Package

10.15.1 Submodules

10.15.2 kiwi.partitionner.base Module

class `kiwi.partitionner.base.PartitionerBase` (*disk_provider*,
start_sector=None)

Bases: `object`

Base class for partitioners

Parameters

- **disk_provider** (*object*) – Instance of DeviceProvider
- **start_sector** (*int*) – sector number

create (*name*, *mbsize*, *type_name*, *flags=None*)

Create partition

Implementation in specialized partitioner class

Parameters

- **name** (*string*) – unused
- **mbsize** (*int*) – unused
- **type_name** (*string*) – unused
- **flags** (*list*) – unused

get_id ()

Current partition number

Zero indicates no partition has been created so far

Returns partition number

Return type `int`

post_init ()

Post initialization method

Implementation in specialized partitioner class

resize_table (*entries=None*)

Resize partition table

Parameters **entries** (*int*) – unused

set_flag (*partition_id*, *flag_name*)

Set partition flag

Implementation in specialized partitioner class

Parameters

- **partition_id** (*int*) – unused

- **flag_name** (*string*) – unused

set_hybrid_mbr ()

Turn partition table into hybrid table if supported

Implementation in specialized partitioner class

set_mbr ()

Turn partition table into MBR (msdos table)

Implementation in specialized partitioner class

10.15.3 kiwi.partitionner.dasd Module

class `kiwi.partitionner.dasd.PartitionerDasd` (*disk_provider*,
start_sector=None)

Bases: `kiwi.partitionner.base.PartitionerBase`

Implements DASD partition setup

create (*name*, *mbsize*, *type_name*, *flags*=None)

Create DASD partition

Parameters

- **name** (*string*) – partition name

- **mbsize** (*int*) – partition size

- **type_name** (*string*) – unused

- **flags** (*list*) – unused

post_init ()

Post initialization method

Setup fdasd partition type/flag map

resize_table (*entries*=None)

Resize partition table

Nothing to be done here for DASD devices

Parameters **entries** (*int*) – unused

10.15.4 `kiwi.partitionner.gpt` Module

class `kiwi.partitionner.gpt.PartitionerGpt` (*disk_provider*,
start_sector=None)

Bases: `kiwi.partitionner.base.PartitionerBase`

Implements GPT partition setup

create (*name*, *mbsize*, *type_name*, *flags=None*)

Create GPT partition

Parameters

- **name** (*string*) – partition name
- **mbsize** (*int*) – partition size
- **type_name** (*string*) – partition type
- **flags** (*list*) – additional flags

post_init ()

Post initialization method

Setup gdisk partition type/flag map

resize_table (*entries=128*)

Resize partition table

Parameters **entries** (*int*) – number of default entries

set_flag (*partition_id*, *flag_name*)

Set GPT partition flag

Parameters

- **partition_id** (*int*) – partition number
- **flag_name** (*string*) – name from flag map

set_hybrid_mbr ()

Turn partition table into hybrid GPT/MBR table

set_mbr ()

Turn partition table into MBR (msdos table)

10.15.5 `kiwi.partitionner.msdos` Module

class `kiwi.partitionner.msdos.PartitionerMsDos` (*disk_provider*,
start_sector=None)

Bases: `kiwi.partitionner.base.PartitionerBase`

Implement old style msdos partition setup

create (*name*, *mbsize*, *type_name*, *flags=None*)

Create msdos partition

Parameters

- **name** (*string*) – partition name
- **mbsize** (*int*) – partition size
- **type_name** (*string*) – partition type
- **flags** (*list*) – additional flags

post_init ()

Post initialization method

Setup sfdisk partition type/flag map

resize_table (*entries=None*)

Resize partition table

Nothing to be done here for msdos table

Parameters **entries** (*int*) – unused

set_flag (*partition_id, flag_name*)

Set msdos partition flag

Parameters

- **partition_id** (*int*) – partition number
- **flag_name** (*string*) – name from flag map

10.15.6 Module Contents

class `kiwi.partitionner.Partitioner`

Bases: `object`

Partitioner factory

Parameters

- **table_type** (*string*) – Table type name
- **storage_provider** (*object*) – Instance of class based on DeviceProvider
- **start_sector** (*int*) – sector number

10.16 kiwi.repository Package

10.16.1 Submodules

10.16.2 `kiwi.repository.base` Module

class `kiwi.repository.base.RepositoryBase` (*root_bind*, *custom_args=None*)

Bases: `object`

Implements base class for package manager repository handling

Attributes

Parameters

- **root_bind** (*object*) – instance of RootBind
- **root_dir** (*str*) – root directory path name
- **shared_location** (*str*) – shared directory between image root and build system root

add_repo (*name, uri, repo_type, prio, dist, components, user, secret, credentials_file, repo_gpgcheck, pkg_gpgcheck, sourcetype*)

Add repository

Implementation in specialized repository class

Parameters

- **name** (*str*) – unused
- **uri** (*str*) – unused
- **repo_type** – unused
- **prio** (*int*) – unused
- **dist** (*str*) – unused
- **components** (*str*) – unused
- **user** (*str*) – unused
- **secret** (*str*) – unused
- **credentials_file** (*str*) – unused
- **repo_gpgcheck** (*bool*) – unused
- **pkg_gpgcheck** (*bool*) – unused
- **sourcetype** (*str*) – unused

cleanup_unused_repos ()

Cleanup/Delete unused repositories

Only configured repositories according to the image configuration are allowed to be active when building

Implementation in specialized repository class

delete_all_repos ()

Delete all repositories

Implementation in specialized repository class

delete_repo (*name*)

Delete repository

Implementation in specialized repository class

Parameters `name` (*str*) – unused

delete_repo_cache (*name*)

Delete repository cache

Implementation in specialized repository class

Parameters `name` (*str*) – unused

import_trusted_keys (*signing_keys*)

Imports trusted keys into the image

Implementation in specialized repository class

Parameters `signing_keys` (*list*) – list of the key files to import

post_init (*custom_args*)

Post initialization method

Implementation in specialized repository class

Parameters `custom_args` (*list*) – unused

runtime_config ()

Repository runtime configuration and environment

Implementation in specialized repository class

setup_package_database_configuration ()

Setup package database configuration

Implementation in specialized repository class

use_default_location ()

Call repository operations with default repository manager setup

Implementation in specialized repository class

10.16.3 kiwi.repository.dnf Module

class `kiwi.repository.dnf.RepositoryDnf` (*root_bind*, *cus-*
tom_args=None)

Bases: `kiwi.repository.base.RepositoryBase`

Implements repository handling for dnf package manager

Parameters

- **shared_dnf_dir** (*str*) – shared directory between image root and build system root
- **runtime_dnf_config_file** (*str*) – dnf runtime config file name
- **command_env** (*dict*) – customized os.environ for dnf
- **runtime_dnf_config** (*str*) – instance of ConfigParser


```
add_repo(name, uri, repo_type='rpm-md', prio=None, dist=None, components=None, user=None, secret=None, credentials_file=None, repo_gpgcheck=None, pkg_gpgcheck=None, sourcetype=None)
```

Add dnf repository

Parameters

- **name** (*str*) – repository base file name
- **uri** (*str*) – repository URI
- **repo_type** – repostory type name
- **prio** (*int*) – dnf repostory priority
- **dist** (*str*) – unused
- **components** (*str*) – unused
- **user** (*str*) – unused
- **secret** (*str*) – unused
- **credentials_file** (*str*) – unused
- **repo_gpgcheck** (*bool*) – enable repository signature validation
- **pkg_gpgcheck** (*bool*) – enable package signature validation
- **sourcetype** (*str*) – source type, one of ‘baseurl’, ‘metalink’ or ‘mirrorlist’

```
cleanup_unused_repos()
```

Delete unused dnf repositories

Repository configurations which are not used for this build must be removed otherwise they are taken into account for the package installations

```
delete_all_repos()
```

Delete all dnf repositories

```
delete_repo(name)
```

Delete dnf repository

Parameters **name** (*str*) – repository base file name

```
delete_repo_cache(name)
```

Delete dnf repository cache

The cache data for each repository is stored in a directory and additional files all starting with the repository name. The method glob deletes all files and directories matching the repository name followed by any characters to cleanup the cache information

Parameters **name** (*str*) – repository name

```
import_trusted_keys(signing_keys)
```

Imports trusted keys into the image

Parameters **signing_keys** (*list*) – list of the key files to import

post_init (*custom_args=None*)

Post initialization method

Store custom dnf arguments and create runtime configuration and environment

Parameters **custom_args** (*list*) – dnf arguments

runtime_config ()

dnf runtime configuration and environment

Returns dnf_args:list, command_env:dict

Return type dict

setup_package_database_configuration ()

Setup rpm macros for bootstrapping and image building

1. Create the rpm image macro which persists during the build
2. Create the rpm bootstrap macro to make sure for bootstrapping the rpm database location matches the host rpm database setup. This macro only persists during the bootstrap phase. If the image was already bootstrapped a compat link is created instead.

use_default_location ()

Setup dnf repository operations to store all data in the default places

10.16.4 kiwi.repository.zypper Module

class kiwi.repository.zypper.**RepositoryZypper** (*root_bind*, *custom_args=None*)

Bases: *kiwi.repository.base.RepositoryBase*

Implements repo handling for zypper package manager

Parameters

- **shared_zypper_dir** (*str*) – shared directory between image root and build system root
- **runtime_zypper_config_file** (*str*) – zypper runtime config file name
- **runtime_zypp_config_file** (*str*) – libzypp runtime config file name
- **zypper_args** (*list*) – zypper caller args plus additional custom args
- **command_env** (*dict*) – customized os.environ for zypper
- **runtime_zypper_config** (*object*) – instance of ConfigParser

add_repo (*name*, *uri*, *repo_type*='rpm-md', *prio*=None, *dist*=None, *components*=None, *user*=None, *secret*=None, *credentials_file*=None, *repo_gpgcheck*=None, *pkg_gpgcheck*=None, *sourcetype*=None)
Add zypper repository

Parameters

- **name** (*str*) – repository name
- **uri** (*str*) – repository URI
- **repo_type** – repository type name
- **prio** (*int*) – zypper repository priority
- **dist** (*str*) – unused
- **components** (*str*) – unused
- **user** (*str*) – credentials username
- **secret** (*str*) – credentials password
- **credentials_file** (*str*) – zypper credentials file
- **repo_gpgcheck** (*bool*) – enable repository signature validation
- **pkg_gpgcheck** (*bool*) – enable package signature validation
- **sourcetype** (*str*) – unused

cleanup_unused_repos ()

Delete unused zypper repositories

zypper creates a system solvable which is unwanted for the purpose of building images. In addition zypper fails with an error message ‘Failed to cache rpm database’ if such a system solvable exists and a new root system is created

All other repository configurations which are not used for this build must be removed too, otherwise they are taken into account for the package installations

delete_all_repos ()

Delete all zypper repositories

delete_repo (*name*)

Delete zypper repository

Parameters **name** (*str*) – repository name

delete_repo_cache (*name*)

Delete zypper repository cache

The cache data for each repository is stored in a list of directories of the same name as the repository name. The method deletes these directories to cleanup the cache information

Parameters **name** (*str*) – repository name

import_trusted_keys (*signing_keys*)

Imports trusted keys into the image

Parameters **signing_keys** (*list*) – list of the key files to import

post_init (*custom_args=None*)

Post initialization method

Store custom zypper arguments and create runtime configuration and environment

Parameters **custom_args** (*list*) – zypper arguments

runtime_config ()

zypper runtime configuration and environment

setup_package_database_configuration ()

Setup rpm macros for bootstrapping and image building

1. Create the rpm image macro which persists during the build
2. Create the rpm bootstrap macro to make sure for bootstrapping the rpm database location matches the host rpm database setup. This macro only persists during the bootstrap phase. If the image was already bootstrapped a compat link is created instead.
3. Create zypper compat link

use_default_location ()

Setup zypper repository operations to store all data in the default places

10.16.5 Module Contents

class `kiwi.repository.Repository`

Bases: `object`

Repository factory

Parameters

- **root_bind** (*object*) – instance of RootBind
- **package_manager** (*str*) – package manager name
- **custom_args** (*list*) – list of custom package manager arguments to setup the repository

Raises `KiwiRepositorySetupError` – if package_manager is not supported

10.17 kiwi.repository.template Package

10.17.1 Submodules

10.17.2 kiwi.repository.template.apr Module

class `kiwi.repository.template.apr.PackageManagerTemplateAprGet`
Bases: `object`

apr-get configuration file template

get_host_template (*exclude_docs=False*)
apr-get package manager template for apr-get called outside of the image, not chrooted

Return type Template

get_image_template (*exclude_docs=False*)
apr-get package manager template for apr-get called inside of the image, chrooted

Return type Template

10.17.3 Module Contents

10.18 kiwi.solver.repository Package

10.18.1 Submodules

10.18.2 kiwi.solver.repository.base Module

class `kiwi.solver.repository.base.SolverRepositoryBase` (*uri*,
user=None,
secret=None)
Bases: `object`

Base class interface for SAT solvable creation.

Parameters

- **uri** (*object*) – Instance of `Uri`
- **user** (*string*) – User name for uri authentication
- **secret** (*string*) – Secret token for uri authentication

create_repository_solvable (*target_dir='/var/tmp/kiwi/satsolver'*)
Create SAT solvable for this repository from previously created intermediate solvables by merge and store the result solvable in the specified *target_dir*

Parameters **target_dir** (*str*) – path name

Returns file path to solvable

Return type `str`

download_from_repository (*repo_source*, *target*)

Download given source file from the repository and store it as target file

The *repo_source* location is used relative to the repository location and will be part of a mime type source like: `file:///repo_path/repo_source`

Parameters

- **repo_source** (*str*) – source file in the repo
- **target** (*str*) – file path

Raises `KiwiUriOpenError` – if the download fails

is_uptodate (*target_dir*='/*var/tmp/kiwi/satsolver*')

Check if repository metadata is up to date

Returns True or False

Return type `bool`

timestamp ()

Return repository timestamp

The retrieval of the repository timestamp depends on the type of the repository and is therefore supposed to be implemented in the specialized Solver Repository classes. If no such implementation exists the method returns the value 'static' to indicate there is no timestamp information available.

Return type `str`

```
class kiwi.solver.repository.rpm_md.SolverRepositoryRpmMd (uri,  
                                                         user=None,  
                                                         se-  
                                                         cret=None)
```

Bases: `kiwi.solver.repository.base.SolverRepositoryBase`

Class for SAT solvable creation for rpm-md type repositories.

timestamp ()

Get timestamp from the first primary metadata

Returns time value as text

Return type `str`

```
class kiwi.solver.repository.rpm_dir.SolverRepositoryRpmDir (uri,  
                                                            user=None,  
                                                            se-  
                                                            cret=None)
```

Bases: `kiwi.solver.repository.base.SolverRepositoryBase`

Class for SAT solvable creation for rpm_dir type repositories.


```
class kiwi.solver.repository.suse.SolverRepositorySUSE (uri,  
                                                    user=None,  
                                                    se-  
                                                    cret=None)  
Bases: kiwi.solver.repository.base.SolverRepositoryBase
```

Class for SAT solvable creation for SUSE type repositories.

10.18.3 Module Contents

```
class kiwi.solver.repository.SolverRepository  
Bases: object
```

Repository factory for creation of SAT solvables

- **param object uri** Instance of *Uri*

10.19 kiwi.solver Package

10.19.1 Submodules

10.19.2 kiwi.solver.sat Module

```
class kiwi.solver.sat.Sat  
Bases: object
```

Sat Solver class to run package solver operations

The class uses SUSE's libsolvable plugin

add_repository (*solver_repository*)

Add a repository solvable to the pool. This basically add the required repository metadata which is needed to run a solver operation later.

Parameters **solver_repository** (*object*) – Instance of *SolverRepository*

solve (*job_names*, *skip_missing=False*, *ignore_recommended=True*)

Solve dependencies for the given job list. The list is allowed to contain element names of the following format:

- name describes a package name
- pattern:name describes a package collection name whose metadata type is called 'pattern' and stored as such in the repository metadata. Usually SUSE repos uses that
- group:name describes a package collection name whose metadata type is called 'group' and stored as such in the repository metadata. Usually RHEL/CentOS/Fedora repos uses that

Parameters

- **job_names** (*list*) – list of strings
- **skip_missing** (*bool*) – skip job if not found
- **ignore_recommended** (*bool*) – do not include recommended packages

Raises *KiwiSatSolverJobProblems* – if solver reports solving problems

Returns Transaction result information

Return type *dict*

10.19.3 Module Contents

10.20 kiwi.storage Package

10.20.1 Submodules

10.20.2 kiwi.storage.device_provider Module

class *kiwi.storage.device_provider.DeviceProvider*

Bases: *object*

Base class for any class providing storage devices

get_byte_size (*device*)

Size of device in bytes

Parameters *device* (*string*) – node name

Returns byte value from blockdev

Return type *int*

get_device ()

Representation of device nodes

Could provide one ore more devices representing the storage Implementation in specialized device provider class

get_uuid (*device*)

UUID of device

Parameters *device* (*string*) – node name

Returns UUID from blkid

Return type *str*

is_loop()

Check if device provider is loop based

By default this is always False and needs an implementation in the the specialized device provider class

Returns True or False

Return type bool

10.20.3 kiwi.storage.disk Module

class `kiwi.storage.disk.Disk`(*table_type*, *storage_provider*,
start_sector=None)

Bases: `kiwi.storage.device_provider.DeviceProvider`

Implements storage disk and partition table setup

Parameters

- **table_type** (*string*) – Partition table type name
- **storage_provider** (*object*) – Instance of class based on DeviceProvider
- **start_sector** (*int*) – sector number

activate_boot_partition()

Activate boot partition

Note: not all Partitioner instances supports this

create_boot_partition (*mbsize*)

Create boot partition

Populates `kiwi_BootPart`(id)

Parameters **mbsize** (*int*) – partition size

create_efi_csm_partition (*mbsize*)

Create EFI bios grub partition

Populates `kiwi_BiosGrub`(id)

Parameters **mbsize** (*int*) – partition size

create_efi_partition (*mbsize*)

Create EFI partition

Populates `kiwi_EfiPart`(id)

Parameters **mbsize** (*int*) – partition size

create_hybrid_mbr()

Turn partition table into a hybrid GPT/MBR table

Note: only GPT tables supports this

create_mbr()

Turn partition table into MBR (msdos table)

Note: only GPT tables supports this

create_prep_partition(*mbsize*)

Create prep partition

Populates kiwi_PrepPart(id)

Parameters **mbsize** (*int*) – partition size

create_root_lvm_partition(*mbsize*)

Create root partition for use with LVM

Populates kiwi_RootPart(id) and kiwi_RootPartVol(LVRoot)

Parameters **mbsize** (*int*) – partition size

create_root_partition(*mbsize*)

Create root partition

Populates kiwi_RootPart(id) and kiwi_BootPart(id) if no extra boot partition is requested

Parameters **mbsize** (*int*) – partition size

create_root_raid_partition(*mbsize*)

Create root partition for use with MD Raid

Populates kiwi_RootPart(id) and kiwi_RaidPart(id) as well as the default raid device node at boot time which is configured to be kiwi_RaidDev(/dev/mdX)

Parameters **mbsize** (*int*) – partition size

create_root_readonly_partition(*mbsize*)

Create root readonly partition for use with overlayfs

Populates kiwi_ReadOnlyPart(id), the partition is meant to contain a squashfs read-only filesystem. The partition size should be the size of the squashfs filesystem in order to avoid wasting disk space

Parameters **mbsize** (*int*) – partition size

create_spare_partition(*mbsize*)

Create spare partition for custom use

Populates kiwi_SparePart(id)

Parameters **mbsize** (*int*) – partition size

create_swap_partition(*mbsize*)

Create swap partition

Populates kiwi_SwapPart(id)

Parameters **mbsize** (*int*) – partition size

get_device()

Names of partition devices

Note that the mapping requires an explicit map() call

Returns instances of MappedDevice

Return type dict

get_public_partition_id_map()

Populated partition name to number map

is_loop()

Check if storage provider is loop based

The information is taken from the storage provider. If the storage provider is loop based the disk is it too

Returns True or False

Return type bool

map_partitions()

Map/Activate partitions

In order to access the partitions through a device node it is required to map them if the storage provider is loop based

wipe()

Zap (destroy) any GPT and MBR data structures if present For DASD disks create a new VTOC table

10.20.4 kiwi.storage.loop_device Module

```
class kiwi.storage.loop_device.LoopDevice(filename, file-  
                                         size_mbytes=None,  
                                         blocksize_bytes=None)
```

Bases: *kiwi.storage.device_provider.DeviceProvider*

Create and manage loop device file for block operations

Parameters

- **filename** (*string*) – loop file name to create
- **filesize_mbytes** (*int*) – size of the loop file
- **blocksize_bytes** (*int*) – blocksize used in loop driver

create (*overwrite=True*)

Setup a loop device of the blocksize given in the constructor The file to loop is created with the size specified in the constructor unless an existing one should not be overwritten

Parameters **overwrite** (*bool*) – overwrite existing file to loop

get_device()

Device node name

Returns device node name

Return type `str`

is_loop()

Always True

Returns True

Return type `bool`

10.20.5 kiwi.storage.luks_device Module

class `kiwi.storage.luks_device.LuksDevice` (*storage_provider*)

Bases: `kiwi.storage.device_provider.DeviceProvider`

Implements luks setup on a storage device

Parameters `storage_provider` (*object*) – Instance of class based on DeviceProvider

create_crypto_luks (*passphrase*, *os=None*, *options=None*, *keyfile=None*)

Create luks device. Please note the passphrase is readable at creation time of this image. Make sure your host system is secure while this process runs

Parameters

- **passphrase** (*string*) – credentials
- **os** (*string*) – distribution name to match distribution specific options for cryptsetup
- **options** (*list*) – further cryptsetup options
- **keyfile** (*string*) – file path name file path name which contains an alternative key to unlock the luks device

create_crypttab (*filename*)

Create crypttab, setting the UUID of the storage device

Parameters `filename` (*string*) – file path name

static create_random_keyfile (*filename*)

Create keyfile with random data

Parameters `filename` (*string*) – file path name

get_device()

Instance of MappedDevice providing the luks device

Returns mapped luks device

Return type `MappedDevice`

is_loop()

Check if storage provider is loop based

Return loop status from base storage provider

Returns True or False

Return type bool

10.20.6 kiwi.storage.mapped_device Module

class kiwi.storage.mapped_device.**MappedDevice** (*device*, *device_provider*)
Bases: *kiwi.storage.device_provider.DeviceProvider*

Hold a reference on a single device

Parameters

- **device_provider** (*object*) – Instance of class based on DeviceProvider
- **device** (*string*) – Device node name

get_device()

Mapped device node name

Returns device node name

Return type str

is_loop()

Check if storage provider is loop based

Return loop status from base storage provider

Returns True or False

Return type bool

10.20.7 kiwi.storage.raid_device Module

class kiwi.storage.raid_device.**RaidDevice** (*storage_provider*)
Bases: *kiwi.storage.device_provider.DeviceProvider*

Implement raid setup on a storage device

Parameters **storage_provider** (*object*) – Instance of class based on DeviceProvider

create_degraded_raid (*raid_level*)

Create a raid array in degraded mode with one device missing. This only works in the raid levels 0(striping) and 1(mirroring)

Parameters **raid_level** (*string*) – raid level name

create_raid_config(*filename*)

Create mdadm config file from mdadm request

Parameters **filename** (*string*) – config file name

get_device()

Instance of MappedDevice providing the raid device

Returns mapped raid device

Return type *MappedDevice*

is_loop()

Check if storage provider is loop based

Return loop status from base storage provider

Returns True or False

Return type *bool*

10.20.8 kiwi.storage.setup Module

class `kiwi.storage.setup.DiskSetup` (*xml_state*, *root_dir*)

Bases: *object*

Implements disk setup methods

Methods from this class provides information required before building a disk image

Parameters

- **xml_state** (*object*) – Instance of XMLState
- **root_dir** (*string*) – root directory path name

boot_partition_size()

Size of the boot partition in mbytes

Returns boot size mbytes

Return type *int*

get_boot_label()

Filesystem Label to use for the boot partition

Returns label name

Return type *str*

get_disksize_mbytes()

Precalculate disk size requirements in mbytes

Returns disk size mbytes

Return type *int*

get_efi_label()

Filesystem Label to use for the EFI partition

Returns label name

Return type `str`

get_root_label()

Filesystem Label to use for the root partition

If not specified in the XML configuration the default root label is set to 'ROOT'

Returns label name

Return type `str`

need_boot_partition()

Decide if an extra boot partition is needed. This is done with the bootpartition attribute from the type, however if it is not set it depends on some other type configuration parameters if we need a boot partition or not

Returns True or False

Return type `bool`

10.20.9 Module Contents

10.21 kiwi.storage.subformat Package

10.21.1 Submodules

10.21.2 kiwi.storage.subformat.base Module

```
class kiwi.storage.subformat.base.DiskFormatBase(xml_state,  
                                                root_dir, tar-  
                                                get_dir, cus-  
                                                tom_args=None)
```

Bases: `object`

Base class to create disk formats from a raw disk image

Parameters

- **xml_state** (*object*) – Instance of XMLState
- **root_dir** (*string*) – root directory path name
- **arch** (*string*) – platform.machine
- **target_dir** (*string*) – target directory path name
- **custom_args** (*dict*) – custom format options dictionary

create_image_format()

Create disk format

Implementation in specialized disk format class required

get_qemu_option_list (*custom_args*)

Create list of qemu options from custom_args dict

Parameters **custom_args** (*dict*) – arguments

Returns qemu option list

Return type *list*

get_target_file_path_for_format (*format_name*)

Create target file path name for specified format

Parameters **format_name** (*string*) – disk format name

Returns file path name

Return type *str*

has_raw_disk ()

Check if the base raw disk image exists

Returns True or False

Return type *bool*

post_init (*custom_args*)

Post initialization method

Implementation in specialized disk format class if required

Parameters **custom_args** (*list*) – unused

resize_raw_disk (*size_bytes*, *append=False*)

Resize raw disk image to specified size. If the request would actually shrink the disk an exception is raised. If the disk got changed the method returns True, if the new size is the same as the current size nothing gets resized and the method returns False

Parameters **size** (*int*) – size in bytes

Returns True or False

Return type *bool*

store_to_result (*result*)

Store result file of the format conversion into the provided result instance.

By default only the converted image file will be stored as compressed file. Subformats which creates additional metadata files or want to use other result flags needs to overwrite this method

Parameters **result** (*object*) – Instance of Result

10.21.3 kiwi.storage.subformat.gce Module

```
class kiwi.storage.subformat.gce.DiskFormatGce (xml_state,
                                                root_dir,      tar-
                                                get_dir,      cus-
                                                tom_args=None)
```

Bases: *kiwi.storage.subformat.base.DiskFormatBase*

Create GCE - Google Compute Engine image format

create_image_format ()

Create GCE disk format and manifest

get_target_file_path_for_format (format_name)

Google requires the image name to follow their naming convention. Therefore it's required to provide a suitable name by overriding the base class method

Parameters **format_name** (*string*) – gce

Returns file path name

Return type *str*

post_init (custom_args)

GCE disk format post initialization method

Store disk tag from custom args

Parameters **custom_args** (*dict*) – custom gce argument dictionary

```
{ '--tag': 'billing_code' }
```

store_to_result (result)

Store result file of the gce format conversion into the provided result instance. In this case compression is unwanted because the gce tarball is already created as a compressed archive

Parameters **result** (*object*) – Instance of Result

10.21.4 kiwi.storage.subformat.ova Module

```
class kiwi.storage.subformat.ova.DiskFormatOva (xml_state,
                                                root_dir,      tar-
                                                get_dir,      cus-
                                                tom_args=None)
```

Bases: *kiwi.storage.subformat.base.DiskFormatBase*

Create ova disk format, based on vmdk

create_image_format ()

Create ova disk format using ovftool from <https://www.vmware.com/support/developer/ovf>

post_init (*custom_args*)

vmdk disk format post initialization method

Store qemu options as list from custom args dict

Parameters **custom_args** (*dict*) – custom qemu arguments dictionary

store_to_result (*result*)

Store the resulting ova file into the provided result instance.

Parameters **result** (*object*) – Instance of Result

10.21.5 kiwi.storage.subformat.qcow2 Module

```
class kiwi.storage.subformat.qcow2.DiskFormatQcow2 (xml_state,  
                                                    root_dir,  
                                                    target_dir,  
                                                    custom_args=None)
```

Bases: *kiwi.storage.subformat.base.DiskFormatBase*

Create qcow2 disk format

create_image_format ()

Create qcow2 disk format

post_init (*custom_args*)

qcow2 disk format post initialization method

Store qemu options as list from custom args dict

Parameters **custom_args** (*dict*) – custom qemu arguments dictionary

store_to_result (*result*)

Store result file of the format conversion into the provided result instance.

In case of a qcow2 format we store the result uncompressed Since the format conversion only takes the real bytes into account such that the sparseness of the raw disk will not result in the output format and can be taken one by one

Parameters **result** (*object*) – Instance of Result

10.21.6 `kiwi.storage.subformat.vagrant_base` Module

```
class kiwi.storage.subformat.vagrant_base.DiskFormatVagrantBase (xml_state,  
                                                                root_dir,  
                                                                tar-  
                                                                get_dir,  
                                                                cus-  
                                                                tom_args=None)
```

Bases: `kiwi.storage.subformat.base.DiskFormatBase`

Base class for creating vagrant boxes.

The documentation of the vagrant box format can be found here: <https://www.vagrantup.com/docs/boxes/format.html> In a nutshell, a vagrant box is a tar, tar.gz or zip archive of the following:

1. `metadata.json`: A json file that contains the name of the provider and arbitrary additional data (that vagrant doesn't care about).
2. `Vagrantfile`: A Vagrantfile which defines the boxes' MAC address. It can be also used to define other settings of the box, e.g. the method via which the `/vagrant/` directory is shared. This file is either automatically generated by KIWI or we use a file that has been provided by the user (depends on the setting in `vagrantconfig.embebbbed_vagrantfile`)
3. The actual virtual disk image: this is provider specific and vagrant simply forwards it to your virtual machine provider.

Required methods/variables that child classes must implement:

- `vagrant_post_init()`
post initializing method that has to specify the vagrant provider name in `provider` and the box name in `image_format`. Note: new providers also needs to be specified in the schema and the box name needs to be registered to `kiwi.defaults.Defaults.get_disk_format_types()`
- `create_box_img()`

Optional methods:

- `get_additional_metadata()`
- `get_additional_vagrant_config_settings()`

`create_box_img(temp_image_dir)`

Provider specific image creation step: this function creates the actual box image. It must be implemented by a child class.

Parameters `temp_image_dir` (*str*) – path to a temporary directory inside which the image should be built

Returns A list of files that were created by this function and that should be included in the vagrant box

Return type `list`

create_image_format()

Create a vagrant box for any provider. This includes:

- creation of box metadata.json
- creation of box Vagrantfile (either from scratch or by using the user provided Vagrantfile)
- creation of result format tarball from the files created above

get_additional_metadata()

Provide `create_image_format()` with additional metadata that will be included in `metadata.json`.

The default implementation returns an empty dictionary.

Returns A dictionary that is serializable to JSON

Return type `dict`

get_additional_vagrant_config_settings()

Supply additional configuration settings for vagrant to be included in the resulting box.

This function can be used by child classes to customize the behavior for different providers: the supplied configuration settings get forwarded to `VagrantConfigTemplate.get_template()` as the parameter `custom_settings` and included in the Vagrantfile.

The default implementation returns nothing.

Returns additional vagrant settings

Return type `str`

post_init(custom_args)

vagrant disk format post initialization method

store vagrantconfig information provided via `custom_args`

Parameters `custom_args` (`dict`) – Contains instance of `xml_parse::vagrantconfig`

```
{'vagrantconfig': object}
```

store_to_result(result)

Store result file of the vagrant format conversion into the provided result instance. In this case compression is unwanted because the box is already created as a compressed tarball

Parameters `result` (`object`) – Instance of Result

vagrant_post_init()

Vagrant provider specific post initialization method

Setup vagrant provider and box name. This information must be set by the specialized provider class implementation to make the this base class methods effective

10.21.7 `kiwi.storage.subformat.vagrant_libvirt` Module

class `kiwi.storage.subformat.vagrant_libvirt.DiskFormatVagrantLibVirt` (*xml_*
root_
tar_
get_
cus_
tom_

Bases: `kiwi.storage.subformat.vagrant_base.DiskFormatVagrantBase`

Create a vagrant box for the libvirt provider

create_box_img (*temp_image_dir*)

Creates the qcow2 disk image box for libvirt vagrant provider

get_additional_metadata ()

Returns a dictionary containing the virtual image format and the size of the image.

get_additional_vagrant_config_settings ()

Returns settings for the libvirt provider telling vagrant to use kvm.

vagrant_post_init ()

Vagrant provider specific post initialization method

Setup vagrant provider and box name. This information must be set by the specialized provider class implementation to make the this base class methods effective

10.21.8 `kiwi.storage.subformat.vagrant_virtualbox` Module

class `kiwi.storage.subformat.vagrant_virtualbox.DiskFormatVagrantVirtualB`

Bases: `kiwi.storage.subformat.vagrant_base.DiskFormatVagrantBase`

Create a vagrant box for the virtualbox provider

create_box_img (*temp_image_dir*)

Create the virtual machine image for the Virtualbox vagrant provider.

This function creates the vmdk disk image and the ovf file. The latter is created via the class `kiwi.storage.subformat.template.virtualbox_ovf.VirtualboxOvfTemplate`.

get_additional_vagrant_config_settings ()

Configure the default shared folder to use rsync when guest additions are not present inside the box.

vagrant_post_init()

Vagrant provider specific post initialization method

Setup vagrant provider and box name. This information must be set by the specialized provider class implementation to make the this base class methods effective

10.21.9 `kiwi.storage.subformat.vdi` Module

```
class kiwi.storage.subformat.vdi.DiskFormatVdi (xml_state,  
                                              root_dir,      tar-  
                                              get_dir,      cus-  
                                              tom_args=None)
```

Bases: `kiwi.storage.subformat.base.DiskFormatBase`

Create vdi disk format

create_image_format()

Create vdi disk format

post_init(custom_args)

vdi disk format post initialization method

Store qemu options as list from custom args dict

Parameters **custom_args** (*dict*) – custom qemu arguments dictionary

10.21.10 `kiwi.storage.subformat.vhd` Module

```
class kiwi.storage.subformat.vhd.DiskFormatVhd (xml_state,  
                                              root_dir,      tar-  
                                              get_dir,      cus-  
                                              tom_args=None)
```

Bases: `kiwi.storage.subformat.base.DiskFormatBase`

Create vhd disk format

create_image_format()

Create vhd disk format

post_init(custom_args)

vhd disk format post initialization method

Store qemu options as list from custom args dict

Parameters **custom_args** (*dict*) – custom qemu arguments dictionary

10.21.11 `kiwi.storage.subformat.vhdfixed` Module

```
class kiwi.storage.subformat.vhdfixed.DiskFormatVhdFixed(xml_state,
                                                         root_dir,
                                                         tar-
                                                         get_dir,
                                                         cus-
                                                         tom_args=None)
```

Bases: `kiwi.storage.subformat.base.DiskFormatBase`

Create vhd image format in fixed subformat

```
create_image_format()
    Create vhd fixed disk format
```

```
post_init(custom_args)
    vhd disk format post initialization method
```

Store qemu options as list from custom args dict Extract disk tag from custom args

Parameters **custom_args** (*dict*) – custom vhdfixed and qemu argument dictionary

```
{'--tag': 'billing_code', '--qemu-opt': 'value'}
```

```
store_to_result(result)
    Store result file of the vhdfixed format conversion into the provided result instance.
    In this case compressing the result is preferred as vhdfixed is not a compressed or
    dynamic format.
```

Parameters **result** (*object*) – Instance of Result

10.21.12 `kiwi.storage.subformat.vhdx` Module

```
class kiwi.storage.subformat.vhdx.DiskFormatVhdx(xml_state,
                                                  root_dir, tar-
                                                  get_dir, cus-
                                                  tom_args=None)
```

Bases: `kiwi.storage.subformat.base.DiskFormatBase`

Create vhdx image format in dynamic subformat

```
create_image_format()
    Create vhdx dynamic disk format
```

```
post_init(custom_args)
    vhdx disk format post initialization method
```

Store qemu options as list from custom args dict

Parameters **custom_args** (*dict*) – custom qemu arguments dictionary

10.21.13 `kiwi.storage.subformat.vmdk` Module

```
class kiwi.storage.subformat.vmdk.DiskFormatVmdk (xml_state,  
                                                root_dir, tar-  
                                                get_dir, cus-  
                                                tom_args=None)
```

Bases: `kiwi.storage.subformat.base.DiskFormatBase`

Create vmdk disk format

```
create_image_format ()
```

Create vmdk disk format and machine settings file

```
post_init (custom_args)
```

vmdk disk format post initialization method

Store qemu options as list from custom args dict

Parameters `custom_args` (*dict*) – custom qemu arguments dictionary

```
store_to_result (result)
```

Store result files of the vmdk format conversion into the provided result instance.
This includes the vmdk image file and the VMware settings file

Parameters `result` (*object*) – Instance of Result

10.21.14 Module Contents

```
class kiwi.storage.subformat.DiskFormat
```

Bases: `object`

DiskFormat factory

Parameters

- **name** (*string*) – Format name
- **xml_state** (*object*) – Instance of XMLState
- **root_dir** (*string*) – root directory path name
- **target_dir** (*string*) – target directory path name

10.22 kiwi.storage.subformat.template Package

10.22.1 Submodules

10.22.2 kiwi.storage.subformat.template. vmware_settings Module

class `kiwi.storage.subformat.template.vmware_settings.VmwareSettingsTempl`
Bases: `object`

VMware machine settings template

get_template (*memory_setup=False, cpu_setup=False, network_setup=False, iso_setup=False, disk_controller='ide', iso_controller='ide'*)
VMware machine configuration template

Parameters

- **memory_setup** (*bool*) – with main memory setup `true/false`
- **cpu_setup** (*bool*) – with number of CPU's setup `true/false`
- **network_setup** (*bool*) – with network emulation `true/false`
- **iso_setup** (*bool*) – with CD/DVD drive emulation `true/false`
- **disk_controller** (*string*) – add disk controller setup to template
- **iso_controller** (*string*) – add CD/DVD controller setup to template
- **network_mac** (*string*) – add static MAC address setup to template
- **network_driver** (*string*) – add network driver setup to template
- **network_connection_type** (*string*) – add connection type to template

Return type `Template`

10.22.3 kiwi.storage.subformat.template.vagrant_config Module

class `kiwi.storage.subformat.template.vagrant_config.VagrantConfigTemplat`
Bases: `object`

Generate a Vagrantfile configuration template

This class creates a simple template for the Vagrantfile that is included inside a vagrant box.

The included Vagrantfile carries additional information for vagrant: by default that is nothing, but depending on the provider additional information need to be present. These can be passed via the parameter `custom_settings` to the method `get_template()`.

Example usage:

The default without any additional settings will result in this Vagrantfile:

```
>>> vagrant_config = VagrantConfigTemplate()
>>> print(
...     vagrant_config.get_template()
... )
Vagrant.configure("2") do |config|
end
```

If your provider/box requires additional settings, provide them as follows:

```
>>> extra_settings = dedent('''
... config.vm.hostname = "no-dead-beef"
... config.vm.provider :special do |special|
...     special.secret_settings = "please_work"
... end
... ''').strip()
>>> print(
...     vagrant_config.get_template(extra_settings)
... )
Vagrant.configure("2") do |config|
  config.vm.hostname = "no-dead-beef"
  config.vm.provider :special do |special|
    special.secret_settings = "please_work"
  end
end
end
```

get_template (*custom_settings=None*)

Return a new template with `custom_settings` included and indented appropriately.

Parameters `custom_settings` (*str*) – String of additional settings that get pasted into the Vagrantfile template. The string is put at the correct indentation level for you, but the internal indentation has to be provided by the caller.

Returns A string with `custom_settings` inserted at the appropriate position. The template has one the variable `mac_address` that must be substituted.

Return type *str*

10.22.4 `kiwi.storage.subformat.template.virtualbox_ovf` Module

class `kiwi.storage.subformat.template.virtualbox_ovf.VirtualboxOvfTemplate`
Bases: `object`

Generate a OVF file template for a vagrant virtualbox box

This class provides a template for virtualbox' ovf configuration file that is embedded inside the vagrant box. The template itself was extracted from a vagrant box that was build via packer and from a script provided by Neal Gompa.

`get_template()`

Return the actual ovf template. The following values must be substituted: - `vm_name`: the name of this VM - `disk_image_capacity`: Size of the virtual disk image in GB - `vm_description`: a description of this VM

10.22.5 Module contents

10.23 `kiwi.system` Package

10.23.1 Submodules

10.23.2 `kiwi.system.identifier` Module

class `kiwi.system.identifier.SystemIdentifier`
Bases: `object`

Create a random ID to identify the system

The information is used to create the mbrid file as an example

Parameters `image_id` (*str*) – hex identifier string

`calculate_id()`

Calculate random hex id

Using 4 tuples of rand in range from 1..0xfe

`get_id()`

Current hex identifier

Returns hex id

Return type *str*

`write(filename)`

Write current hex identifier to file

Parameters `filename` (*str*) – file path name

write_to_disk (*device_provider*)

Write current hex identifier to MBR at offset 0x1b8 on disk

Parameters **device_provider** (*object*) – Instance based on DeviceProvider

10.23.3 kiwi.system.kernel Module

class `kiwi.system.kernel.Kernel` (*root_dir*)

Bases: `object`

Implements kernel lookup and extraction from given root tree

Parameters

- **root_dir** (*str*) – root directory path name
- **kernel_names** (*list*) – list of kernel names to search for functions.sh::suseStripKernel() provides a normalized file so that we do not have to search for many different names in this code

copy_kernel (*target_dir, file_name=None*)

Copy kernel to specified target

If no file_name is given the target filename is set as kernel-<kernel.version>.kernel

Parameters

- **target_dir** (*str*) – target path name
- **filename** (*str*) – base filename in target

copy_xen_hypervisor (*target_dir, file_name=None*)

Copy xen hypervisor to specified target

If no file_name is given the target filename is set as hypervisor-<xen.name>

Parameters

- **target_dir** (*str*) – target path name
- **filename** (*str*) – base filename in target

get_kernel (*raise_on_not_found=False*)

Lookup kernel files and provide filename and version

Parameters **raise_on_not_found** (*bool*) – sets the method to raise an exception if the kernel is not found

Raises `KiwiKernelLookupError` – if raise_on_not_found flag is active and kernel is not found

Returns tuple with filename, kernelname and version

Return type namedtuple

get_xen_hypervisor ()

Lookup xen hypervisor and provide filename and hypervisor name

Returns tuple with filename and hypervisor name

Return type namedtuple

10.23.4 kiwi.system.prepare Module

class `kiwi.system.prepare.SystemPrepare` (*xml_state*, *root_dir*, *allow_existing=False*)

Bases: `object`

Implements preparation and installation of a new root system

Parameters

- **xml_state** (*object*) – instance of XMLState
- **profiles** (*list*) – list of configured profiles
- **root_bind** (*object*) – instance of RootBind
- **uri_list** (*list*) – a list of Uri references

delete_packages (*manager*, *packages*, *force=False*)

Delete one or more packages using the package manager inside of the new root directory. If the removal is set with `force` flag only listed packages are deleted and any dependency break or leftover is ignored.

Parameters

- **manager** (*object*) – instance of a PackageManager subclass
- **packages** (*list*) – package list
- **force** (*bool*) – force deletion true/false

Raises `KiwiSystemDeletePackagesFailed` – if installation process fails

install_bootstrap (*manager*, *plus_packages=None*)

Install system software using the package manager from the host, also known as bootstrapping

Parameters

- **manager** (*object*) – instance of a PackageManager subclass
- **plus_packages** (*list*) – list of additional packages

Raises `KiwiBootStrapPhaseFailed` – if the bootstrapping process fails either installing packages or including bootstrap archives

install_packages (*manager*, *packages*)

Install one or more packages using the package manager inside of the new root directory

Parameters

- **manager** (*object*) – instance of a PackageManager subclass

- **packages** (*list*) – package list

Raises *KiwiSystemInstallPackagesFailed* – if installation process fails

install_system (*manager*)

Install system software using the package manager inside of the new root directory. This is done via a chroot operation and requires the desired package manager to become installed via the bootstrap phase

Parameters **manager** (*object*) – instance of a *PackageManager* subclass

Raises *KiwiInstallPhaseFailed* – if the install process fails either installing packages or including any archive

pinch_system (*manager=None, force=False*)

Delete packages marked for deletion in the XML description. If force param is set to False uninstalls packages marked with `type="uninstall"` if any; if force is set to True deletes packages marked with `type="delete"` if any.

Parameters

- **manager** (*object*) – instance of *PackageManager*
- **force** (*bool*) – Forced deletion True|False

Raises *KiwiPackagesDeletePhaseFailed* – if the deletion packages process fails

setup_repositories (*clear_cache=False, signing_keys=None*)

Set up repositories for software installation and return a package manager for performing software installation tasks

Parameters

- **clear_cache** (*bool*) – flag the clear cache before configure anything
- **signing_keys** (*list*) – keys imported to the package manager

Returns instance of *PackageManager*

Return type *PackageManager*

update_system (*manager*)

Install package updates from the used repositories. the process uses the package manager from inside of the new root directory

Parameters **manager** (*object*) – instance of a *PackageManager* subclass

Raises *KiwiSystemUpdateFailed* – if packages update fails

10.23.5 kiwi.system.profile Module

class `kiwi.system.profile.Profile` (*xml_state*)

Bases: `object`

Create bash readable .profile environment from the XML description

The information is used by the kiwi first boot code.

Parameters

- **xml_state** (*object*) – instance of :class`XMLState`
- **dot_profile** (*dict*) – profile dictionary

add (*key*, *value*)

Add key/value pair to profile dictionary

Parameters

- **key** (*str*) – profile key
- **value** (*str*) – profile value

create (*filename*)

Create bash quoted profile

Parameters **filename** (*str*) – file path name

delete (*key*)

10.23.6 kiwi.system.result Module

class `kiwi.system.result.Result` (*xml_state*)

Bases: `object`

Collect image building results

Parameters

- **result_files** (*list*) – list of result files
- **class_version** (*object*) – *Result* class version
- **xml_state** (*object*) – instance of XMLState

add (*key*, *filename*, *use_for_bundle=True*, *compress=False*, *shasum=True*)

Add result tuple to result_files list

Parameters

- **key** (*str*) – name
- **filename** (*str*) – file path name
- **use_for_bundle** (*bool*) – use when bundling results true/false
- **compress** (*bool*) – compress when bundling true/false

- **shasum** (*bool*) – create shasum when bundling true/false

dump (*filename*)

Pickle dump this instance to a file

Parameters **filename** (*str*) – file path name

Raises **KiwiResultError** – if pickle fails to dump *Result* instance

get_results ()

Current list of result tuples

static load (*filename*)

Load pickle dumped filename into a Result instance

Parameters **filename** (*str*) – file path name

Raises **KiwiResultError** – if filename does not exist or pickle fails to load filename

print_results ()

Print results human readable

verify_image_size (*size_limit, filename*)

Verifies the given image file does not exceed the size limit. Throws an exception if the limit is exceeded. If the size limit is set to None no verification is done.

Parameters

- **size_limit** (*int*) – The size limit for filename in bytes.
- **filename** (*str*) – File to verify.

Raises **KiwiResultError** – if filename exceeds the size limit

```
class kiwi.system.result.result_file_type (filename,  
                                           use_for_bundle,    com-  
                                           press, shasum)
```

Bases: *tuple*

property compress

Alias for field number 2

property filename

Alias for field number 0

property shasum

Alias for field number 3

property use_for_bundle

Alias for field number 1

10.23.7 kiwi.system.root_bind Module

class `kiwi.system.root_bind.RootBind(root_init)`

Bases: `object`

Implements binding/copying of host system paths into the new root directory

Parameters

- **root_dir** (*str*) – root directory path name
- **cleanup_files** (*list*) – list of files to cleanup, delete
- **mount_stack** (*list*) – list of mounted directories for cleanup
- **dir_stack** (*list*) – list of directories for cleanup
- **config_files** (*list*) – list of initial config files
- **bind_locations** (*list*) – list of kernel filesystems to bind mount
- **shared_location** (*str*) – shared directory between image root and build system root

cleanup ()

Cleanup mounted locations, directories and intermediate config files

mount_kernel_file_systems ()

Bind mount kernel filesystems

Raises **KiwiMountKernelFileSystemsError** – if some kernel filesystem fails to mount

mount_shared_directory (*host_dir=None*)

Bind mount shared location

The shared location is a directory which shares data from the image buildsysteem host with the image root system. It is used for the repository setup and the package manager cache to allow chroot operations without being forced to duplicate this data

Parameters **host_dir** (*str*) – directory to share between image root and build system root

Raises **KiwiMountSharedDirectoryError** – if mount fails

setup_intermediate_config ()

Create intermediate config files

Some config files e.g etc/hosts needs to be temporarily copied from the buildsysteem host to the image root system in order to allow e.g DNS resolution in the way as it is configured on the buildsysteem host. These config files only exists during the image build process and are not part of the final image

Raises **KiwiSetupIntermediateConfigError** – if the management of intermediate configuration files fails

10.23.8 kiwi.system.root_init Module

class `kiwi.system.root_init.RootInit` (*root_dir*, *allow_existing=False*)

Bases: `object`

Implements creation of new root directory for a linux system

Host system independent static default files and device nodes are created to initialize a new base system

Parameters `root_dir` (*str*) – root directory path name

create ()

Create new system root directory

The method creates a temporary directory and initializes it for the purpose of building a system image from it. This includes the following setup:

- create core system paths
- create static core device nodes

On success the contents of the temporary location are synced to the specified `root_dir` and the temporary location will be deleted. That way we never work on an incomplete initial setup

Raises `KiwiRootInitCreationError` – if the init creation fails at some point

delete ()

Force delete root directory and its contents

10.23.9 kiwi.system.setup Module

class `kiwi.system.setup.SystemSetup` (*xml_state*, *root_dir*)

Bases: `object`

Implementation of system setup steps supported by kiwi

Kiwi is not responsible for the system configuration, however some setup steps needs to be performed in order to provide a minimal work environment inside of the image according to the desired image type.

Parameters

- **arch** (*str*) – platform.machine. The 32bit x86 platform is handled as 'ix86'
- **xml_state** (*object*) – instance of `XMLState`
- **description_dir** (*str*) – path to image description directory
- **derived_description_dir** – path to `derived_description_dir` boot image descriptions inherits data from the system image description, thus they are derived from another image description directory which is needed to e.g find system image archives, overlay files

- **root_dir** (*str*) – root directory path name

call_config_script ()

Call config.sh script chrooted

call_edit_boot_config_script (*filesystem*, *boot_part_id*, *working_directory=None*)

Call configured editbootconfig script _NON_ chrooted

Pass the boot filesystem name and the partition number of the boot partition as parameters to the call

Parameters

- **filesystem** (*str*) – boot filesystem name
- **boot_part_id** (*int*) – boot partition number
- **working_directory** (*str*) – directory name

call_edit_boot_install_script (*diskname*, *boot_device_node*, *working_directory=None*)

Call configured editbootinstall script _NON_ chrooted

Pass the disk file name and the device node of the boot partition as parameters to the call

Parameters

- **diskname** (*str*) – file path name
- **boot_device_node** (*str*) – boot device node name
- **working_directory** (*str*) – directory name

call_image_script ()

Call images.sh script chrooted

cleanup ()

Delete all traces of a kiwi description which are not required in the later image

create_fstab (*fstab*)

Create etc/fstab from given Fstab object

Custom fstab modifications are possible and handled in the following order:

1. Look for an optional fstab.append file which allows to append custom fstab entries to the final fstab. Once embedded the fstab.append file will be deleted
2. Look for an optional fstab.patch file which allows to patch the current contents of the fstab file with a given patch file. Once patched the fstab.patch file will be deleted
3. Look for an optional fstab.script file which is called chrooted for the purpose of updating the fstab file as appropriate. Note: There is no validation in place that checks if the script actually handles fstab or any other file in the image rootfs. Once called the fstab.script file will be deleted

Parameters `fstab` (*list*) – instance of Fstab

create_init_link_from_linuxrc ()

kiwi boot images provides the linuxrc script, however the kernel also expects an init executable to be present. This method creates a hard link to the linuxrc file

create_recovery_archive ()

Create a compressed recovery archive from the root tree for use with kiwi's recovery system. The method creates additional data into the image root filesystem which is deleted prior to the creation of a new recovery data set

export_modprobe_setup (*target_root_dir*)

Export etc/modprobe.d to given root_dir

Parameters `target_root_dir` (*str*) – path name

export_package_list (*target_dir*)

Export image package list as metadata reference used by the open builds service

Parameters `target_dir` (*str*) – path name

export_package_verification (*target_dir*)

Export package verification result as metadata reference used by the open builds service

Parameters `target_dir` (*str*) – path name

import_cdroot_files (*target_dir*)

Copy cdroot files from the image description to the specified target directory. Supported is a tar archive named config-cdroot.tar[.compression-postfix]

Parameters `target_dir` (*str*) – directory to unpack archive to

import_description ()

Import XML descriptions, custom scripts, archives and script helper methods

import_image_identifier ()

Create etc/ImageID identifier file

import_overlay_files (*follow_links=False*, *serve_owner_group=False*) *pre-*

Copy overlay files from the image description to the image root tree. Supported are a root/ directory or a root.tar.gz tarball. The root/ directory takes precedence over the tarball

Parameters

- **follow_links** (*bool*) – follow symlinks true/false
- **preserve_owner_group** (*bool*) – preserve permissions true/false

import_repositories_marked_as_imageinclude ()

Those <repository> sections which are marked with the imageinclude attribute should be permanently added to the image repository configuration

set_selinux_file_contexts (*security_context_file*)

Initialize the security context fields (extended attributes) on the files matching the security_context_file

Parameters **security_context_file** (*str*) – path file name

setup_groups ()

Add groups for configured users

setup_keyboard_map ()

Setup console keyboard

setup_locale ()

Setup UTF8 system wide locale

setup_machine_id ()

Setup systemd machine id

Empty out the machine id which was provided by the package installation process. This will instruct the dracut initrd code to create a new machine id. This way a golden image produces unique machine id's on first deployment and boot of the image.

Note: Requires dracut connected image type

This method must only be called if the image is of a type which gets booted via a dracut created initrd. Deleting the machine-id without the dracut initrd creating a new one produces an inconsistent system

setup_permissions ()

Check and Fix permissions using chkstat

Call chkstat in system mode which reads /etc/sysconfig/security to determine the configured security level and applies the appropriate permission definitions from the /etc/permissions* files. It's possible to provide those files as overlay files in the image description to apply a certain permission setup when needed. Otherwise the default setup as provided on the package level applies.

It's required that the image root system has chkstat installed. If not present KIWI skips this step and continues with a warning.

setup_plymouth_splash ()

Setup the KIWI configured splash theme as default

The method uses the plymouth-set-default-theme tool to setup the theme for the plymouth splash system. Only in case the tool could be found in the image root, it is assumed plymouth splash is in use and the tool is called in a chroot operation

setup_timezone ()

Setup timezone symlink

setup_users ()

Add/Modify configured users

10.23.10 kiwi.system.shell Module

class `kiwi.system.shell.Shell`

Bases: `object`

Special character handling for shell evaluated code

static `quote(message)`

Quote characters which have a special meaning for bash but should be used as normal characters. actually I had planned to use `pipes.quote` but it does not quote as I had expected it. e.g `'name_wit_a_$'` does not quote the `$` so we do it on our own for the scope of kiwi

Parameters `message` (*str*) – message text

Returns quoted text

Return type *str*

static `quote_key_value_file(filename)`

Quote given input file which has to be of the form `key=value` to be able to become sourced by the shell

Parameters `filename` (*str*) – file path name

Returns quoted text

Return type *str*

static `run_common_function(name, parameters)`

Run a function implemented in `config/functions.sh`

Parameters

- **name** (*str*) – function name
- **parameters** (*list*) – function arguments

10.23.11 kiwi.system.size Module

class `kiwi.system.size.SystemSize(source_dir)`

Bases: `object`

Provide source tree size information

Parameters `source_dir` (*str*) – source directory path name

accumulate_files()

Calculate sum of all files in the source tree

Returns number of files

Return type *int*

accumulate_mbyte_file_sizes(exclude=None)

Calculate data size of all data in the source tree

Parameters `exclude` (*list*) – list of paths to exclude

Returns mbytes

Return type `int`

customize (*size*, *requested_filesystem*)

Increase the sum of all file sizes by an empiric factor

Each filesystem has some overhead it needs to manage itself. Thus the plain data size is always smaller as the size of the container which embeds it. This method increases the given size by a filesystem specific empiric factor to ensure the given data size can be stored in a filesystem of the customized size

Parameters

- **size** (*int*) – mbsize to update
- **requested_filesystem** (*str*) – filesystem name

Returns mbytes

Return type `int`

10.23.12 kiwi.system.uri Module

class `kiwi.system.uri.Uri` (*uri*, *repo_type=None*)

Bases: `object`

Normalize url types available in a kiwi configuration into standard mime types

Parameters

- **repo_type** (*str*) – repository type name. Only needed if the uri is not enough to determine the repository type e.g for yast2 vs. rpm-md obs repositories
- **uri** (*str*) – URI, repository location, file
- **mount_stack** (*list*) – list of mounted locations
- **remote_uri_types** (*dict*) – dictionary of remote uri type names
- **local_uri_type** (*dict*) – dictionary of local uri type names

alias ()

Create hexdigest from URI as alias

If the repository definition from the XML description does not provide an alias, kiwi creates one for you. However it's better to assign a human readable alias in the XML configuration

Returns alias name as hexdigest

Return type `str`

credentials_file_name()

Filename to store repository credentials

Returns credentials file name

Return type `str`

get_fragment()

Returns the fragment part of the URI.

Returns fragment part of the URI if any, empty string otherwise

Return type `str`

is_public()

Check if URI is considered to be publicly reachable

Returns True or False

Return type `bool`

is_remote()

Check if URI is a remote or local location

Returns True or False

Return type `bool`

translate(*check_build_environment=True*)

Translate repository location according to their URI type

Depending on the URI type the provided location needs to be adapted e.g loop mounted in case of an ISO or updated by the service URL in case of an open build-service project name

Raises *KiwiUriStyleUnknown* – if the uri scheme can't be detected, is unknown or it is inconsistent with the build environment

Parameters *check_build_environment* (*bool*) – specify if the uri translation should depend on the environment the build is called in. As of today this only effects the translation result if the image build happens inside of the Open Build Service

Return type `str`

10.23.13 kiwi.system.users Module

class `kiwi.system.users.Users` (*root_dir*)

Bases: `object`

Operations on users and groups in a root directory

Parameters *root_dir* (*str*) – root directory path name

group_add (*group_name*, *options*)

Add group with options

Parameters

- **group_name** (*str*) – group name
- **options** (*list*) – groupadd options

group_exists (*group_name*)

Check if group exists

Parameters **group_name** (*str*) – group name**Returns** True or False**Return type** *bool***setup_home_for_user** (*user_name*, *group_name*, *home_path*)

Setup user home directory

Parameters

- **user_name** (*str*) – user name
- **group_name** (*str*) – group name
- **home_path** (*str*) – path name

user_add (*user_name*, *options*)

Add user with options

Parameters

- **user_name** (*str*) – user name
- **options** (*list*) – useradd options

user_exists (*user_name*)

Check if user exists

Parameters **user_name** (*str*) – user name**Return type** *bool***user_modify** (*user_name*, *options*)

Modify user with options

Parameters

- **user_name** (*str*) – user name
- **options** (*list*) – usermod options

10.23.14 Module Contents

10.24 kiwi.tasks package

10.24.1 Submodules

10.24.2 kiwi.tasks.base Module

class `kiwi.tasks.base.CliTask` (*should_perform_task_setup=True*)

Bases: `object`

Base class for all task classes, loads the task and provides the interface to the command options and the XML description

Attributes

- **should_perform_task_setup** Indicates if the task should perform the setup steps which covers the following task configurations: * setup debug level * setup logfile * setup color output

load_xml_description (*description_directory*)

Load, upgrade, validate XML description

Attributes

- **xml_data** instance of XML data toplevel domain (image), stateless data
- **config_file** used config file path
- **xml_state** Instance of XMLState, stateful data

quadruple_token (*option*)

Helper method for commandline options of the form `–option a,b,c,d`

Make sure to provide a common result for option values which separates the information in a comma separated list of values

Returns common option value representation

Return type `str`

run_checks (*checks*)

This method runs the given runtime checks excluding the ones disabled in the runtime configuration file.

Parameters **checks** (*dict*) – A dictionary with the runtime method names as keys and their arguments list as the values.

sextuple_token (*option*)

Helper method for commandline options of the form `–option a,b,c,d,e,f`

Make sure to provide a common result for option values which separates the information in a comma separated list of values

Returns common option value representation

Return type `str`

10.24.3 `kiwi.tasks.result_bundle` Module

class `kiwi.tasks.result_bundle.ResultBundleTask` (*should_perform_task_setup=True*)

Bases: `kiwi.tasks.base.CliTask`

Implements result bundler

Attributes

- **manual** Instance of Help

process ()

Create result bundle from the image build results in the specified target directory. Each result image will contain the specified bundle identifier as part of its filename. Uncompressed image files will also become xz compressed and a sha sum will be created from every result image

10.24.4 `kiwi.tasks.result_list` Module

class `kiwi.tasks.result_list.ResultListTask` (*should_perform_task_setup=True*)

Bases: `kiwi.tasks.base.CliTask`

Implements result listing

Attributes

- **manual** Instance of Help

process ()

List result information from a previous system command

10.24.5 `kiwi.tasks.system_build` Module

class `kiwi.tasks.system_build.SystemBuildTask` (*should_perform_task_setup=True*)

Bases: `kiwi.tasks.base.CliTask`

Implements building of system images

Attributes

- **manual** Instance of Help

process ()

Build a system image from the specified description. The build command combines the prepare and create commands

10.24.6 `kiwi.tasks.system_create` Module

class `kiwi.tasks.system_create.SystemCreateTask` (*should_perform_task_setup=True*)

Bases: `kiwi.tasks.base.CliTask`

Implements creation of system images

Attributes

- **manual** Instance of Help

process ()

Create a system image from the specified root directory the root directory is the result of a system prepare command

10.24.7 `kiwi.tasks.system_prepare` Module

class `kiwi.tasks.system_prepare.SystemPrepareTask` (*should_perform_task_setup=True*)

Bases: `kiwi.tasks.base.CliTask`

Implements preparation and installation of a new root system

Attributes

- **manual** Instance of Help

process ()

Prepare and install a new system for chroot access

10.24.8 `kiwi.tasks.system_update` Module

class `kiwi.tasks.system_update.SystemUpdateTask` (*should_perform_task_setup=True*)

Bases: `kiwi.tasks.base.CliTask`

Implements update and maintenance of root systems

Attributes

- **manual** Instance of Help

process ()

Update root system with latest repository updates and optionally allow to add or delete packages. the options to add or delete packages can be used multiple times

10.24.9 Module Contents

10.25 kiwi.utils Package

10.25.1 Submodules

10.25.2 kiwi.utils.checksum Module

class `kiwi.utils.block.BlockID(device)`

Bases: `object`

Get information from a block device

Parameters `device` (*str*) – block device node name name

get_blkid(*id_type*)

Retrieve information for specified metadata ID from block device

Parameters `id_type` (*string*) – metadata ID, see man blkid for details

Returns ID of the block device

Return type *str*

get_filesystem()

Retrieve filesystem type from block device

Returns filesystem type

Return type *str*

get_label()

Retrieve filesystem label from block device

Returns block device label

Return type *str*

get_uuid()

Retrieve filesystem uuid from block device

Returns uuid for the filesystem of the block device

Return type *str*

10.25.3 kiwi.utils.block Module

class `kiwi.utils.checksum.Checksum` (*source_filename*)

Bases: `object`

Manage checksum creation for files

Parameters

- **source_filename** (*str*) – source file name to build checksum for
- **checksum_filename** (*str*) – target file with checksum information

matches (*checksum, filename*)

Compare given checksum with reference checksum stored in the provided filename. If the checksum matches the method returns True, or False in case it does not match

Parameters

- **checksum** (*str*) – checksum string to compare
- **filename** (*str*) – filename containing checksum

Returns True or False

Return type `bool`

md5 (*filename=None*)

Create md5 checksum

Parameters **filename** (*str*) – filename for checksum

Returns checksum

Return type `str`

sha256 (*filename=None*)

Create sha256 checksum

Parameters **filename** (*str*) – filename for checksum

10.25.4 kiwi.utils.compress Module

class `kiwi.utils.compress.Compress` (*source_filename*,
keep_source_on_compress=False)

Bases: `object`

File compression / decompression

Parameters

- **keep_source** (*bool*) – Request to keep the uncompressed source
- **source_filename** (*str*) – Source file name to compress

- **supported_zipper** (*list*) – List of supported compression tools
- **compressed_filename** (*str*) – Compressed file name path with compression suffix
- **uncompressed_filename** (*str*) – Uncompressed file name path

get_format ()

Detect compression format

Returns compression format name or None if it couldn't be inferred

Return type Optional[*str*]

gzip ()

Create gzip(max compression) compressed file

uncompress (*temporary=False*)

Uncompress with format autodetection

By default the original source file will be changed into the uncompressed variant. If temporary is set to True a temporary file is created instead

Parameters **temporary** (*bool*) – uncompress to a temporary file

xz (*options=None*)

Create XZ compressed file

Parameters **options** (*list*) – custom xz compression options

10.25.5 kiwi.utils.sync Module

class kiwi.utils.sync.DataSync (*source_dir, target_dir*)

Bases: *object*

Sync data from a source directory to a target directory using the rsync protocol

Parameters

- **source_dir** (*str*) – source directory path name
- **target_dir** (*str*) – target directory path name

sync_data (*options=None, exclude=None*)

Sync data from source to target using rsync

Parameters

- **options** (*list*) – rsync options
- **exclude** (*list*) – file patterns to exclude

target_supports_extended_attributes ()

Check if the target directory supports extended filesystem attributes

Returns True or False

Return type `bool`

10.25.6 kiwi.utils.sysconfig Module

class `kiwi.utils.sysconfig.SysConfig` (*source_file*)

Bases: `object`

Read and Write sysconfig style files

Parameters **source_file** (*str*) – source file path

get (*key*)

write ()

Write back source file with changed content but in same order

10.25.7 Module Contents

10.26 kiwi.volume_manager Package

10.26.1 Submodules

10.26.2 kiwi.volume_manager.base Module

class `kiwi.volume_manager.base.VolumeManagerBase` (*device_map*,
root_dir,
volumes, *custom_args=None*)

Bases: `kiwi.storage.device_provider.DeviceProvider`

Implements base class for volume management interface

Parameters

- **mountpoint** (*str*) – root mountpoint for volumes
- **device_map** (*object*) – dictionary of low level DeviceProvider instances
- **root_dir** (*str*) – root directory path name
- **volumes** (*list*) – list of volumes from `XMLState::get_volumes()`
- **volume_group** (*str*) – volume group name
- **volume_map** (*map*) – map volume name to device node
- **mount_list** (*list*) – list of volume MountManager's
- **device** (*str*) – storage device node name

- **custom_args** (*dict*) – custom volume manager arguments for all volume manager and filesystem specific tasks
- **custom_filesystem_args** (*list*) – custom filesystem creation and mount arguments, subset of the custom_args information suitable to be passed to a FileSystem instance

Raises **KiwiVolumeManagerSetupError** – if the given root_dir doesn't exist

apply_attributes_on_volume (*toplevel, volume*)

create_volume_paths_in_root_dir ()

Implements creation of volume paths in the given root directory

create_volumes (*filesystem_name*)

Implements creation of volumes

Implementation in specialized volume manager class required

Parameters **filesystem_name** (*str*) – unused

get_canonical_volume_list ()

Implements hierarchical sorting of volumes according to their paths and provides information about the volume configured as the one eating all the rest space

Returns list of canonical_volume_type tuples

Return type *list*

get_device ()

Return current DeviceProvider dictionary

Returns device_map

Return type *dict*

get_fstab (*persistency_type, filesystem_name*)

Implements setup of the fstab entries. The method should return a list of fstab compatible entries

Parameters

- **persistency_type** (*str*) – unused
- **filesystem_name** (*str*) – unused

get_volume_mbsize (*volume, all_volumes, filesystem_name, image_type=None*)

Implements size lookup for the given path and desired filesystem according to the specified size type

Parameters

- **volume** (*tuple*) – volume to check size for
- **all_volumes** (*list*) – list of all volume tuples
- **filesystem_name** (*str*) – filesystem name

- **image_type** – build type name

Returns mbsize

Return type `int`

get_volumes ()

Implements return of dictionary of volumes and their mount options

is_loop ()

Check if storage provider is loop based

The information is taken from the storage provider. If the storage provider is loop based the volume manager is it too

Returns True of False

Return type `bool`

mount_volumes ()

Implements mounting of all volumes below one master directory

Implementation in specialized volume manager class required

post_init (*custom_args*)

Post initialization method

Implementation in specialized volume manager class if required

Parameters **custom_args** (*dict*) – unused

set_property_readonly_root ()

Implements setup of read-only root property

setup (*name=None*)

Implements setup required prior to the creation of volumes

Implementation in specialized volume manager class required

Parameters **name** (*str*) – unused

setup_mountpoint ()

Implements creation of a master directory holding the mounts of all volumes

sync_data (*exclude=None*)

Implements sync of root directory to mounted volumes

Parameters **exclude** (*list*) – file patterns to exclude

umount_volumes ()

Implements unmounting of all volumes

Implementation in specialized volume manager class required

10.26.3 kiwi.volume_manager.btrfs Module

```
class kiwi.volume_manager.btrfs.VolumeManagerBtrfs (device_map,  
                                                    root_dir,  
                                                    volumes,  
                                                    cus-  
                                                    tom_args=None)
```

Bases: *kiwi.volume_manager.base.VolumeManagerBase*

Implements btrfs sub-volume management

Parameters

- **subvol_mount_list** (*list*) – list of mounted btrfs subvolumes
- **toplevel_mount** (*object*) – MountManager for root mount-point

create_volumes (*filesystem_name*)

Create configured btrfs subvolumes

Any btrfs subvolume is of the same btrfs filesystem. There is no way to have different filesystems per btrfs subvolume. Thus the filesystem_name has no effect for btrfs

Parameters **filesystem_name** (*string*) – unused

get_fstab (*persistency_type='by-label', filesystem_name=None*)

Implements creation of the fstab entries. The method returns a list of fstab compatible entries

Parameters

- **persistency_type** (*string*) – by-label | by-uuid
- **filesystem_name** (*string*) – unused

Returns list of fstab entries

Return type *list*

get_volumes ()

Return dict of volumes

Returns volumes dictionary

Return type *dict*

mount_volumes ()

Mount btrfs subvolumes

post_init (*custom_args*)

Post initialization method

Store custom btrfs initialization arguments

Parameters **custom_args** (*list*) – custom btrfs volume manager arguments

set_property_readonly_root ()

Sets the root volume to be a readonly filesystem

setup (*name=None*)

Setup btrfs volume management

In case of btrfs a toplevel(@) subvolume is created and marked as default volume. If snapshots are activated via the custom_args the setup method also created the @/snapshots/1/snapshot subvolumes. There is no concept of a volume manager name, thus the name argument is not used for btrfs

Parameters **name** (*string*) – unused

sync_data (*exclude=None*)

Sync data into btrfs filesystem

If snapshots are activated the root filesystem is synced into the first snapshot

Parameters **exclude** (*list*) – files to exclude from sync

umount_volumes ()

Umount btrfs subvolumes

Returns True if all subvolumes are successfully unmounted

Return type *bool*

10.26.4 kiwi.volume_manager.lvm Module

```
class kiwi.volume_manager.lvm.VolumeManagerLVM(device_map,  
                                              root_dir,      vol-  
                                              umes,        cus-  
                                              tom_args=None)
```

Bases: *kiwi.volume_manager.base.VolumeManagerBase*

Implements LVM volume management

create_volumes (*filesystem_name*)

Create configured lvm volumes and filesystems

All volumes receive the same filesystem

Parameters **filesystem_name** (*str*) – volumes filesystem name

get_device ()

Dictionary of MappedDevice instances per volume

Note: The mapping requires an explicit create_volumes() call

Returns root plus volume device map

Return type *dict*

get_fstab (*persistence_type, filesystem_name*)

Implements creation of the fstab entries. The method returns a list of fstab compatible entries

Parameters

- **persistency_type** (*str*) – unused
- **filesystem_name** (*str*) – volumes filesystem name

Returns fstab entries

Return type *list*

get_volumes ()

Return dict of volumes

Returns volumes dictionary

Return type *dict*

mount_volumes ()

Mount lvm volumes

post_init (*custom_args*)

Post initialization method

Store custom lvm initialization arguments

Parameters **custom_args** (*list*) – custom lvm volume manager arguments

setup (*volume_group_name*='systemVG')

Setup lvm volume management

In case of LVM a new volume group is created on a PV initialized storage device

Parameters **name** (*str*) – volume group name

umount_volumes ()

Umount lvm volumes

Returns True if all subvolumes are successfully unmounted

Return type *bool*

10.26.5 Module Contents

class `kiwi.volume_manager.VolumeManager`

Bases: *object*

VolumeManager factory

Parameters

- **name** (*str*) – volume management name
- **device_map** (*dict*) – dictionary of low level DeviceProvider instances
- **root_dir** (*str*) – root directory path name

- **volumes** (*list*) – list of volumes from `XMLState::get_volumes()`
- **custom_args** (*dict*) – dictionary of custom volume manager arguments

Links

- [GitHub Sources](#)
- [GitHub Releases](#)
- [RPM Packages](#)
- [Build Tests SUSE\(x86\)](#)
- [Build Tests SUSE\(arm\)](#)
- [Build Tests SUSE\(s390\)](#)
- [Build Tests SUSE\(ppc64le\)](#)
- [Build Tests Fedora\(x86\)](#)
- [Build Tests Fedora\(arm\)](#)
- [Build Tests Fedora\(ppc64le\)](#)
- [Build Tests CentOS\(x86\)](#)
- [Build Tests Ubuntu\(x86\)](#)

THE APPLIANCE CONCEPT

An appliance is a ready to use image of an operating system including a pre-configured application for a specific use case. The appliance is provided as an image file and needs to be deployed to, or activated in the target system or service.

KIWI NG can create appliances in various forms: beside classical installation ISOs and images for virtual machines it can also build images that boot via PXE or Vagrant boxes.

In KIWI NG, the appliance is specified via a collection of human readable files in a directory, also called the `image description`. At least one XML file `config.xml` or `.kiwi` is required. In addition there may as well be other files like scripts or configuration data.

USE CASES

Not convinced yet? You can find a selection of the possible uses cases below:

- You are a system administrator and wish to create a customized installer for your network that includes additional software and your organizations certificates? KIWI NG allows you to select which packages will be included in the final image. On top of that you can add files to arbitrary locations in the filesystem, for example to include SSL or SSH keys. You can also tell KIWI NG to create an image that can be booted via PXE, so that you don't even have to leave your desk to reinstall a system.
- You want to create a custom spin of your favorite Linux distribution with additional repositories and packages that are not present by default? With KIWI NG you can configure the repositories of your final image via the image description and tweak the list of packages that are going to be installed to match your target audience.
- The Raspberry Pi that is coordinating your home's Internet of Thing (IoT) devices got very popular among your friends and every single one of them wants a copy of that? KIWI NG will build you ready to deploy images for your Raspberry Pi, tweaked to your needs.

CHAPTER THIRTEEN

CONTACT

- [Mailing list](#)

The `kiwi-images` group is an open group and anyone can [subscribe](#), even if you do not have a Google account.

- [Matrix](#)

An open network for secure, decentralized communication. Please find the `#kiwi` room via [Riot](#) on the web or by using the supported [clients](#).

PYTHON MODULE INDEX

k

- kiwi, 192
- kiwi.app, 136
- kiwi.archive, 194
- kiwi.archive.cpio, 192
- kiwi.archive.tar, 192
- kiwi.boot, 213
- kiwi.boot.image, 198
- kiwi.boot.image.base, 194
- kiwi.boot.image.builtin_kiwi, 197
- kiwi.boot.image.dracut, 196
- kiwi.bootloader, 210
- kiwi.bootloader.config, 207
- kiwi.bootloader.config.base, 198
- kiwi.bootloader.config.grub2, 203
- kiwi.bootloader.config.isolinux, 205
- kiwi.bootloader.config.zipl, 206
- kiwi.bootloader.install, 210
- kiwi.bootloader.install.base, 208
- kiwi.bootloader.install.grub2, 208
- kiwi.bootloader.install.zipl, 209
- kiwi.bootloader.template, 213
- kiwi.bootloader.template.grub2, 210
- kiwi.bootloader.template.isolinux, 211
- kiwi.bootloader.template.zipl, 213
- kiwi.builder, 219
- kiwi.builder.archive, 213
- kiwi.builder.container, 214
- kiwi.builder.disk, 215
- kiwi.builder.filesystem, 216
- kiwi.builder.install, 217
- kiwi.builder.live, 218
- kiwi.builder.pxe, 218
- kiwi.cli, 136
- kiwi.command, 138
- kiwi.command_process, 139
- kiwi.container, 220
- kiwi.container.oci, 219
- kiwi.container.setup, 222
- kiwi.container.setup.base, 221
- kiwi.container.setup.docker, 222
- kiwi.defaults, 140
- kiwi.exceptions, 153
- kiwi.filesystem, 228
- kiwi.filesystem.base, 222
- kiwi.filesystem.btrfs, 224
- kiwi.filesystem.clicfs, 224
- kiwi.filesystem.ext2, 225
- kiwi.filesystem.ext3, 225
- kiwi.filesystem.ext4, 225
- kiwi.filesystem.fat16, 226
- kiwi.filesystem.fat32, 226
- kiwi.filesystem.isofs, 226
- kiwi.filesystem.setup, 227
- kiwi.filesystem.squashfs, 227
- kiwi.filesystem.xfs, 228
- kiwi.firmware, 164
- kiwi.help, 165
- kiwi.iso_tools, 232
- kiwi.iso_tools.base, 228
- kiwi.iso_tools.cdrtools, 229
- kiwi.iso_tools.iso, 231
- kiwi.kiwi, 166
- kiwi.logger, 166
- kiwi.logger_color_formatter, 167
- kiwi.logger_filter, 168
- kiwi.mount_manager, 170

- `kiwi.package_manager`, [239](#)
- `kiwi.package_manager.base`, [232](#)
- `kiwi.package_manager.dnf`, [235](#)
- `kiwi.package_manager.zypper`, [237](#)
- `kiwi.partitioner`, [243](#)
- `kiwi.partitioner.base`, [240](#)
- `kiwi.partitioner.dasd`, [241](#)
- `kiwi.partitioner.gpt`, [242](#)
- `kiwi.partitioner.msdos`, [242](#)
- `kiwi.path`, [170](#)
- `kiwi.privileges`, [172](#)
- `kiwi.repository`, [249](#)
- `kiwi.repository.base`, [243](#)
- `kiwi.repository.dnf`, [245](#)
- `kiwi.repository.template`, [250](#)
- `kiwi.repository.template.apt`, [250](#)
- `kiwi.repository.zypper`, [247](#)
- `kiwi.runtime_checker`, [173](#)
- `kiwi.runtime_config`, [175](#)
- `kiwi.solver`, [253](#)
- `kiwi.solver.repository`, [252](#)
- `kiwi.solver.repository.base`, [250](#)
- `kiwi.solver.repository.rpm_dir`, [251](#)
- `kiwi.solver.repository.rpm_md`, [251](#)
- `kiwi.solver.repository.suse`, [251](#)
- `kiwi.solver.sat`, [252](#)
- `kiwi.storage`, [260](#)
- `kiwi.storage.device_provider`, [253](#)
- `kiwi.storage.disk`, [254](#)
- `kiwi.storage.loop_device`, [256](#)
- `kiwi.storage.luks_device`, [257](#)
- `kiwi.storage.mapped_device`, [258](#)
- `kiwi.storage.raid_device`, [258](#)
- `kiwi.storage.setup`, [259](#)
- `kiwi.storage.subformat`, [269](#)
- `kiwi.storage.subformat.base`, [260](#)
- `kiwi.storage.subformat.gce`, [262](#)
- `kiwi.storage.subformat.ova`, [262](#)
- `kiwi.storage.subformat.qcow2`, [263](#)
- `kiwi.storage.subformat.template`, [272](#)
- `kiwi.storage.subformat.template.vagrant_base`, [270](#)
- `kiwi.storage.subformat.template.virtualbox`, [272](#)
- `kiwi.storage.subformat.template.vmware_se`, [270](#)
- `kiwi.storage.subformat.vagrant_base`, [264](#)
- `kiwi.storage.subformat.vagrant_libvirt`, [266](#)
- `kiwi.storage.subformat.vagrant_virtualbox`, [266](#)
- `kiwi.storage.subformat.vdi`, [267](#)
- `kiwi.storage.subformat.vhd`, [267](#)
- `kiwi.storage.subformat.vhdfixed`, [268](#)
- `kiwi.storage.subformat.vhdx`, [268](#)
- `kiwi.storage.subformat.vmdk`, [269](#)
- `kiwi.system`, [287](#)
- `kiwi.system.identifier`, [272](#)
- `kiwi.system.kernel`, [273](#)
- `kiwi.system.prepare`, [274](#)
- `kiwi.system.profile`, [276](#)
- `kiwi.system.result`, [276](#)
- `kiwi.system.root_bind`, [278](#)
- `kiwi.system.root_init`, [279](#)
- `kiwi.system.setup`, [279](#)
- `kiwi.system.shell`, [283](#)
- `kiwi.system.size`, [283](#)
- `kiwi.system.uri`, [284](#)
- `kiwi.system.users`, [285](#)
- `kiwi.tasks`, [290](#)
- `kiwi.tasks.base`, [287](#)
- `kiwi.tasks.result_bundle`, [288](#)
- `kiwi.tasks.result_list`, [288](#)
- `kiwi.tasks.system_build`, [288](#)
- `kiwi.tasks.system_create`, [289](#)
- `kiwi.tasks.system_prepare`, [289](#)
- `kiwi.tasks.system_update`, [289](#)
- `kiwi.utils`, [293](#)
- `kiwi.utils.block`, [290](#)
- `kiwi.utils.checksum`, [291](#)
- `kiwi.utils.compress`, [291](#)
- `kiwi.utils.sync`, [292](#)
- `kiwi.utils.sysconfig`, [293](#)
- `kiwi.version`, [177](#)
- `kiwi.volume_manager`, [298](#)
- `kiwi.volume_manager.base`, [293](#)
- `kiwi.volume_manager.btrfs`, [296](#)
- `kiwi.volume_manager.lvm`, [297](#)

`kiwi.xml_description`, [177](#)
`kiwi.xml_state`, [178](#)

INDEX

A

- `access()` (*kiwi.path.Path* static method), 170
 - `accumulate_files()` (*kiwi.system.size.SystemSize* method), 283
 - `accumulate_mbyte_file_sizes()` (*kiwi.system.size.SystemSize* method), 283
 - `activate_boot_partition()` (*kiwi.storage.disk.Disk* method), 254
 - `add()` (*kiwi.system.profile.Profile* method), 276
 - `add()` (*kiwi.system.result.Result* method), 276
 - `add_container_config_label()` (*kiwi.xml_state.XMLState* method), 178
 - `add_efi_loader_parameters()` (*kiwi.iso_tools.base.IsoToolsBase* method), 228
 - `add_efi_loader_parameters()` (*kiwi.iso_tools.cdrtools.IsoToolsCdrTools* method), 229
 - `add_repo()` (*kiwi.repository.base.RepositoryBase* method), 244
 - `add_repo()` (*kiwi.repository.dnf.RepositoryDnf* method), 245
 - `add_repo()` (*kiwi.repository.zypper.RepositoryZypper* method), 247
 - `add_repository()` (*kiwi.solver.sat.Sat* method), 252
 - `add_repository()` (*kiwi.xml_state.XMLState* method), 178
 - `alias()` (*kiwi.system.uri.Uri* method), 284
 - `App` (class in *kiwi.app*), 136
 - `append_files()` (*kiwi.archive.tar.ArchiveTar* method), 193
 - `append_unpartitioned_space()` (*kiwi.builder.disk.DiskBuilder* method), 215
 - `apply_attributes_on_volume()` (*kiwi.volume_manager.base.VolumeManagerBase* method), 294
 - `ArchiveBuilder` (class in *kiwi.builder.archive*), 213
 - `ArchiveCpio` (class in *kiwi.archive.cpio*), 192
 - `ArchiveTar` (class in *kiwi.archive.tar*), 192
- ### B
- `bind_mount()` (*kiwi.mount_manager.MountManager* method), 170
 - `bios_mode()` (*kiwi.firmware.FirmWare* method), 164
 - `BlockID` (class in *kiwi.utils.block*), 290
 - `boot_partition_size()` (*kiwi.storage.setup.DiskSetup* method), 259
 - `BootImage` (class in *kiwi.boot.image*), 198
 - `BootImageBase` (class in *kiwi.boot.image.base*), 194
 - `BootImageDracut` (class in *kiwi.boot.image.dracut*), 196
 - `BootImageKiwi` (class in *kiwi.boot.image.builtin_kiwi*), 197
 - `BootLoaderConfig` (class in *kiwi.bootloader.config*), 207
 - `BootLoaderConfigBase` (class in

- kiwi.bootloader.config.base*), 198
- BootLoaderConfigGrub2 (class in *kiwi.bootloader.config.grub2*), 203
- BootLoaderConfigIsoLinux (class in *kiwi.bootloader.config.isolinux*), 205
- BootLoaderConfigZipl (class in *kiwi.bootloader.config.zipl*), 206
- BootLoaderInstall (class in *kiwi.bootloader.install*), 210
- BootLoaderInstallBase (class in *kiwi.bootloader.install.base*), 208
- BootLoaderInstallGrub2 (class in *kiwi.bootloader.install.grub2*), 208
- BootLoaderInstallZipl (class in *kiwi.bootloader.install.zipl*), 209
- BootLoaderTemplateGrub2 (class in *kiwi.bootloader.template.grub2*), 210
- BootLoaderTemplateIsoLinux (class in *kiwi.bootloader.template.isolinux*), 211
- BootLoaderTemplateZipl (class in *kiwi.bootloader.template.zipl*), 213
- ## C
- calculate_id() (*kiwi.system.identifier.SystemIdentifier* method), 272
- call() (*kiwi.command.Command* static method), 138
- call_config_script() (*kiwi.system.setup.SystemSetup* method), 280
- call_edit_boot_config_script() (*kiwi.system.setup.SystemSetup* method), 280
- call_edit_boot_install_script() (*kiwi.system.setup.SystemSetup* method), 280
- call_image_script() (*kiwi.system.setup.SystemSetup* method), 280
- check_appx_naming_conventions_value() (*kiwi.runtime_checker.RuntimeChecker* method), 173
- check_architecture_supports_iso_file() (*kiwi.runtime_checker.RuntimeChecker* method), 173
- check_boot_description_exists() (*kiwi.runtime_checker.RuntimeChecker* method), 173
- check_consistent_kernel_in_boot_and_system() (*kiwi.runtime_checker.RuntimeChecker* method), 173
- check_container_tool_chain_installed() (*kiwi.runtime_checker.RuntimeChecker* method), 173
- check_dracut_module_for_disk_oem_in_package() (*kiwi.runtime_checker.RuntimeChecker* method), 173
- check_dracut_module_for_disk_overlay_in_package() (*kiwi.runtime_checker.RuntimeChecker* method), 173
- check_dracut_module_for_live_iso_in_package() (*kiwi.runtime_checker.RuntimeChecker* method), 173
- check_dracut_module_for_oem_install_in_package() (*kiwi.runtime_checker.RuntimeChecker* method), 174
- check_efi_mode_for_disk_overlay_correctly() (*kiwi.runtime_checker.RuntimeChecker* method), 174
- check_for_root_permissions() (*kiwi.privileges.Privileges* static method), 172
- check_image_include_repos_publicly_resolvable() (*kiwi.runtime_checker.RuntimeChecker* method), 174
- check_mediacheck_installed() (*kiwi.runtime_checker.RuntimeChecker* method), 174
- check_minimal_required_preferences() (*kiwi.runtime_checker.RuntimeChecker* method), 174
- check_repositories_configured() (*kiwi.runtime_checker.RuntimeChecker* method), 174
- check_target_directory_not_in_shared_cache() (*kiwi.runtime_checker.RuntimeChecker* method), 174
- check_volume_label_used_with_lvm() (*kiwi.runtime_checker.RuntimeChecker* method), 174
- check_volume_setup_defines_multiple_fullsize() (*kiwi.runtime_checker.RuntimeChecker* method), 174
- check_volume_setup_defines_reserved_label() (*kiwi.runtime_checker.RuntimeChecker* method), 174

(*kiwi.runtime_checker.RuntimeChecker* *kiwi.container.oci*), 219
method), 174 ContainerSetup (class in
check_volume_setup_has_no_root_definition(*kiwi.container.setup*), 222
(*kiwi.runtime_checker.RuntimeChecker* ContainerSetupBase (class in
method), 174 *kiwi.container.setup.base*), 221
check_xen_uniquely_setup_as_server_container(*kiwi.container.setup.docker* (class in
(*kiwi.runtime_checker.RuntimeChecker* *kiwi.container.setup.docker*), 222
method), 175 copy_bootdelete_packages()
Checksum (class in *kiwi.utils.checksum*), (*kiwi.xml_state.XMLState* method),
291 179
cleanup() (*kiwi.system.root_bind.RootBind* copy_bootincluded_archives()
method), 278 (*kiwi.xml_state.XMLState* method),
cleanup() (*kiwi.system.setup.SystemSetup* 179
method), 280 copy_bootincluded_packages()
cleanup_requests() (*kiwi.xml_state.XMLState* method),
(*kiwi.package_manager.base.PackageManagerBase* 179
method), 232 copy_build_type_attributes()
cleanup_unused_repos() (*kiwi.xml_state.XMLState* method),
(*kiwi.repository.base.RepositoryBase* 179
method), 244 copy_displayname()
cleanup_unused_repos() (*kiwi.xml_state.XMLState* method),
(*kiwi.repository.dnf.RepositoryDnf* 179
method), 246 copy_drivers_sections()
cleanup_unused_repos() (*kiwi.xml_state.XMLState* method),
(*kiwi.repository.zypper.RepositoryZypper* 179
method), 248 copy_kernel()
Cli (class in *kiwi.cli*), 136 (*kiwi.system.kernel.Kernel* method),
CliTask (class in *kiwi.tasks.base*), 287 273
ColorFormatter (class in *kiwi.logger_color_formatter*), 167 copy_machine_section()
ColorMessage (class in *kiwi.logger_color_formatter*), 168 (*kiwi.xml_state.XMLState* method),
Command (class in *kiwi.command*), 138 179
command_type (in module *kiwi.command*), copy_name() (*kiwi.xml_state.XMLState*
139 method), 179
CommandIterator (class in *kiwi.command_process*), 139 copy_oemconfig_section()
CommandProcess (class in *kiwi.command_process*), 139 (*kiwi.xml_state.XMLState* method),
Compress (class in *kiwi.utils.compress*), 291 180
compress() copy_preferences_subsections()
(*kiwi.system.result.result_file_type* (*kiwi.xml_state.XMLState* method),
property), 277 180
ContainerBuilder (class in *kiwi.builder.container*), 214 copy_repository_sections()
ContainerImage (class in *kiwi.container*), copy_strip_sections()
220 (*kiwi.xml_state.XMLState* method),
ContainerImageOCI (class in 180
copy_systemdisk_section()
(*kiwi.xml_state.XMLState* method),

[copy_xen_hypervisor\(\)](#) [method](#), 266
 ([kiwi.system.kernel.Kernel](#) [method](#)), [create_crypto_luks\(\)](#)
 273 [\(kiwi.storage.luks_device.LuksDevice](#)
[create\(\)](#) [\(kiwi.archive.cpio.ArchiveCpio](#) [method](#)), 257
 [method](#)), 192 [create_crypttab\(\)](#)
[create\(\)](#) [\(kiwi.archive.tar.ArchiveTar](#) [\(kiwi.storage.luks_device.LuksDevice](#)
 [method](#)), 193 [method](#)), 257
[create\(\)](#) [\(kiwi.builder.archive.ArchiveBuilder](#) [create_degraded_raid\(\)](#)
 [method](#)), 213 [\(kiwi.storage.raid_device.RaidDevice](#)
[create\(\)](#) [\(kiwi.builder.container.ContainerBuilder](#) [method](#)), 258
 [method](#)), 214 [create_disk\(\)](#)
[create\(\)](#) [\(kiwi.builder.disk.DiskBuilder](#) [\(kiwi.builder.disk.DiskBuilder](#)
 [method](#)), 215 [method](#)), 215
[create\(\)](#) [\(kiwi.builder.filesystem.FileSystemBuilder](#) [create_disk_format\(\)](#)
 [method](#)), 216 [\(kiwi.builder.disk.DiskBuilder](#)
[create\(\)](#) [\(kiwi.builder.live.LiveImageBuilder](#) [method](#)), 215
 [method](#)), 218 [create_efi_csm_partition\(\)](#)
[create\(\)](#) [\(kiwi.builder.pxe.PxeBuilder](#) [\(kiwi.storage.disk.Disk](#) [method](#)), 254
 [method](#)), 219 [create_efi_partition\(\)](#)
[create\(\)](#) [\(kiwi.container.oci.ContainerImageOCI](#) [\(kiwi.storage.disk.Disk](#) [method](#)),
 [method](#)), 220 [254](#)
[create\(\)](#) [\(kiwi.partitionner.base.PartitionerBase](#) [create_efi_path\(\)](#)
 [method](#)), 240 [\(kiwi.bootloader.config.base.BootLoaderConfigBase](#)
[create\(\)](#) [\(kiwi.partitionner.dasd.PartitionerDasd](#) [method](#)), 199
 [method](#)), 241 [create_fstab\(\)](#)
[create\(\)](#) [\(kiwi.partitionner.gpt.PartitionerGpt](#) [\(kiwi.system.setup.SystemSetup](#)
 [method](#)), 242 [method](#)), 280
[create\(\)](#) [\(kiwi.partitionner.msdos.PartitionerMsDos](#) [create_gnu_gzip_compressed\(\)](#)
 [method](#)), 242 [\(kiwi.archive.tar.ArchiveTar](#) [method](#)),
[create\(\)](#) [\(kiwi.path.Path](#) [static method](#)), [193](#)
 171 [create_header_end_block\(\)](#)
[create\(\)](#) [\(kiwi.storage.loop_device.LoopDevice](#) [\(kiwi.iso_tools.iso.Iso](#) [method](#)),
 [method](#)), 256 [231](#)
[create\(\)](#) [\(kiwi.system.profile.Profile](#) [create_header_end_marker\(\)](#)
 [method](#)), 276 [\(kiwi.iso_tools.iso.Iso](#) [method](#)), 231
[create\(\)](#) [\(kiwi.system.root_init.RootInit](#) [create_hybrid\(\)](#) [\(kiwi.iso_tools.iso.Iso](#)
 [method](#)), 279 [static method](#)), 231
[create_boot_partition\(\)](#) [create_hybrid_mbr\(\)](#)
 [\(kiwi.storage.disk.Disk](#) [method](#)), [\(kiwi.storage.disk.Disk](#) [method](#)),
 254 [254](#)
[create_box_img\(\)](#) [create_image_format\(\)](#)
 [\(kiwi.storage.subformat.vagrant_base.DiskFormatVagrantBase](#) [\(kiwi.storage.subformat.base.DiskFormatBase](#)
 [method](#)), 264 [method](#)), 260
[create_box_img\(\)](#) [create_image_format\(\)](#)
 [\(kiwi.storage.subformat.vagrant_libvirt.DiskFormatVagrantLibvirt](#) [\(kiwi.storage.subformat.gce.DiskFormatGce](#)
 [method](#)), 266 [method](#)), 262
[create_box_img\(\)](#) [create_image_format\(\)](#)
 [\(kiwi.storage.subformat.vagrant_virtualbox.DiskFormatVagrantVirtualBox](#) [\(kiwi.storage.subformat.ova.DiskFormatOva](#)
 [method](#)), 266 [method](#)), 262

<i>method</i>), 262	<i>method</i>), 230
<code>create_image_format()</code> (<i>kiwi.storage.subformat.qcow2.DiskFormatQcow2</i> <i>method</i>), 263	<code>create_match_method()</code> (<i>kiwi.command_process.CommandProcess</i> <i>method</i>), 140
<code>create_image_format()</code> (<i>kiwi.storage.subformat.vagrant_base.DiskFormatVagrantBase</i> <i>method</i>), 264	<code>create_mbr()</code> (<i>kiwi.storage.disk.Disk</i> <i>method</i>), 154
<code>create_image_format()</code> (<i>kiwi.storage.subformat.vdi.DiskFormatVdi</i> <i>method</i>), 267	<code>create_on_device()</code> (<i>kiwi.filesystem.base.FileSystemBase</i> <i>method</i>), 223
<code>create_image_format()</code> (<i>kiwi.storage.subformat.vhd.DiskFormatVhd</i> <i>method</i>), 267	<code>create_on_device()</code> (<i>kiwi.filesystem.btrfs.FileSystemBtrfs</i> <i>method</i>), 224
<code>create_image_format()</code> (<i>kiwi.storage.subformat.vhdfixed.DiskFormatVhdfixed</i> <i>method</i>), 268	<code>create_on_device()</code> (<i>kiwi.filesystem.ext2.FileSystemExt2</i> <i>method</i>), 225
<code>create_image_format()</code> (<i>kiwi.storage.subformat.vhdx.DiskFormatVhdx</i> <i>method</i>), 268	<code>create_on_device()</code> (<i>kiwi.filesystem.ext3.FileSystemExt3</i> <i>method</i>), 225
<code>create_image_format()</code> (<i>kiwi.storage.subformat.vmdk.DiskFormatVmdk</i> <i>method</i>), 269	<code>create_on_device()</code> (<i>kiwi.filesystem.ext4.FileSystemExt4</i> <i>method</i>), 225
<code>create_init_link_from_linuxrc()</code> (<i>kiwi.system.setup.SystemSetup</i> <i>method</i>), 281	<code>create_on_device()</code> (<i>kiwi.filesystem.fat16.FileSystemFat16</i> <i>method</i>), 226
<code>create_initrd()</code> (<i>kiwi.boot.image.base.BootImageBase</i> <i>method</i>), 194	<code>create_on_device()</code> (<i>kiwi.filesystem.fat32.FileSystemFat32</i> <i>method</i>), 226
<code>create_initrd()</code> (<i>kiwi.boot.image.builtin_kiwi.BootImageKiwi</i> <i>method</i>), 198	<code>create_on_device()</code> (<i>kiwi.filesystem.xfs.FileSystemXfs</i> <i>method</i>), 228
<code>create_initrd()</code> (<i>kiwi.boot.image.dracut.BootImageDracut</i> <i>method</i>), 196	<code>create_on_file()</code> (<i>kiwi.filesystem.base.FileSystemBase</i> <i>method</i>), 223
<code>create_install_iso()</code> (<i>kiwi.builder.install.InstallImageBuilder</i> <i>method</i>), 217	<code>create_on_file()</code> (<i>kiwi.filesystem.clicfs.FileSystemClicFs</i> <i>method</i>), 224
<code>create_install_media()</code> (<i>kiwi.builder.disk.DiskBuilder</i> <i>method</i>), 216	<code>create_on_file()</code> (<i>kiwi.filesystem.iso9660.FileSystemIso9660</i> <i>method</i>), 226
<code>create_install_pxe_archive()</code> (<i>kiwi.builder.install.InstallImageBuilder</i> <i>method</i>), 217	<code>create_on_file()</code> (<i>kiwi.filesystem.squashfs.FileSystemSquashFs</i> <i>method</i>), 227
<code>create_iso()</code> (<i>kiwi.iso_tools.base.IsoToolsBase</i> <i>method</i>), 229	<code>create_prep_partition()</code> (<i>kiwi.storage.disk.Disk</i> <i>method</i>), 255
<code>create_iso()</code> (<i>kiwi.iso_tools.cdrtools.IsoToolsCdrTools</i> <i>method</i>), 258	<code>create_raid_config()</code> (<i>kiwi.storage.raid_device.RaidDevice</i> <i>method</i>), 258

`create_random_keyfile()` (*kiwi.storage.luks_device.LuksDevice static method*), 257
`create_recovery_archive()` (*kiwi.system.setup.SystemSetup method*), 281
`create_repository_solvable()` (*kiwi.solver.repository.base.SolverRepositoryBase method*), 250
`create_root_lvm_partition()` (*kiwi.storage.disk.Disk method*), 255
`create_root_partition()` (*kiwi.storage.disk.Disk method*), 255
`create_root_raid_partition()` (*kiwi.storage.disk.Disk method*), 255
`create_root_readonly_partition()` (*kiwi.storage.disk.Disk method*), 255
`create_spare_partition()` (*kiwi.storage.disk.Disk method*), 255
`create_swap_partition()` (*kiwi.storage.disk.Disk method*), 255
`create_volume_paths_in_root_dir()` (*kiwi.volume_manager.base.VolumeManagerBase method*), 294
`create_volumes()` (*kiwi.volume_manager.base.VolumeManagerBase method*), 294
`create_volumes()` (*kiwi.volume_manager.btrfs.VolumeManagerBtrfs method*), 296
`create_volumes()` (*kiwi.volume_manager.lvm.VolumeManagerLVM method*), 297
`create_xz_compressed()` (*kiwi.archive.tar.ArchiveTar method*), 193
`credentials_file_name()` (*kiwi.system.uri.Uri method*), 284
`customize()` (*kiwi.system.size.SystemSize method*), 284
D
`database_consistent()` (*kiwi.package_manager.base.PackageManagerBase method*), 233
`DataSync` (*class in kiwi.utils.sync*), 292
`deactivate_bootloader_setup()` (*kiwi.container.setup.base.ContainerSetupBase method*), 221
`deactivate_root_filesystem_check()` (*kiwi.container.setup.base.ContainerSetupBase method*), 221
`deactivate_systemd_service()` (*kiwi.container.setup.base.ContainerSetupBase method*), 221
`DebugFilter` (*class in kiwi.logger_filter*), 168
`Defaults` (*class in kiwi.defaults*), 140
`delete()` (*kiwi.system.profile.Profile method*), 276
`delete()` (*kiwi.system.root_init.RootInit method*), 279
`delete_all_repos()` (*kiwi.repository.base.RepositoryBase method*), 244
`delete_all_repos()` (*kiwi.repository.dnf.RepositoryDnf method*), 246
`delete_all_repos()` (*kiwi.repository.zypper.RepositoryZypper method*), 248
`delete_packages()` (*kiwi.system.prepare.SystemPrepare method*), 274
`delete_repo()` (*kiwi.repository.base.RepositoryBase method*), 244
`delete_repo()` (*kiwi.repository.dnf.RepositoryDnf method*), 246
`delete_repo()` (*kiwi.repository.zypper.RepositoryZypper method*), 248
`delete_repo_cache()` (*kiwi.repository.base.RepositoryBase method*), 245
`delete_repo_cache()` (*kiwi.repository.dnf.RepositoryDnf method*), 246
`delete_repo_cache()` (*kiwi.repository.zypper.RepositoryZypper method*), 248
`delete_repository_sections()`

(*kiwi.xml_state.XMLState* method),
180
delete_repository_sections_used_for_install(*kiwi.xml_state.XMLState* method),
180
DeviceProvider (class in *kiwi.storage.device_provider*), 253
disable_cleanup()
(*kiwi.boot.image.base.BootImageBase* method), 194
Disk (class in *kiwi.storage.disk*), 254
DiskBuilder (class in *kiwi.builder.disk*),
215
DiskFormat (class in *kiwi.storage.subformat*), 269
DiskFormatBase (class in *kiwi.storage.subformat.base*), 260
DiskFormatGce (class in *kiwi.storage.subformat.gce*), 262
DiskFormatOva (class in *kiwi.storage.subformat.ova*), 262
DiskFormatQcow2 (class in *kiwi.storage.subformat.qcow2*),
263
DiskFormatVagrantBase (class in *kiwi.storage.subformat.vagrant_base*),
264
DiskFormatVagrantLibVirt (class in *kiwi.storage.subformat.vagrant_libvirt*),
266
DiskFormatVagrantVirtualBox (class in *kiwi.storage.subformat.vagrant_virtualbox*),
266
DiskFormatVdi (class in *kiwi.storage.subformat.vdi*), 267
DiskFormatVhd (class in *kiwi.storage.subformat.vhd*), 267
DiskFormatVhdFixed (class in *kiwi.storage.subformat.vhdfixed*),
268
DiskFormatVhdx (class in *kiwi.storage.subformat.vhdx*), 268
DiskFormatVmdk (class in *kiwi.storage.subformat.vmdk*), 269
DiskSetup (class in *kiwi.storage.setup*),
259
download_from_repository()
(*kiwi.solver.repository.base.SolverRepositoryBase* method), 251
dump(*kiwi.boot.image.base.BootImageBase* method), 194
dump()
(*kiwi.system.result.Result* method), 277
dump_reload_package_database()
(*kiwi.package_manager.base.PackageManagerBase* method), 233
E
ec2_mode()
(*kiwi.firmware.FirmWare* method), 164
efi_mode()
(*kiwi.firmware.FirmWare* method), 164
enable_cleanup()
(*kiwi.boot.image.base.BootImageBase* method), 194
ErrorFilter (class in *kiwi.logger_filter*),
168
export_modprobe_setup()
(*kiwi.system.setup.SystemSetup* method), 281
export_package_list()
(*kiwi.system.setup.SystemSetup* method), 281
export_package_verification()
(*kiwi.system.setup.SystemSetup* method), 281
extract()
(*kiwi.archive.cpio.ArchiveCpio* method), 192
extract()
(*kiwi.archive.tar.ArchiveTar* method), 193
extras()
(in module *kiwi.kiwi*), 166
F
failsafe_boot_entry_requested()
(*kiwi.bootloader.config.base.BootLoaderConfigBase* method), 199
filename()
(*kiwi.system.result.result_file_type* property), 277
FileSystem (class in *kiwi.filesystem*), 228
FileSystemBase (class in *kiwi.filesystem.base*), 222
FileSystemBtrfs (class in *kiwi.filesystem.btrfs*), 224
FileSystemBuilder (class in *kiwi.builder.filesystem*), 216

FileSystemClicFs	(class in <i>kiwi.filesystem.clicfs</i>), 224	(<i>kiwi.storage.subformat.vagrant_libvirt.DiskFormatV</i> method), 266
FileSystemExt2	(class in <i>kiwi.filesystem.ext2</i>), 225	<i>get_additional_vagrant_config_settings()</i> (<i>kiwi.storage.subformat.vagrant_base.DiskFormatV</i> method), 265
FileSystemExt3	(class in <i>kiwi.filesystem.ext3</i>), 225	<i>get_additional_vagrant_config_settings()</i> (<i>kiwi.storage.subformat.vagrant_libvirt.DiskFormatV</i> method), 266
FileSystemExt4	(class in <i>kiwi.filesystem.ext4</i>), 225	<i>get_additional_vagrant_config_settings()</i> (<i>kiwi.storage.subformat.vagrant_virtualbox.DiskForm</i> method), 266
FileSystemFat16	(class in <i>kiwi.filesystem.fat16</i>), 226	<i>get_archive_image_types()</i> (<i>kiwi.defaults.Defaults</i> static method), 140
FileSystemFat32	(class in <i>kiwi.filesystem.fat32</i>), 226	<i>get_bios_image_name()</i> (<i>kiwi.defaults.Defaults</i> static method), 140
FileSystemIsoFs	(class in <i>kiwi.filesystem.isoFs</i>), 226	<i>get_bios_module_directory_name()</i> (<i>kiwi.defaults.Defaults</i> static method), 141
FileSystemSetup	(class in <i>kiwi.filesystem.setup</i>), 227	<i>get_blkid()</i> (<i>kiwi.utils.block.BlockID</i> method), 290
FileSystemSquashFs	(class in <i>kiwi.filesystem.squashfs</i>), 227	<i>get_boot_cmdline()</i> (<i>kiwi.bootloader.config.base.BootLoaderConfigBase</i> method), 199
FileSystemXfs	(class in <i>kiwi.filesystem.xfs</i>), 228	<i>get_boot_description_directory()</i> (<i>kiwi.boot.image.base.BootImageBase</i> method), 194
filter()	(<i>kiwi.logger_filter.DebugFilter</i> method), 168	<i>get_boot_image_description_path()</i> (<i>kiwi.defaults.Defaults</i> static method), 141
filter()	(<i>kiwi.logger_filter.ErrorFilter</i> method), 168	<i>get_boot_image_strip_file()</i> (<i>kiwi.defaults.Defaults</i> static method), 141
filter()	(<i>kiwi.logger_filter.InfoFilter</i> method), 169	<i>get_boot_label()</i> (<i>kiwi.storage.setup.DiskSetup</i> method), 259
filter()	(<i>kiwi.logger_filter.LoggerSchedulerFilter</i> method), 169	<i>get_boot_names()</i> (<i>kiwi.boot.image.base.BootImageBase</i> method), 194
filter()	(<i>kiwi.logger_filter.WarningFilter</i> method), 169	<i>get_boot_path()</i> (<i>kiwi.bootloader.config.base.BootLoaderConfigBase</i> method), 199
FirmWare	(class in <i>kiwi.firmware</i>), 164	<i>get_boot_theme()</i> (<i>kiwi.bootloader.config.base.BootLoaderConfigBase</i> method), 199
fix_boot_catalog()	(<i>kiwi.iso_tools.iso.Iso</i> static method), 231	<i>get_boot_timeout_seconds()</i> (<i>kiwi.bootloader.config.base.BootLoaderConfigBase</i> method), 199
format()	(<i>kiwi.logger_color_formatter.ColorFormatter</i> method), 167	
format_message()	(<i>kiwi.logger_color_formatter.ColorMessage</i> method), 168	
G		
get()	(<i>kiwi.defaults.Defaults</i> method), 140	
get()	(<i>kiwi.utils.sysconfig.SysConfig</i> method), 293	
get_additional_metadata()	(<i>kiwi.storage.subformat.vagrant_base.DiskFormatV</i> method), 265	
get_additional_metadata()		

method), 199	182
get_bootstrap_archives() (kiwi.xml_state.XMLState method), 180	get_build_type_vagrant_config_section() (kiwi.xml_state.XMLState method), 182
get_bootstrap_collection_type() (kiwi.xml_state.XMLState method), 180	get_build_type_vmconfig_entries() (kiwi.xml_state.XMLState method), 182
get_bootstrap_collections() (kiwi.xml_state.XMLState method), 180	get_build_type_vmdisk_section() (kiwi.xml_state.XMLState method), 183
get_bootstrap_packages() (kiwi.xml_state.XMLState method), 181	get_build_type_vmdvd_section() (kiwi.xml_state.XMLState method), 183
get_bootstrap_packages_sections() (kiwi.xml_state.XMLState method), 181	get_build_type_vmnics_entries() (kiwi.xml_state.XMLState method), 183
get_bootstrap_products() (kiwi.xml_state.XMLState method), 181	get_buildservice_env_name() (kiwi.defaults.Defaults static method), 141
get_build_type_containerconfig_section() (kiwi.xml_state.XMLState method), 181	get_build_type_compression() (kiwi.runtime_config.RuntimeConfig method), 175
get_build_type_format_options() (kiwi.xml_state.XMLState method), 181	get_byte_size() (kiwi.storage.device_provider.DeviceProvider method), 253
get_build_type_machine_section() (kiwi.xml_state.XMLState method), 181	get_canonical_volume_list() (kiwi.volume_manager.base.VolumeManagerBase method), 294
get_build_type_name() (kiwi.xml_state.XMLState method), 181	get_collection_type() (kiwi.xml_state.XMLState method), 183
get_build_type_oemconfig_section() (kiwi.xml_state.XMLState method), 181	get_collections() (kiwi.xml_state.XMLState method), 183
get_build_type_size() (kiwi.xml_state.XMLState method), 182	get_command() (kiwi.cli.Cli method), 136 get_command_args() (kiwi.cli.Cli method), 136
get_build_type_spare_part_fs_attributes() (kiwi.xml_state.XMLState method), 182	get_container_functions_file() (kiwi.defaults.Defaults static method), 141
get_build_type_spare_part_size() (kiwi.xml_state.XMLState method), 182	get_container_base_image_tag() (kiwi.defaults.Defaults static method), 141
get_build_type_system_disk_section() (kiwi.xml_state.XMLState method), 182	get_container_compression() (kiwi.defaults.Defaults static method), 141
get_build_type_unpartitioned_bytes() (kiwi.xml_state.XMLState method),	get_container_compression() (kiwi.runtime_config.RuntimeConfig

<i>method</i>), 175	<i>method</i>), 143
get_container_config() (<i>kiwi.xml_state.XMLState</i> <i>method</i>), 183	get_default_firmware() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 143
get_container_image_types() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 141	get_default_inode_size() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 143
get_container_name() (<i>kiwi.container.setup.base.ContainerSetupBase</i> <i>static</i> <i>method</i>), 221	get_default_legacy_bios_mbytes() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 143
get_continue_on_timeout() (<i>kiwi.bootloader.config.base.BootLoaderConfigBase</i> <i>static</i> <i>method</i>), 200	get_default_live_iso_root_filesystem() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 143
get_custom_rpm_bootstrap_macro_name() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 142	get_default_live_iso_type() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 143
get_custom_rpm_image_macro_name() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 142	get_default_packager_tool() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 144
get_custom_rpm_macros_path() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 142	get_default_prep_mbytes() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 144
get_default_boot_mbytes() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 142	get_default_uri_type() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 144
get_default_boot_timeout_seconds() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 142	get_default_video_mode() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 144
get_default_container_created_by() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 142	get_default_volume_group_name() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 144
get_default_container_name() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 142	get_derived_from_image_uri() (<i>kiwi.xml_state.XMLState</i> <i>method</i>), 183
get_default_container_subcommand() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 142	get_description_section() (<i>kiwi.xml_state.XMLState</i> <i>method</i>), 183
get_default_container_tag() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 143	get_device() (<i>kiwi.storage.device_provider.DeviceProvider</i> <i>method</i>), 253
get_default_disk_start_sector() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 143	get_device() (<i>kiwi.storage.disk.Disk</i> <i>method</i>), 255
get_default_efi_boot_mbytes() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 143	get_device() (<i>kiwi.storage.loop_device.LoopDevice</i> <i>method</i>), 256
get_default_efi_partition_table_type() (<i>kiwi.defaults.Defaults</i> <i>static</i> <i>method</i>), 257	get_device() (<i>kiwi.storage.luks_device.LuksDevice</i> <i>method</i>), 257

<code>get_device()</code> (<i>kiwi.storage.mapped_device.MappedDevice</i> <i>method</i>), 258	<code>get_efi_module_directory_name()</code> (<i>kiwi.defaults.Defaults</i> <i>static method</i>), 145
<code>get_device()</code> (<i>kiwi.storage.raid_device.RaidDevice</i> <i>method</i>), 259	<code>get_efi_partition_size()</code> (<i>kiwi.firmware.FirmWare</i> <i>method</i>), 164
<code>get_device()</code> (<i>kiwi.volume_manager.base.VolumeManagerBase</i> <i>method</i>), 294	<code>get_error_code()</code> (<i>kiwi.command_process.CommandIterator</i> <i>method</i>), 139
<code>get_device()</code> (<i>kiwi.volume_manager.lvm.VolumeManagerLVM</i> <i>method</i>), 297	<code>get_error_output()</code> (<i>kiwi.command_process.CommandIterator</i> <i>method</i>), 139
<code>get_disabled_runtime_checks()</code> (<i>kiwi.runtime_config.RuntimeConfig</i> <i>method</i>), 175	<code>get_exclude_list_for_non_physical_devices()</code> (<i>kiwi.defaults.Defaults</i> <i>method</i>), 145
<code>get_disk_format_types()</code> (<i>kiwi.defaults.Defaults</i> <i>static method</i>), 144	<code>get_exclude_list_for_root_data_sync()</code> (<i>kiwi.defaults.Defaults</i> <i>static method</i>), 145
<code>get_disk_image_types()</code> (<i>kiwi.defaults.Defaults</i> <i>static method</i>), 144	<code>get_extension_xml_data()</code> (<i>kiwi.xml_description.XMLDescription</i> <i>method</i>), 178
<code>get_disk_start_sector()</code> (<i>kiwi.xml_state.XMLState</i> <i>method</i>), 184	<code>get_failsafe_kernel_options()</code> (<i>kiwi.defaults.Defaults</i> <i>static method</i>), 145
<code>get_disksize_mbytes()</code> (<i>kiwi.storage.setup.DiskSetup</i> <i>method</i>), 259	<code>get_filesystem()</code> (<i>kiwi.utils.block.BlockID</i> <i>method</i>), 290
<code>get_distribution_name_from_boot_attr()</code> (<i>kiwi.xml_state.XMLState</i> <i>method</i>), 184	<code>get_filesystem_image_types()</code> (<i>kiwi.defaults.Defaults</i> <i>static method</i>), 146
<code>get_dracut_conf_name()</code> (<i>kiwi.defaults.Defaults</i> <i>static method</i>), 144	<code>get_firmware_types()</code> (<i>kiwi.defaults.Defaults</i> <i>static method</i>), 146
<code>get_drivers_list()</code> (<i>kiwi.xml_state.XMLState</i> <i>method</i>), 184	<code>get_format()</code> (<i>kiwi.utils.compress.Compress</i> <i>method</i>), 292
<code>get_ec2_capable_firmware_names()</code> (<i>kiwi.defaults.Defaults</i> <i>static method</i>), 145	<code>get_fragment()</code> (<i>kiwi.system.uri.Uri</i> <i>method</i>), 285
<code>get_efi_capable_firmware_names()</code> (<i>kiwi.defaults.Defaults</i> <i>static method</i>), 145	<code>get_fs_create_option_list()</code> (<i>kiwi.xml_state.XMLState</i> <i>method</i>), 184
<code>get_efi_image_name()</code> (<i>kiwi.defaults.Defaults</i> <i>static method</i>), 145	<code>get_fs_mount_option_list()</code> (<i>kiwi.xml_state.XMLState</i> <i>method</i>), 184
<code>get_efi_label()</code> (<i>kiwi.storage.setup.DiskSetup</i> <i>method</i>), 259	<code>get_fstab()</code> (<i>kiwi.volume_manager.base.VolumeManagerBase</i> <i>method</i>), 294
	<code>get_fstab()</code> (<i>kiwi.volume_manager.btrfs.VolumeManagerBtrfs</i> <i>method</i>), 294

<i>method</i>), 296	<i>get_imported_root_image()</i>
<i>get_fstab()</i>	<i>(kiwi.defaults.Defaults static</i>
<i>(kiwi.volume_manager.lvm.VolumeManagerLVMmethod), 147</i>	<i>method</i>), 297
<i>method</i>), 297	<i>get_initrd_system()</i>
<i>get_gfxmode()</i>	<i>(kiwi.xml_state.XMLState method),</i>
<i>(kiwi.bootloader.config.base.BootLoaderConfigBase</i>	<i>method</i>), 200
<i>method</i>), 200	<i>get_install_image_boot_default()</i>
<i>get_global_args()</i>	<i>(kiwi.cli.Cli (kiwi.bootloader.config.base.BootLoaderConfigBase</i>
<i>method</i>), 137	<i>method</i>), 200
<i>get_grub_basic_modules()</i>	<i>get_install_message_template()</i>
<i>(kiwi.defaults.Defaults static</i>	<i>(kiwi.bootloader.template.isolinux.BootLoaderTempl</i>
<i>method</i>), 146	<i>method</i>), 211
<i>get_grub_bios_core_loader()</i>	<i>get_install_template()</i>
<i>(kiwi.defaults.Defaults static</i>	<i>(kiwi.bootloader.template.grub2.BootLoaderTemplat</i>
<i>method</i>), 146	<i>method</i>), 210
<i>get_grub_bios_modules()</i>	<i>get_install_template()</i>
<i>(kiwi.defaults.Defaults static</i>	<i>(kiwi.bootloader.template.isolinux.BootLoaderTempl</i>
<i>method</i>), 146	<i>method</i>), 211
<i>get_grub_boot_directory_name()</i>	<i>get_install_volume_id()</i>
<i>(kiwi.defaults.Defaults static</i>	<i>(kiwi.defaults.Defaults static</i>
<i>method</i>), 146	<i>method</i>), 148
<i>get_grub_efi_font_directory()</i>	<i>get_iso_boot_path()</i>
<i>(kiwi.defaults.Defaults static</i>	<i>(kiwi.defaults.Defaults static</i>
<i>method</i>), 147	<i>method</i>), 148
<i>get_grub_efi_modules()</i>	<i>get_iso_template()</i>
<i>(kiwi.defaults.Defaults static</i>	<i>(kiwi.bootloader.template.grub2.BootLoaderTemplat</i>
<i>method</i>), 147	<i>method</i>), 210
<i>get_grub_ofw_modules()</i>	<i>get_iso_tool_category()</i>
<i>(kiwi.defaults.Defaults static</i>	<i>(kiwi.defaults.Defaults static</i>
<i>method</i>), 147	<i>method</i>), 148
<i>get_grub_path()</i>	<i>get_iso_tool_category()</i>
<i>(kiwi.defaults.Defaults static method), 147</i>	<i>(kiwi.runtime_config.RuntimeConfig</i>
<i>get_host_template()</i>	<i>method</i>), 176
<i>(kiwi.repository.template.apr.PackageManagerTemplateAprGet</i>	<i>get_legacy_bios_grub_loader()</i>
<i>method</i>), 250	<i>(kiwi.defaults.Defaults static</i>
<i>get_id()</i>	<i>method</i>), 148
<i>(kiwi.partitioner.base.PartitionerBase</i>	<i>get_kernel()</i>
<i>method</i>), 240	<i>(kiwi.system.kernel.Kernel</i>
<i>get_id()</i>	<i>method</i>), 273
<i>(kiwi.system.identifier.SystemIdentifier</i>	<i>get_label()</i>
<i>method</i>), 272	<i>(kiwi.utils.block.BlockID</i>
<i>get_image_packages_sections()</i>	<i>method</i>), 290
<i>(kiwi.xml_state.XMLState method),</i>	<i>get_legacy_bios_partition_size()</i>
<i>184</i>	<i>(kiwi.firmware.FirmWare method),</i>
<i>get_image_template()</i>	<i>164</i>
<i>(kiwi.repository.template.apr.PackageManagerTemplateAprGet</i>	<i>get_module_from_flag()</i>
<i>method</i>), 250	<i>(kiwi.defaults.Defaults static</i>
<i>get_image_version()</i>	<i>method</i>), 148
<i>(kiwi.xml_state.XMLState method),</i>	<i>get_live_image_types()</i>
<i>184</i>	<i>(kiwi.defaults.Defaults static</i>

<code>method</code>), 148	<code>(kiwi.defaults.Defaults</code>	<code>static</code>
<code>get_live_iso_persistent_boot_options()</code>	<code>method</code>), 149	
<code>(kiwi.defaults.Defaults</code>	<code>static</code>	<code>get_obs_download_server_url()</code>
<code>method</code>), 148		<code>(kiwi.runtime_config.RuntimeConfig</code>
<code>get_locale()</code>	<code>(kiwi.xml_state.XMLState</code>	<code>method</code>), 176
<code>method</code>), 185		<code>get_oci_archive_tool()</code>
<code>get_logfile()</code>	<code>(kiwi.logger.Logger</code>	<code>(kiwi.defaults.Defaults</code>
<code>method</code>), 167	<code>method</code>), 149	<code>static</code>
<code>get_luks_key_length()</code>	<code>get_oci_archive_tool()</code>	
<code>(kiwi.defaults.Defaults</code>	<code>(kiwi.runtime_config.RuntimeConfig</code>	
<code>static</code>	<code>method</code>), 176	
<code>method</code>), 149		<code>get_oemconfig_oem_multipath_scan()</code>
<code>get_lvm_overhead_mbytes()</code>	<code>get_oemconfig_swap_mbytes()</code>	<code>(kiwi.xml_state.XMLState</code>
<code>(kiwi.defaults.Defaults</code>	<code>(kiwi.xml_state.XMLState</code>	<code>method</code>),
<code>static</code>		185
<code>method</code>), 149		
<code>get_max_size_constraint()</code>	<code>get_package_manager()</code>	
<code>(kiwi.runtime_config.RuntimeConfig</code>	<code>(kiwi.xml_state.XMLState</code>	<code>method</code>),
<code>method</code>), 176		185
	<code>get_package_sections()</code>	
<code>get_menu_entry_install_title()</code>	<code>(kiwi.xml_state.XMLState</code>	<code>method</code>),
<code>(kiwi.bootloader.config.base.BootLoaderConfigBase</code>		185
<code>method</code>), 200		
<code>get_menu_entry_title()</code>	<code>get_packages_sections()</code>	
<code>(kiwi.bootloader.config.base.BootLoaderConfigBase</code>	<code>(kiwi.xml_state.XMLState</code>	<code>method</code>),
<code>method</code>), 200		185
	<code>get_partition_table_type()</code>	
<code>get_message_template()</code>	<code>(kiwi.firmware.FirmWare</code>	<code>method</code>),
<code>(kiwi.bootloader.template.isolinux.BootLoaderTemplateIsoLinux</code>		164
<code>method</code>), 212		
	<code>get_pid()</code>	<code>(kiwi.command_process.CommandIterator</code>
<code>get_min_partition_mbytes()</code>	<code>method</code>), 139	
<code>(kiwi.defaults.Defaults</code>	<code>get_preferences_sections()</code>	
<code>static</code>	<code>(kiwi.xml_state.XMLState</code>	<code>method</code>),
<code>method</code>), 149		186
	<code>get_prep_partition_size()</code>	
<code>get_multiboot_install_template()</code>	<code>(kiwi.firmware.FirmWare</code>	<code>method</code>),
<code>(kiwi.bootloader.template.grub2.BootLoaderTemplateGrub2</code>		164
<code>method</code>), 211		
<code>get_multiboot_install_template()</code>	<code>get_preparer()</code>	<code>(kiwi.defaults.Defaults</code>
<code>(kiwi.bootloader.template.isolinux.BootLoaderTemplateIsoLinux</code>	<code>static</code>	<code>method</code>), 149
<code>method</code>), 212		
<code>get_multiboot_iso_template()</code>	<code>get_public_partition_id_map()</code>	
<code>(kiwi.bootloader.template.grub2.BootLoaderTemplateGrub2</code>	<code>(kiwi.storage.disk.Disk</code>	<code>method</code>), 256
<code>method</code>), 211		
<code>get_multiboot_template()</code>	<code>get_publisher()</code>	<code>(kiwi.defaults.Defaults</code>
<code>(kiwi.bootloader.template.isolinux.BootLoaderTemplateIsoLinux</code>		
<code>method</code>), 212		
<code>get_network_image_types()</code>		
<code>(kiwi.defaults.Defaults</code>		
<code>static</code>		
<code>method</code>), 149		
<code>get_obs_download_server_url()</code>		

<i>static method</i>), 149	(<i>kiwi.defaults.Defaults</i> <i>static</i>
<code>get_qemu_option_list()</code>	<i>method</i>), 150
(<i>kiwi.storage.subformat.base.DiskFormatBase</i>	<code>get_shim_loader()</code>
<i>method</i>), 260	(<i>kiwi.defaults.Defaults</i> <i>static</i>
<code>get_recovery_spare_mbytes()</code>	<i>method</i>), 150
(<i>kiwi.defaults.Defaults</i> <i>static</i>	<code>get_shim_vendor_directory()</code>
<i>method</i>), 150	(<i>kiwi.defaults.Defaults</i> <i>static</i>
<code>get_repository_sections()</code>	<i>method</i>), 150
(<i>kiwi.xml_state.XMLState</i> <i>method</i>),	<code>get_signed_grub_loader()</code>
186	(<i>kiwi.defaults.Defaults</i> <i>static</i>
<code>get_repository_sections_used_for_build()</code>	<i>method</i>), 151
(<i>kiwi.xml_state.XMLState</i> <i>method</i>),	<code>get_size_mbytes()</code>
186	(<i>kiwi.filesystem.setup.FileSystemSetup</i>
<code>get_repository_sections_used_in_image()</code>	<i>method</i>), 227
(<i>kiwi.xml_state.XMLState</i> <i>method</i>),	<code>get_snapper_config_template_file()</code>
186	(<i>kiwi.defaults.Defaults</i> <i>static</i>
<code>get_results()</code> (<i>kiwi.system.result.Result</i>	<i>method</i>), 151
<i>method</i>), 277	<code>get_solvable_location()</code>
<code>get_root_label()</code>	(<i>kiwi.defaults.Defaults</i> <i>static</i>
(<i>kiwi.storage.setup.DiskSetup</i>	<i>method</i>), 151
<i>method</i>), 260	<code>get_strip_files_to_delete()</code>
<code>get_root_partition_uuid()</code>	(<i>kiwi.xml_state.XMLState</i> <i>method</i>),
(<i>kiwi.xml_state.XMLState</i> <i>method</i>),	187
187	<code>get_strip_libraries_to_keep()</code>
<code>get_rpm_check_signatures()</code>	(<i>kiwi.xml_state.XMLState</i> <i>method</i>),
(<i>kiwi.xml_state.XMLState</i> <i>method</i>),	187
187	<code>get_strip_list()</code>
<code>get_rpm_excludedocs()</code>	(<i>kiwi.xml_state.XMLState</i> <i>method</i>),
(<i>kiwi.xml_state.XMLState</i> <i>method</i>),	187
187	<code>get_strip_tools_to_keep()</code>
<code>get_rpm_locale()</code>	(<i>kiwi.xml_state.XMLState</i> <i>method</i>),
(<i>kiwi.xml_state.XMLState</i> <i>method</i>),	187
187	<code>get_swapsize_mbytes()</code>
<code>get_rpm_locale_filtering()</code>	(<i>kiwi.defaults.Defaults</i> <i>static</i>
(<i>kiwi.xml_state.XMLState</i> <i>method</i>),	<i>method</i>), 151
187	<code>get_syslinux_modules()</code>
<code>get_s390_disk_block_size()</code>	(<i>kiwi.defaults.Defaults</i> <i>static</i>
(<i>kiwi.defaults.Defaults</i> <i>static</i>	<i>method</i>), 151
<i>method</i>), 150	<code>get_syslinux_search_paths()</code>
<code>get_s390_disk_type()</code>	(<i>kiwi.defaults.Defaults</i> <i>static</i>
(<i>kiwi.defaults.Defaults</i> <i>static</i>	<i>method</i>), 151
<i>method</i>), 150	<code>get_system_archives()</code>
<code>get_schema_file()</code>	(<i>kiwi.xml_state.XMLState</i> <i>method</i>),
(<i>kiwi.defaults.Defaults</i> <i>static</i>	188
<i>method</i>), 150	<code>get_system_collection_type()</code>
<code>get_servicename()</code> (<i>kiwi.cli.Cli</i>	(<i>kiwi.xml_state.XMLState</i> <i>method</i>),
<i>method</i>), 137	188
<code>get_shared_cache_location()</code>	<code>get_system_collections()</code>

<code>(kiwi.xml_state.XMLState method),</code>	<code>method), 189</code>
<code>188</code>	<code>get_users_sections()</code>
<code>get_system_ignore_packages()</code>	<code>(kiwi.xml_state.XMLState method),</code>
<code>(kiwi.xml_state.XMLState method),</code>	<code>189</code>
<code>188</code>	<code>get_uuid()</code>
<code>get_system_packages()</code>	<code>(kiwi.storage.device_provider.DeviceProvider</code>
<code>(kiwi.xml_state.XMLState method),</code>	<code>method), 253</code>
<code>188</code>	<code>get_uuid() (kiwi.utils.block.BlockID</code>
<code>get_system_products()</code>	<code>method), 290</code>
<code>(kiwi.xml_state.XMLState method),</code>	<code>get_vagrant_config_virtualbox_guest_addit</code>
<code>188</code>	<code>(kiwi.defaults.Defaults static</code>
<code>get_target_file_path_for_format()</code>	<code>method), 151</code>
<code>(kiwi.storage.subformat.base.DiskFormatBase</code>	<code>get_vagrant_config_virtualbox_guest_addit</code>
<code>method), 261</code>	<code>(kiwi.xml_state.XMLState method),</code>
<code>get_target_file_path_for_format()</code>	<code>189</code>
<code>(kiwi.storage.subformat.gce.DiskFormatGce</code>	<code>get_video_mode_map()</code>
<code>method), 262</code>	<code>(kiwi.defaults.Defaults static</code>
<code>get_template()</code>	<code>method), 152</code>
<code>(kiwi.bootloader.template.isolinux.BootLoaderTemplateIsolinux</code>	<code>get_xml_name()</code>
<code>method), 212</code>	<code>(kiwi.xml_state.XMLState method),</code>
<code>get_template()</code>	<code>189</code>
<code>(kiwi.bootloader.template.zipl.BootLoaderTemplateZipl</code>	<code>get_xml_id() (kiwi.defaults.Defaults</code>
<code>method), 213</code>	<code>static method), 152</code>
<code>get_template()</code>	<code>get_volume_management()</code>
<code>(kiwi.storage.subformat.template.vagrant_config_vmware_settings_template</code>	<code>(kiwi.xml_state.XMLState method),</code>
<code>method), 271</code>	<code>189</code>
<code>get_template()</code>	<code>get_volume_mbsize()</code>
<code>(kiwi.storage.subformat.template.virtualbox_ovf(VirtualBoxOVFTemplateBase.VolumeManagerBase</code>	<code>(kiwi.xml_state.XMLState method),</code>
<code>method), 272</code>	<code>method), 294</code>
<code>get_template()</code>	<code>get_volumes()</code>
<code>(kiwi.storage.subformat.template.vmware_settings_template.VmwareSettingsTemplate.VolumeManagerBase</code>	<code>(kiwi.xml_state.XMLState method), 295</code>
<code>method), 270</code>	<code>get_volumes()</code>
<code>get_to_become_deleted_packages()</code>	<code>(kiwi.xml_state.XMLState method),</code>
<code>(kiwi.xml_state.XMLState method),</code>	<code>(kiwi.volume_manager.btrfs.VolumeManagerBtrfs</code>
<code>188</code>	<code>method), 296</code>
<code>get_tool_name()</code>	<code>get_volumes()</code>
<code>(kiwi.iso_tools.base.IsoToolsBase</code>	<code>(kiwi.volume_manager.lvm.VolumeManagerLVM</code>
<code>method), 229</code>	<code>method), 298</code>
<code>get_tool_name()</code>	<code>get_volumes() (kiwi.xml_state.XMLState</code>
<code>(kiwi.iso_tools.cdrtools.IsoToolsCdrTools</code>	<code>method), 189</code>
<code>method), 230</code>	<code>get_xen_hypervisor()</code>
<code>get_unsigned_grub_loader()</code>	<code>(kiwi.system.kernel.Kernel method),</code>
<code>(kiwi.defaults.Defaults static</code>	<code>273</code>
<code>method), 151</code>	<code>get_xsl_stylesheet_file()</code>
<code>get_user_groups()</code>	<code>(kiwi.defaults.Defaults static</code>
<code>(kiwi.xml_state.XMLState method),</code>	<code>method), 152</code>
<code>189</code>	<code>get_xz_compression_options()</code>
<code>get_users() (kiwi.xml_state.XMLState</code>	<code>(kiwi.defaults.Defaults static</code>

method), 152
 get_xz_options() (kiwi.runtime_config.RuntimeConfig method), 177
 getLogLevel() (kiwi.logger.Logger method), 166
 group_add() (kiwi.system.users.Users method), 285
 group_exists() (kiwi.system.users.Users method), 286
 gzip() (kiwi.utils.compress.Compress method), 292

H

has_failed() (kiwi.package_manager.base.PackageManagerBase method), 233
 has_failed() (kiwi.package_manager.zypper.PackageManagerZypper method), 237
 has_iso_hybrid_capability() (kiwi.iso_tools.base.IsoToolsBase method), 229
 has_iso_hybrid_capability() (kiwi.iso_tools.cdrtools.IsoToolsCdrTools method), 230
 has_raw_disk() (kiwi.storage.subformat.base.DiskFormatBase method), 261
 Help (class in kiwi.help), 165

I

ImageBuilder (class in kiwi.builder), 219
 import_cdroot_files() (kiwi.system.setup.SystemSetup method), 281
 import_description() (kiwi.system.setup.SystemSetup method), 281
 import_image_identifier() (kiwi.system.setup.SystemSetup method), 281
 import_overlay_files() (kiwi.system.setup.SystemSetup method), 281
 import_repositories_marked_as_imageinfo() (kiwi.system.setup.SystemSetup method), 281
 import_system_description_elements() (kiwi.boot.image.base.BootImageBase method), 195
 import_trusted_keys() (kiwi.repository.base.RepositoryBase method), 245
 import_trusted_keys() (kiwi.repository.dnf.RepositoryDnf method), 246
 import_trusted_keys() (kiwi.repository.zypper.RepositoryZypper method), 248
 include_file() (kiwi.boot.image.base.BootImageBase method), 195
 include_file() (kiwi.boot.image.dracut.BootImageDracut method), 197
 include_module() (kiwi.boot.image.base.BootImageBase method), 195
 include_module() (kiwi.boot.image.dracut.BootImageDracut method), 197
 InfoFilter (class in kiwi.logger_filter), 169
 init_iso_creation_parameters() (kiwi.iso_tools.base.IsoToolsBase method), 229
 init_iso_creation_parameters() (kiwi.iso_tools.cdrtools.IsoToolsCdrTools method), 230
 install() (kiwi.bootloader.install.base.BootLoaderInstallBase method), 208
 install() (kiwi.bootloader.install.grub2.BootLoaderInstallGrub2 method), 208
 install() (kiwi.bootloader.install.zipl.BootLoaderInstallZipl method), 209
 install_bootstrap() (kiwi.system.prepare.SystemPrepare method), 274
 install_packages() (kiwi.system.prepare.SystemPrepare method), 274
 install_required() (kiwi.bootloader.install.base.BootLoaderInstallBase method), 208
 install_required()

(kiwi.bootloader.install.grub2.BootLoaderInstallGrub2XmlState method),
 method), 208 190
 install_required() is_xen_server()
 (kiwi.bootloader.install.zipl.BootLoaderInstallZiplKiwiXmlState method),
 method), 209 190
 install_system() Iso (class in kiwi.iso_tools.iso), 231
 (kiwi.system.prepare.SystemPrepare IsoTools (class in kiwi.iso_tools), 232
 method), 275 IsoToolsBase (class in
 InstallImageBuilder (class in kiwi.iso_tools.base), 228
 kiwi.builder.install), 217 IsoToolsCdrTools (class in
 invoke_kiwicompat() (kiwi.cli.Cli kiwi.iso_tools.cdrtools), 229
 method), 137
 is_buildservice_worker() **K**
 (kiwi.defaults.Defaults static Kernel (class in kiwi.system.kernel), 273
 method), 152 kill() (kiwi.command_process.CommandIterator
 is_loop() (kiwi.storage.device_provider.DeviceProvider method), 139
 method), 253 kiwi (module), 192
 is_loop() (kiwi.storage.disk.Disk kiwi.app (module), 136
 method), 256 kiwi.archive (module), 194
 is_loop() (kiwi.storage.loop_device.LoopDevice kiwi.archive.cpio (module), 192
 method), 257 kiwi.archive.tar (module), 192
 is_loop() (kiwi.storage.luks_device.LuksDevice kiwi.boot (module), 213
 method), 257 kiwi.boot.image (module), 198
 is_loop() (kiwi.storage.mapped_device.MappedDevice kiwi.boot.image.base (module), 194
 method), 258 kiwi.boot.image.builtin_kiwi
 is_loop() (kiwi.storage.raid_device.RaidDevice (module), 197
 method), 259 kiwi.boot.image.dracut (module),
 is_loop() (kiwi.volume_manager.base.VolumeManagerBase 196
 method), 295 kiwi.bootloader (module), 210
 is_mounted() kiwi.bootloader.config (module),
 (kiwi.mount_manager.MountManager 207
 method), 170 kiwi.bootloader.config.base
 is_obs_public() (module), 198
 (kiwi.runtime_config.RuntimeConfig kiwi.bootloader.config.grub2
 method), 177 (module), 203
 is_prepared() kiwi.bootloader.config.isolinux
 (kiwi.boot.image.base.BootImageBase (module), 205
 method), 195 kiwi.bootloader.config.zipl
 is_public() (kiwi.system.uri.Uri (module), 206
 method), 285 kiwi.bootloader.install (module),
 is_remote() (kiwi.system.uri.Uri 210
 method), 285 kiwi.bootloader.install.base
 is_uptodate() (module), 208
 (kiwi.solver.repository.base.SolverRepositoryBase kiwi.bootloader.install.grub2
 method), 251 (module), 208
 is_x86_arch() (kiwi.defaults.Defaults static method), 152 kiwi.bootloader.install.zipl
 (module), 209
 is_xen_guest() kiwi.bootloader.template (module),

- 213
- kiwi.bootloader.template.grub2 (module), 210
- kiwi.bootloader.template.isolinux (module), 211
- kiwi.bootloader.template.zipl (module), 213
- kiwi.builder (module), 219
- kiwi.builder.archive (module), 213
- kiwi.builder.container (module), 214
- kiwi.builder.disk (module), 215
- kiwi.builder.filesystem (module), 216
- kiwi.builder.install (module), 217
- kiwi.builder.live (module), 218
- kiwi.builder.pxe (module), 218
- kiwi.cli (module), 136
- kiwi.command (module), 138
- kiwi.command_process (module), 139
- kiwi.container (module), 220
- kiwi.container.oci (module), 219
- kiwi.container.setup (module), 222
- kiwi.container.setup.base (module), 221
- kiwi.container.setup.docker (module), 222
- kiwi.defaults (module), 140
- kiwi.exceptions (module), 153
- kiwi.filesystem (module), 228
- kiwi.filesystem.base (module), 222
- kiwi.filesystem.btrfs (module), 224
- kiwi.filesystem.clicfs (module), 224
- kiwi.filesystem.ext2 (module), 225
- kiwi.filesystem.ext3 (module), 225
- kiwi.filesystem.ext4 (module), 225
- kiwi.filesystem.fat16 (module), 226
- kiwi.filesystem.fat32 (module), 226
- kiwi.filesystem.isofs (module), 226
- kiwi.filesystem.setup (module), 227
- kiwi.filesystem.squashfs (module), 227
- kiwi.filesystem.xfs (module), 228
- kiwi.firmware (module), 164
- kiwi.help (module), 165
- kiwi.iso_tools (module), 232
- kiwi.iso_tools.base (module), 228
- kiwi.iso_tools.cdrtools (module), 229
- kiwi.iso_tools.iso (module), 231
- kiwi.kiwi (module), 166
- kiwi.logger (module), 166
- kiwi.logger_color_formatter (module), 167
- kiwi.logger_filter (module), 168
- kiwi.mount_manager (module), 170
- kiwi.package_manager (module), 239
- kiwi.package_manager.base (module), 232
- kiwi.package_manager.dnf (module), 235
- kiwi.package_manager.zypper (module), 237
- kiwi.partitioner (module), 243
- kiwi.partitioner.base (module), 240
- kiwi.partitioner.dasd (module), 241
- kiwi.partitioner.gpt (module), 242
- kiwi.partitioner.msdos (module), 242
- kiwi.path (module), 170
- kiwi.privileges (module), 172
- kiwi.repository (module), 249
- kiwi.repository.base (module), 243
- kiwi.repository.dnf (module), 245
- kiwi.repository.template (module), 250
- kiwi.repository.template.apt (module), 250
- kiwi.repository.zypper (module), 247
- kiwi.runtime_checker (module), 173
- kiwi.runtime_config (module), 175
- kiwi.solver (module), 253
- kiwi.solver.repository (module), 252
- kiwi.solver.repository.base (module), 250
- kiwi.solver.repository.rpm_dir (module), 251
- kiwi.solver.repository.rpm_md (module), 251
- kiwi.solver.repository.suse (module), 251
- kiwi.solver.sat (module), 252
- kiwi.storage (module), 260

kiwi.storage.device_provider (module), 253	272
kiwi.storage.disk (module), 254	kiwi.system.kernel (module), 273
kiwi.storage.loop_device (module), 256	kiwi.system.prepare (module), 274
kiwi.storage.luks_device (module), 257	kiwi.system.profile (module), 276
kiwi.storage.mapped_device (module), 258	kiwi.system.result (module), 276
kiwi.storage.raid_device (module), 258	kiwi.system.root_bind (module), 278
kiwi.storage.setup (module), 259	kiwi.system.root_init (module), 279
kiwi.storage.subformat (module), 269	kiwi.system.setup (module), 279
kiwi.storage.subformat.base (module), 260	kiwi.system.shell (module), 283
kiwi.storage.subformat.gce (module), 262	kiwi.system.size (module), 283
kiwi.storage.subformat.ova (module), 262	kiwi.system.uri (module), 284
kiwi.storage.subformat.qcow2 (module), 263	kiwi.system.users (module), 285
kiwi.storage.subformat.template (module), 272	kiwi.tasks (module), 290
kiwi.storage.subformat.template.vagrant_base (module), 270	kiwi.tasks.base (module), 287
kiwi.storage.subformat.template.virtio_scsi (module), 272	kiwi.tasks.result_bundle (module), 288
kiwi.storage.subformat.template.virtio_scsi_secureboot (module), 270	kiwi.tasks.result_list (module), 288
kiwi.storage.subformat.template.virtio_scsi_secureboot_secureboot (module), 270	kiwi.tasks.system_build (module), 288
kiwi.storage.subformat.vagrant_base (module), 264	kiwi.tasks.system_create (module), 289
kiwi.storage.subformat.vagrant_lvm (module), 266	kiwi.tasks.system_prepare (module), 289
kiwi.storage.subformat.vagrant_vhdx (module), 266	kiwi.tasks.system_update (module), 289
kiwi.storage.subformat.vdi (module), 267	kiwi.tasks.virtio_scsi (module), 293
kiwi.storage.subformat.vhd (module), 267	kiwi.utils.block (module), 290
kiwi.storage.subformat.vhdfixed (module), 268	kiwi.utils.checksum (module), 291
kiwi.storage.subformat.vhdx (module), 268	kiwi.utils.compress (module), 291
kiwi.storage.subformat.vmdk (module), 269	kiwi.utils.sync (module), 292
kiwi.system (module), 287	kiwi.utils.sysconfig (module), 293
kiwi.system.identifier (module), 287	kiwi.version (module), 177
	kiwi.volume_manager (module), 298
	kiwi.volume_manager.base (module), 293
	kiwi.volume_manager.btrfs (module), 296
	kiwi.volume_manager.lvm (module), 297
	kiwi.xml_description (module), 177
	kiwi.xml_state (module), 178
	KiwiArchiveSetupError, 153
	KiwiArchiveTarError, 153
	KiwiBootImageDumpError, 153
	KiwiBootImageSetupError, 153
	KiwiBootLoaderConfigSetupError, 153

KiwiBootLoaderGrubDataError, [153](#)
KiwiBootLoaderGrubFontError, [154](#)
KiwiBootLoaderGrubInstallError, [154](#)
KiwiBootLoaderGrubModulesError, [154](#)
KiwiBootLoaderGrubPlatformError, [154](#)
KiwiBootLoaderGrubSecureBootError, [154](#)
KiwiBootLoaderInstallSetupError, [154](#)
KiwiBootLoaderTargetError, [154](#)
KiwiBootLoaderZiplInstallError, [154](#)
KiwiBootLoaderZiplPlatformError, [154](#)
KiwiBootLoaderZiplSetupError, [154](#)
KiwiBootStrapPhaseFailed, [154](#)
KiwiBuildahError, [155](#)
KiwiBundleError, [155](#)
KiwiCommandCapabilitiesError, [155](#)
KiwiCommandError, [155](#)
KiwiCommandNotFound, [155](#)
KiwiCommandNotLoaded, [155](#)
KiwiCompatError, [155](#)
KiwiCompressionFormatUnknown, [155](#)
KiwiConfigFileNotFound, [155](#)
KiwiContainerBuilderError, [155](#)
KiwiContainerImageSetupError, [155](#)
KiwiContainerSetupError, [156](#)
KiwiDataStructureError, [156](#)
KiwiDebootstrapError, [156](#)
KiwiDecodingError, [156](#)
KiwiDescriptionConflict, [156](#)
KiwiDescriptionInvalid, [156](#)
KiwiDeviceProviderError, [156](#)
KiwiDiskBootImageError, [156](#)
KiwiDiskFormatSetupError, [156](#)
KiwiDiskGeometryError, [156](#)
KiwiDistributionNameError, [157](#)
KiwiError, [157](#)
KiwiExtensionError, [157](#)
KiwiFileAccessError, [157](#)
KiwiFileNotFound, [157](#)
KiwiFileSystemSetupError, [157](#)
KiwiFileSystemSyncError, [157](#)
KiwiFormatSetupError, [157](#)
KiwiHelpNoCommandGiven, [157](#)
KiwiImageResizeError, [158](#)
KiwiImportDescriptionError, [158](#)
KiwiInstallBootImageError, [158](#)
KiwiInstallMediaError, [158](#)
KiwiInstallPhaseFailed, [158](#)
KiwiIsoLoaderError, [158](#)
KiwiIsoMetadataError, [158](#)
KiwiIsoToolError, [158](#)
KiwiKernelLookupError, [158](#)
KiwiLiveBootImageError, [158](#)
KiwiLoadCommandUndefined, [158](#)
KiwiLogFileSetupFailed, [159](#)
KiwiLoopSetupError, [159](#)
KiwiLuksSetupError, [159](#)
KiwiMappedDeviceError, [159](#)
KiwiMountKernelFileSystemsError, [159](#)
KiwiMountSharedDirectoryError, [159](#)
KiwiNotImplementedError, [159](#)
KiwiOCIArchiveToolError, [159](#)
KiwiPackageManagerSetupError, [159](#)
KiwiPackagesDeletePhaseFailed, [159](#)
KiwiPartitionerGptFlagError, [159](#)
KiwiPartitionerMsDosFlagError, [160](#)
KiwiPartitionerSetupError, [160](#)
KiwiPrivilegesError, [160](#)
KiwiProfileNotFound, [160](#)
KiwiPxeBootImageError, [160](#)
KiwiRaidSetupError, [160](#)
KiwiRepositorySetupError, [160](#)
KiwiRequestedTypeError, [160](#)
KiwiRequestError, [160](#)
KiwiResizeRawDiskError, [160](#)
KiwiResultError, [161](#)
KiwiRootDirExists, [161](#)
KiwiRootImportError, [161](#)
KiwiRootInitCreationError, [161](#)
KiwiRpmDirNotRemoteError, [161](#)
KiwiRuntimeConfigFormatError,

- 161
- KiwiRuntimeError, 161
- KiwiSatSolverJobError, 161
- KiwiSatSolverJobProblems, 161
- KiwiSatSolverPluginError, 162
- KiwiSchemaImportError, 162
- KiwiScriptFailed, 162
- KiwiSetupIntermediateConfigError, 162
- KiwiSizeError, 162
- KiwiSolverRepositorySetupError, 162
- KiwiSystemDeletePackagesFailed, 162
- KiwiSystemInstallPackagesFailed, 162
- KiwiSystemUpdateFailed, 162
- KiwiTargetDirectoryNotFound, 162
- KiwiTemplateError, 162
- KiwiTypeNotFound, 163
- KiwiUnknownServiceName, 163
- KiwiUriOpenError, 163
- KiwiUriStyleUnknown, 163
- KiwiUriTypeUnknown, 163
- KiwiValidationError, 163
- KiwiVhdTagError, 163
- KiwiVolumeGroupConflict, 163
- KiwiVolumeManagerSetupError, 163
- KiwiVolumeRootIDError, 163
- L**
- legacy_bios_mode() (*kiwi.firmware.FirmWare* method), 165
- list_iso() (*kiwi.iso_tools.base.IsoToolsBase* method), 229
- list_iso() (*kiwi.iso_tools.cdrtools.IsoToolsCdrTools* method), 230
- LiveImageBuilder (class in *kiwi.builder.live*), 218
- load() (*kiwi.system.result.Result* static method), 277
- load() (*kiwi.xml_description.XMLDescription* method), 178
- load_boot_xml_description() (*kiwi.boot.image.base.BootImageBase* method), 195
- load_command() (*kiwi.cli.Cli* method), 137
- load_xml_description() (*kiwi.tasks.base.CliTask* method), 287
- Logger (class in *kiwi.logger*), 166
- LoggerSchedulerFilter (class in *kiwi.logger_filter*), 169
- LoopDevice (class in *kiwi.storage.loop_device*), 256
- LuksDevice (class in *kiwi.storage.luks_device*), 257
- M**
- main() (in module *kiwi.kiwi*), 166
- map_partitions() (*kiwi.storage.disk.Disk* method), 256
- MappedDevice (class in *kiwi.storage.mapped_device*), 258
- match_package_deleted() (*kiwi.package_manager.base.PackageManagerBase* method), 233
- match_package_deleted() (*kiwi.package_manager.dnf.PackageManagerDnf* method), 235
- match_package_deleted() (*kiwi.package_manager.zypper.PackageManagerZypper* method), 237
- match_package_installed() (*kiwi.package_manager.base.PackageManagerBase* method), 233
- match_package_installed() (*kiwi.package_manager.dnf.PackageManagerDnf* method), 235
- match_package_installed() (*kiwi.package_manager.zypper.PackageManagerZypper* method), 237
- matches() (*kiwi.utils.checksum.Checksum* method), 291
- md5() (*kiwi.utils.checksum.Checksum* method), 291
- mount() (*kiwi.mount_manager.MountManager* method), 170
- mount_kernel_file_systems() (*kiwi.system.root_bind.RootBind* method), 278

<code>mount_shared_directory()</code> (<i>kiwi.system.root_bind.RootBind</i> method), 278	<code>Partitioner</code> (class in <i>kiwi.partitionner</i>), 243
<code>mount_volumes()</code> (<i>kiwi.volume_manager.base.VolumeManagerBase</i> method), 295	<code>PartitionerBase</code> (class in <i>kiwi.partitionner.base</i>), 240
<code>mount_volumes()</code> (<i>kiwi.volume_manager.btrfs.VolumeManagerBtrfs</i> method), 296	<code>PartitionerDasd</code> (class in <i>kiwi.partitionner.dasd</i>), 241
<code>mount_volumes()</code> (<i>kiwi.volume_manager.btrfs.VolumeManagerBtrfs</i> method), 296	<code>PartitionerGpt</code> (class in <i>kiwi.partitionner.gpt</i>), 242
<code>mount_volumes()</code> (<i>kiwi.volume_manager.lvm.VolumeManagerLvm</i> method), 298	<code>PartitionerMsDos</code> (class in <i>kiwi.partitionner.msdos</i>), 242
<code>MountManager</code> (class in <i>kiwi.mount_manager</i>), 170	<code>PathLvm</code> (class in <i>kiwi.path</i>), 170
<code>move_to_root()</code> (<i>kiwi.path.Path</i> static method), 171	<code>pinch_system()</code> (<i>kiwi.system.prepare.SystemPrepare</i> method), 275
N	<code>poll()</code> (<i>kiwi.command_process.CommandProcess</i> method), 140
<code>need_boot_partition()</code> (<i>kiwi.storage.setup.DiskSetup</i> method), 260	<code>poll_and_watch()</code> (<i>kiwi.command_process.CommandProcess</i> method), 140
O	<code>poll_show_progress()</code> (<i>kiwi.command_process.CommandProcess</i> method), 140
<code>ofw_mode()</code> (<i>kiwi.firmware.FirmWare</i> method), 165	<code>post_init()</code> (<i>kiwi.boot.image.base.BootImageBase</i> method), 196
<code>omit_module()</code> (<i>kiwi.boot.image.base.BootImageBase</i> method), 195	<code>post_init()</code> (<i>kiwi.boot.image.builtin_kiwi.BootImageKiwi</i> method), 198
<code>omit_module()</code> (<i>kiwi.boot.image.dracut.BootImageDracut</i> method), 197	<code>post_init()</code> (<i>kiwi.boot.image.dracut.BootImageDracut</i> method), 197
<code>opal_mode()</code> (<i>kiwi.firmware.FirmWare</i> method), 165	<code>post_init()</code> (<i>kiwi.bootloader.config.base.BootLoaderConfigBase</i> method), 201
P	<code>post_init()</code> (<i>kiwi.bootloader.config.grub2.BootLoaderConfigGrub2</i> method), 203
<code>package_matches_host_architecture()</code> (<i>kiwi.xml_state.XMLState</i> method), 190	<code>post_init()</code> (<i>kiwi.bootloader.config.isolinux.BootLoaderConfigIsolinux</i> method), 205
<code>PackageManager</code> (class in <i>kiwi.package_manager</i>), 239	<code>post_init()</code> (<i>kiwi.bootloader.config.zipl.BootLoaderConfigZipl</i> method), 206
<code>PackageManagerBase</code> (class in <i>kiwi.package_manager.base</i>), 232	<code>post_init()</code> (<i>kiwi.bootloader.install.base.BootLoaderInstallBase</i> method), 208
<code>PackageManagerDnf</code> (class in <i>kiwi.package_manager.dnf</i>), 235	<code>post_init()</code> (<i>kiwi.bootloader.install.grub2.BootLoaderInstallGrub2</i> method), 208
<code>PackageManagerTemplateAptGet</code> (class in <i>kiwi.repository.template.apt</i>), 250	
<code>PackageManagerZypper</code> (class in <i>kiwi.package_manager.zypper</i>), 237	

<i>method</i>), 209	<i>method</i>), 262
post_init() (<i>kiwi.bootloader.install.zipl.BootLoaderInstallZipl</i> <i>method</i>), 209	post_init() (<i>kiwi.storage.subformat.ova.DiskFormatOva</i> <i>method</i>), 262
post_init() (<i>kiwi.container.setup.base.ContainerSetupBase</i> <i>method</i>), 221	post_init() (<i>kiwi.storage.subformat.qcow2.DiskFormatQcow2</i> <i>method</i>), 263
post_init() (<i>kiwi.filesystem.base.FileSystemBase</i> <i>method</i>), 223	post_init() (<i>kiwi.storage.subformat.vagrant_base.DiskFormatVa</i> <i>method</i>), 265
post_init() (<i>kiwi.filesystem.clicfs.FileSystemClicFs</i> <i>method</i>), 224	post_init() (<i>kiwi.storage.subformat.vdi.DiskFormatVdi</i> <i>method</i>), 267
post_init() (<i>kiwi.package_manager.base.PackageManagerBase</i> <i>method</i>), 233	post_init() (<i>kiwi.storage.subformat.vhd.DiskFormatVhd</i> <i>method</i>), 267
post_init() (<i>kiwi.package_manager.dnf.PackageManagerDnf</i> <i>method</i>), 235	post_init() (<i>kiwi.storage.subformat.vhdfixed.DiskFormatVhdFixe</i> <i>method</i>), 268
post_init() (<i>kiwi.package_manager.zypper.PackageManagerZypper</i> <i>method</i>), 238	post_init() (<i>kiwi.storage.subformat.vhdx.DiskFormatVhdx</i> <i>method</i>), 268
post_init() (<i>kiwi.partitioner.base.PartitionerBase</i> <i>method</i>), 240	post_init() (<i>kiwi.storage.subformat.vmdk.DiskFormatVmdk</i> <i>method</i>), 269
post_init() (<i>kiwi.partitioner.dasd.PartitionerDasd</i> <i>method</i>), 241	post_init() (<i>kiwi.volume_manager.base.VolumeManagerBase</i> <i>method</i>), 295
post_init() (<i>kiwi.partitioner.gpt.PartitionerGpt</i> <i>method</i>), 242	post_init() (<i>kiwi.volume_manager.btrfs.VolumeManagerBtrfs</i> <i>method</i>), 296
post_init() (<i>kiwi.partitioner.msdos.PartitionerMsDos</i> <i>method</i>), 243	post_init() (<i>kiwi.volume_manager.lvm.VolumeManagerLVM</i> <i>method</i>), 298
post_init() (<i>kiwi.repository.base.RepositoryBase</i> <i>method</i>), 245	post_process_install_requests_bootstrap() (<i>kiwi.package_manager.base.PackageManagerBase</i> <i>method</i>), 233
post_init() (<i>kiwi.repository.dnf.RepositoryDnf</i> <i>method</i>), 246	post_process_install_requests_bootstrap() (<i>kiwi.package_manager.dnf.PackageManagerDnf</i> <i>method</i>), 235
post_init() (<i>kiwi.repository.zypper.RepositoryZypper</i> <i>method</i>), 249	post_process_install_requests_bootstrap() (<i>kiwi.package_manager.zypper.PackageManagerZypp</i> <i>method</i>), 238
post_init() (<i>kiwi.storage.subformat.base.DiskFormatBase</i> <i>method</i>), 261	prepare() (<i>kiwi.boot.image.base.BootImageBase</i> <i>method</i>), 196
post_init() (<i>kiwi.storage.subformat.gce.DiskFormatGce</i> <i>method</i>), 261	prepare() (<i>kiwi.boot.image.builtin_kiwi.BootImageKiwi</i> <i>method</i>), 198
	prepare() (<i>kiwi.boot.image.dracut.BootImageDracut</i> <i>method</i>), 198

method), 197
 print_results() (kiwi.system.result.Result *method*), 277
 Privileges (class in kiwi.privileges), 172
 process() (kiwi.tasks.result_bundle.ResultBundleTask *method*), 288
 process() (kiwi.tasks.result_list.ResultListTask *method*), 288
 process() (kiwi.tasks.system_build.SystemBuildTask *method*), 288
 process() (kiwi.tasks.system_create.SystemCreateTask *method*), 289
 process() (kiwi.tasks.system_prepare.SystemPrepareTask *method*), 289
 process() (kiwi.tasks.system_update.SystemUpdateTask *method*), 289
 process_delete_requests() (kiwi.package_manager.base.PackageManagerBase *method*), 233
 process_delete_requests() (kiwi.package_manager.dnf.PackageManagerDnf *method*), 236
 process_delete_requests() (kiwi.package_manager.zypper.PackageManagerZypper *method*), 238
 process_install_requests() (kiwi.package_manager.base.PackageManagerBase *method*), 234
 process_install_requests() (kiwi.package_manager.dnf.PackageManagerDnf *method*), 236
 process_install_requests() (kiwi.package_manager.zypper.PackageManagerZypper *method*), 238
 process_install_requests_bootstrap() (kiwi.package_manager.base.PackageManagerBase *method*), 234
 process_install_requests_bootstrap() (kiwi.package_manager.dnf.PackageManagerDnf *method*), 236
 process_install_requests_bootstrap() (kiwi.package_manager.zypper.PackageManagerZypper *method*), 238
 process_only_required() (kiwi.package_manager.base.PackageManagerBase *method*), 234
 process_only_required() (kiwi.package_manager.dnf.PackageManagerDnf *method*), 236
 process_only_required() (kiwi.package_manager.zypper.PackageManagerZypper *method*), 238
 process_plus_recommended() (kiwi.package_manager.dnf.PackageManagerDnf *method*), 236
 process_plus_recommended() (kiwi.package_manager.zypper.PackageManagerZypper *method*), 238
 Profile (class in kiwi.system.profile), 276
 update_matches_host_architecture() (kiwi.xml_state.XMLState *method*), 191
 project_file() (kiwi.defaults.Defaults *static method*), 153
 PxeBuilder (class in kiwi.builder.pxe), 218

Q

quadruple_token() (kiwi.tasks.base.CliTask *method*), 287
 quote() (kiwi.system.shell.Shell *static method*), 283
 set_key_value_file() (kiwi.system.shell.Shell *static method*), 283
 NoteTitle() (kiwi.bootloader.config.base.BootLoaderConfigBase *method*), 201
 RaidDevice (class in kiwi.storage.raid_device), 258
 rebase_to_root() (kiwi.path.Path *static method*), 171
 relocate_boot_catalog() (kiwi.iso_tools.iso.Iso *static method*), 232
 remove() (kiwi.path.Path *static method*), 171
 remove_hierarchy() (kiwi.path.Path *static method*), 171

Repository (class in *kiwi.repository*), 249 method), 261
RepositoryBase (class in *kiwi.repository.base*), 243 (*kiwi.partitionner.base.PartitionerBase*
method), 240
RepositoryDnf (class in *kiwi.repository.dnf*), 245 *resize_table()*
RepositoryZypper (class in *kiwi.repository.zypper*), 247 (*kiwi.partitionner.dasd.PartitionerDasd*
method), 241
request_collection() *resize_table()*
(*kiwi.package_manager.base.PackageManagerBase*), 234 (*kiwi.partitionner.gpt.PartitionerGpt*
method), 242
request_collection() *resize_table()*
(*kiwi.package_manager.dnf.PackageManagerDnf*), 236 (*kiwi.partitionner.msdos.PartitionerMsDos*
method), 243
request_collection() Result (class in *kiwi.system.result*), 276
(*kiwi.package_manager.zypper.PackageManagerZypper*), 238 *kiwi.system.result_type* (class in
method), 238 *kiwi.system.result*), 277
request_package() ResultBundleTask (class in
(*kiwi.package_manager.base.PackageManagerBase*), 234 (*kiwi.tasks.result_bundle*), 288
method), 234 ResultListTask (class in
method), 236 *kiwi.tasks.result_list*), 288
request_package() *run_dnf_code()*
(*kiwi.package_manager.dnf.PackageManagerDnf*), 236 (*kiwi.command_process.CommandProcess*
method), 140
request_package() *method*), 140
(*kiwi.package_manager.zypper.PackageManagerZypper*), 239 (*kiwi.system.root_bind*),
method), 239 278
request_package_exclusion() RootInit (class in *kiwi.system.root_init*),
(*kiwi.package_manager.base.PackageManagerBase*), 234 239
method), 234 *run()* (*kiwi.command.Command* static
method), 138
request_package_exclusion() *method*), 138
(*kiwi.package_manager.dnf.PackageManagerDnf*), 236 (*kiwi.tasks.base.CliTask*
method), 236 *method*), 287
request_package_exclusion() *run_common_function()*
(*kiwi.package_manager.zypper.PackageManagerZypper*), 239 (*kiwi.system.shell.Shell* static
method), 239 *method*), 283
request_package_lock() *runtime_config()*
(*kiwi.package_manager.base.PackageManagerBase*), 234 (*kiwi.repository.base.RepositoryBase*
method), 234 *method*), 245
request_product() *runtime_config()*
(*kiwi.package_manager.base.PackageManagerBase*), 234 (*kiwi.repository.dnf.RepositoryDnf*
method), 234 *method*), 247
request_product() *runtime_config()*
(*kiwi.package_manager.dnf.PackageManagerDnf*), 236 (*kiwi.repository.zypper.RepositoryZypper*
method), 236 *method*), 249
request_product() RuntimeChecker (class in
(*kiwi.package_manager.zypper.PackageManagerZypper*), 239 (*kiwi.runtime_checker*), 173
method), 239 RuntimeConfig (class in
method), 239 *kiwi.runtime_config*), 175
resize_raw_disk() *method*), 175
(*kiwi.storage.subformat.base.DiskFormatBase*

S

Sat (class in *kiwi.solver.sat*), 252

set_color_format() (*kiwi.logger.Logger* method), 167

set_container_config_tag() (*kiwi.xml_state.XMLState* method), 191

set_derived_from_image_uri() (*kiwi.xml_state.XMLState* method), 191

set_flag() (*kiwi.partitionner.base.PartitionerBase* method), 240

set_flag() (*kiwi.partitionner.gpt.PartitionerGpt* method), 242

set_flag() (*kiwi.partitionner.msdos.PartitionerMsDos* method), 243

set_hybrid_mbr() (*kiwi.partitionner.base.PartitionerBase* method), 241

set_hybrid_mbr() (*kiwi.partitionner.gpt.PartitionerGpt* method), 242

set_logfile() (*kiwi.logger.Logger* method), 167

set_mbr() (*kiwi.partitionner.base.PartitionerBase* method), 241

set_mbr() (*kiwi.partitionner.gpt.PartitionerGpt* method), 242

set_media_tag() (*kiwi.iso_tools.iso.Iso* static method), 232

set_property_readonly_root() (*kiwi.volume_manager.base.VolumeManagerBase* method), 295

set_property_readonly_root() (*kiwi.volume_manager.btrfs.VolumeManagerBtrfs* method), 296

set_repository() (*kiwi.xml_state.XMLState* method), 191

set_root_partition_uuid() (*kiwi.xml_state.XMLState* method), 191

set_selinux_file_contexts() (*kiwi.system.setup.SystemSetup* method), 281

setLogLevel() (*kiwi.logger.Logger* method), 167

setup() (*kiwi.container.setup.base.ContainerSetupBase* method), 221

setup() (*kiwi.volume_manager.base.VolumeManagerBase* method), 295

setup() (*kiwi.volume_manager.btrfs.VolumeManagerBtrfs* method), 297

setup() (*kiwi.volume_manager.lvm.VolumeManagerLVM* method), 298

setup_disk_boot_images() (*kiwi.bootloader.config.base.BootLoaderConfigBase* method), 201

setup_disk_boot_images() (*kiwi.bootloader.config.grub2.BootLoaderConfigGrub2* method), 203

setup_disk_boot_images() (*kiwi.bootloader.config.zipl.BootLoaderConfigZipl* method), 207

setup_disk_image_config() (*kiwi.bootloader.config.base.BootLoaderConfigBase* method), 201

setup_disk_image_config() (*kiwi.bootloader.config.grub2.BootLoaderConfigGrub2* method), 203

setup_disk_image_config() (*kiwi.bootloader.config.zipl.BootLoaderConfigZipl* method), 207

setup_groups() (*kiwi.system.setup.SystemSetup* method), 282

setup_home_for_user() (*kiwi.system.users.Users* method), 286

setup_install_boot_images() (*kiwi.bootloader.config.base.BootLoaderConfigBase* method), 202

setup_install_boot_images() (*kiwi.bootloader.config.grub2.BootLoaderConfigGrub2* method), 204

setup_install_boot_images() (*kiwi.bootloader.config.isolinux.BootLoaderConfigIsolinux* method), 205

setup_install_image_config() (*kiwi.bootloader.config.base.BootLoaderConfigBase* method), 202

setup_install_image_config() (*kiwi.bootloader.config.grub2.BootLoaderConfigGrub2* method), 203

method), 204
setup_install_image_config()
 (*kiwi.bootloader.config.isolinux.BootLoaderConfigIsolinux*
 method), 206
setup_intermediate_config()
 (*kiwi.system.root_bind.RootBind*
 method), 278
setup_isolinux_boot_path()
 (*kiwi.iso_tools.iso.Iso* *method*), 232
setup_keyboard_map()
 (*kiwi.system.setup.SystemSetup*
 method), 282
setup_live_boot_images()
 (*kiwi.bootloader.config.base.BootLoaderConfigBase*
 method), 202
setup_live_boot_images()
 (*kiwi.bootloader.config.grub2.BootLoaderConfigGrub2*
 method), 204
setup_live_boot_images()
 (*kiwi.bootloader.config.isolinux.BootLoaderConfigIsolinux*
 method), 206
setup_live_image_config()
 (*kiwi.bootloader.config.base.BootLoaderConfigBase*
 method), 202
setup_live_image_config()
 (*kiwi.bootloader.config.grub2.BootLoaderConfigGrub2*
 method), 204
setup_live_image_config()
 (*kiwi.bootloader.config.isolinux.BootLoaderConfigIsolinux*
 method), 206
setup_locale()
 (*kiwi.system.setup.SystemSetup*
 method), 282
setup_machine_id()
 (*kiwi.system.setup.SystemSetup*
 method), 282
setup_media_loader_directory()
 (*kiwi.iso_tools.base.IsoToolsBase*
 static method), 229
setup_mountpoint()
 (*kiwi.volume_manager.base.VolumeManagerBase*
 method), 295
setup_package_database_configuration()
 (*kiwi.repository.base.RepositoryBase*
 method), 245
setup_package_database_configuration()
 (*kiwi.repository.dnf.RepositoryDnf*
 method), 247
setup_package_database_configuration()
 (*kiwi.repository.zypper.RepositoryZypper*
 method), 249
setup_permissions()
 (*kiwi.system.setup.SystemSetup*
 method), 282
setup_plymouth_splash()
 (*kiwi.system.setup.SystemSetup*
 method), 282
setup_repositories()
 (*kiwi.system.prepare.SystemPrepare*
 method), 275
setup_root_console()
 (*kiwi.container.setup.base.ContainerSetupBase*
 method), 222
setup_static_device_nodes()
 (*kiwi.container.setup.base.ContainerSetupBase*
 method), 222
setup_sysconfig_bootloader()
 (*kiwi.bootloader.config.base.BootLoaderConfigBase*
 method), 202
setup_timezone()
 (*kiwi.system.setup.SystemSetup*
 method), 282
setup_users()
 (*kiwi.system.setup.SystemSetup*
 method), 282
sextuple_token()
 (*kiwi.iso_tools.base.CliTask* *method*),
 287
sha256() (*kiwi.utils.checksum.Checksum*
 method), 291
shasum() (*kiwi.system.result.result_file_type*
 property), 277
Shell (*class in kiwi.system.shell*), 283
show() (*kiwi.help.Help* *method*), 165
show_and_exit_on_help_request()
 (*kiwi.cli.Cli* *method*), 137
solve() (*kiwi.solver.sat.Sat* *method*), 252
SolverRepository (*class in*
 kiwi.solver.repository), 252
SolverRepositoryBase (*class in*
 kiwi.solver.repository.base), 250
SolverRepositoryRpmDir (*class in*
 kiwi.solver.repository.rpm_dir), 251
SolverRepositoryRpmMd (*class in*
 kiwi.solver.repository.rpm_md), 251
SolverRepositorySUSE (*class in*

kiwi.solver.repository.suse), 251
 sort_by_hierarchy() (class in *kiwi.path.Path*), 172
 store_to_result() (class in *kiwi.storage.subformat.base.DiskFormatBase*), 261
 store_to_result() (class in *kiwi.storage.subformat.gce.DiskFormatGce*), 262
 store_to_result() (class in *kiwi.storage.subformat.ova.DiskFormatOva*), 263
 store_to_result() (class in *kiwi.storage.subformat.qcow2.DiskFormatQcow2*), 263
 store_to_result() (class in *kiwi.storage.subformat.vagrant_base.DiskFormatVagrantBase*), 265
 store_to_result() (class in *kiwi.storage.subformat.vhdfixed.DiskFormatVhdFixed*), 268
 store_to_result() (class in *kiwi.storage.subformat.vmdk.DiskFormatVmdk*), 269
 sync_data() (class in *kiwi.filesystem.base.FileSystemBase*), 223
 sync_data() (class in *kiwi.utils.sync.DataSync*), 292
 sync_data() (class in *kiwi.volume_manager.base.VolumeManagerBase*), 295
 sync_data() (class in *kiwi.volume_manager.btrfs.VolumeManagerBtrfs*), 297
 SysConfig (class in *kiwi.utils.sysconfig*), 293
 SystemBuildTask (class in *kiwi.tasks.system_build*), 288
 SystemCreateTask (class in *kiwi.tasks.system_create*), 289
 SystemIdentifier (class in *kiwi.system.identifier*), 272
 SystemPrepare (class in *kiwi.system.prepare*), 274
 SystemPrepareTask (class in *kiwi.tasks.system_prepare*), 289
 SystemSetup (class in *kiwi.system.setup*), 279
 SystemSize (class in *kiwi.system.size*), 283
 SystemUpdateTask (class in *kiwi.tasks.system_update*), 289
 target_supports_extended_attributes() (class in *kiwi.utils.sync.DataSync*), 292
 timestamp() (class in *kiwi.solver.repository.base.SolverRepositoryBase*), 251
 timestamp() (class in *kiwi.solver.repository.rpm_md.SolverRepositoryRpmMd*), 251
 to_profile() (class in *kiwi.defaults.defaults*), 252
 translate() (class in *kiwi.system.uri.Uri*), 285
 umount() (class in *kiwi.mount_manager.MountManager*), 170
 umount_lazy() (class in *kiwi.mount_manager.MountManager*), 170
 umount_volumes() (class in *kiwi.volume_manager.base.VolumeManagerBase*), 295
 umount_volumes() (class in *kiwi.volume_manager.btrfs.VolumeManagerBtrfs*), 297
 umount_volumes() (class in *kiwi.volume_manager.lvm.VolumeManagerLVM*), 298
 uncompress() (class in *kiwi.utils.compress.Compress*), 292
 update() (class in *kiwi.package_manager.base.PackageManagerBase*), 234
 update() (class in *kiwi.package_manager.dnf.PackageManagerDnf*), 236
 update() (class in *kiwi.package_manager.zypper.PackageManagerZypper*), 239
 update_system() (class in *kiwi.system.prepare.SystemPrepare*), 275
 Uri (class in *kiwi.system.uri*), 284

usage() (in module *kiwi.kiwi*), 166
 use_default_location() (in *kiwi.repository.base.RepositoryBase* method), 245
 use_default_location() (in *kiwi.repository.dnf.RepositoryDnf* method), 247
 use_default_location() (in *kiwi.repository.zypper.RepositoryZypper* method), 249
 use_for_bundle() (in *kiwi.system.result.result_file_type* property), 277
 user_add() (in *kiwi.system.users.Users* method), 286
 user_exists() (in *kiwi.system.users.Users* method), 286
 user_modify() (in *kiwi.system.users.Users* method), 286
 Users (class in *kiwi.system.users*), 285

V

vagrant_post_init() (in *kiwi.storage.subformat.vagrant_base.DiskFormatVagrantBase* method), 265
 vagrant_post_init() (in *kiwi.storage.subformat.vagrant_libvirt.DiskFormatVagrantLibvirt* method), 266
 vagrant_post_init() (in *kiwi.storage.subformat.vagrant_virtualbox.DiskFormatVagrantVirtualbox* method), 266
 VagrantConfigTemplate (class in *kiwi.storage.subformat.template.vagrant_config*), 270
 verify_image_size() (in *kiwi.system.result.Result* method), 277
 VirtualboxOvfTemplate (class in *kiwi.storage.subformat.template.virtualbox_ovf*), 272
 VmwareSettingsTemplate (class in *kiwi.storage.subformat.template.vmware_settings*), 270
 VolumeManager (class in *kiwi.volume_manager*), 298
 VolumeManagerBase (class in *kiwi.volume_manager.base*), 293
 VolumeManagerBtrfs (class in *kiwi.volume_manager.btrfs*), 296
 VolumeManagerLVM (class in *kiwi.volume_manager.lvm*), 297

W

WarningFilter (class in *kiwi.logger_filter*), 169
 which() (in *kiwi.path.Path* static method), 172
 wipe() (in *kiwi.path.Path* static method), 172
 wipe() (in *kiwi.storage.disk.Disk* method), 256
 write() (in *kiwi.bootloader.config.base.BootLoaderConfigBase* method), 203
 write() (in *kiwi.bootloader.config.grub2.BootLoaderConfigGrub2* method), 205
 write() (in *kiwi.bootloader.config.isolinux.BootLoaderConfigIsolinux* method), 206
 write() (in *kiwi.bootloader.config.zipl.BootLoaderConfigZipl* method), 207
 write() (in *kiwi.system.identifier.SystemIdentifier* method), 272
 write() (in *kiwi.utils.sysconfig.SysConfig* method), 293
 write_meta_data() (in *kiwi.bootloader.config.base.BootLoaderConfigBase* method), 203
 write_meta_data() (in *kiwi.bootloader.config.grub2.BootLoaderConfigGrub2* method), 205
 write_system_config_file() (in *kiwi.boot.image.dracut.BootImageDracut* method), 197
 write_to_disk() (in *kiwi.system.identifier.SystemIdentifier* method), 272

X

XMLDescription (class in *kiwi.xml_description*), 177
 XMLState (class in *kiwi.xml_state*), 178
 xz() (in *kiwi.utils.compress.Compress* method), 292