

---

# **KIWI Documentation**

***Release 9.16.19***

**Marcus Schäfer**

**Dec 18, 2018**

---

## Contents

---

<b>1</b>	<b>Overview</b>	<b>2</b>
<b>2</b>	<b>Installation</b>	<b>21</b>
<b>3</b>	<b>Quick Start</b>	<b>23</b>
<b>4</b>	<b>Building Images</b>	<b>25</b>
<b>5</b>	<b>KIWI Commands</b>	<b>62</b>
<b>6</b>	<b>Development and Contributing</b>	<b>75</b>
<b>7</b>	<b>Appliance ?</b>	<b>232</b>
<b>8</b>	<b>Contact</b>	<b>233</b>
	<b>Python Module Index</b>	<b>234</b>
	<b>Index</b>	<b>236</b>

Welcome to the documentation for KIWI 9.16.19- the command line utility to build linux system appliances.

### Links

- [GitHub Sources](#)
- [GitHub Releases](#)
- [RPM Packages](#)
- [Build Tests\(x86\)](#)
- [Build Tests\(arm\)](#)
- [Build Tests\(s390\)](#)

# CHAPTER 1

---

## Overview

---

---

### Hint: Abstract

This document provides a conceptual overview about the steps of creating an image with KIWI. It also explains the terminology regarding the concept and process when building system images with KIWI 9.16.19.

---

## 1.1 Legacy KIWI vs. Next Generation

---

### Hint: 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** 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 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 with a maintained and stable version in the background.

Users will be able to use both versions in parallel. In addition, the next generation KIWI 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 the legacy and the next generation KIWI, if the distribution and all configured features are supported by the used KIWI version.

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

The legacy KIWI 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: [Legacy KIWI packages](#)

### 1.1.1 When Do I need to use the legacy kiwi

- 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*.

### 1.1.2 Dropped Features

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

**Split systems** The legacy KIWI 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 legacy KIWI version will not be continued.

**Partition based install method in OEM install image** The section *Deployment Methods* describes the supported OEM installation procedures. The legacy KIWI 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.

### 1.1.3 Compatibility

The legacy KIWI version can be installed and used together with the next generation KIWI.

---

**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 KIWI versions are fully compatible. Users can build their appliances with both versions and the same appliance description. If the appliance description uses features the next generation KIWI 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 legacy KIWI, unless the appliance description properly encapsulates the differences into a profile.

The next generation KIWI also provides the `--compat` option and the **kiwicompat** tool to be able to use the same commandline as provided with the legacy KIWI version.

## 1.2 Basic Workflow

---

### Hint: 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 9.16.19.

---

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 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 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.

---

## 1.2.1 Building Images

KIWI 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. The `config.xml` file is part of the *configuration directory (tree)* that describes the image to be created by KIWI.

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

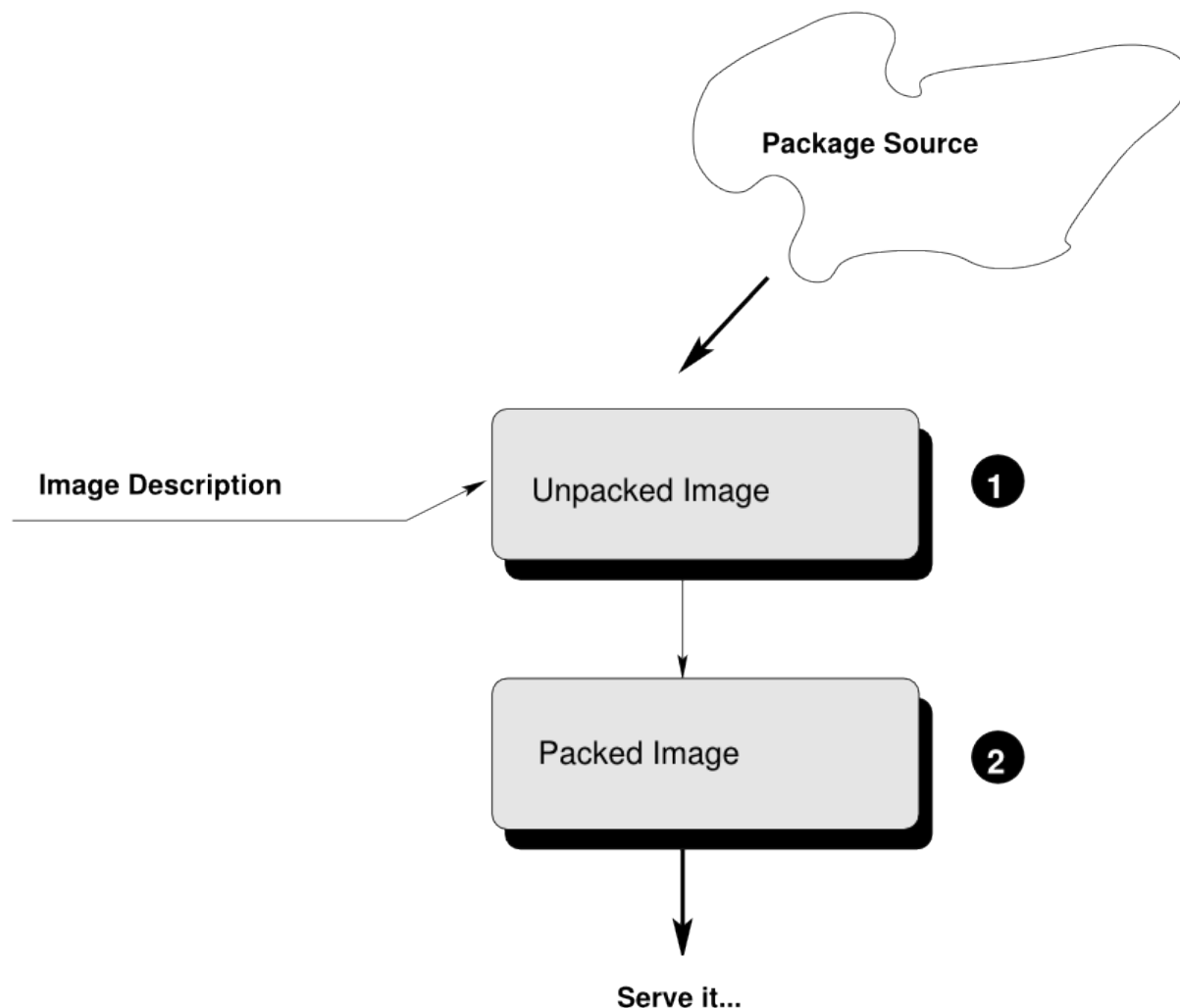


Fig. 1: Image Creation Architecture

- (1) Unpacked Image Encapsulated system reachable via chroot
- (2) Packed Image Encapsulated system reachable via kernel file system/extension drivers such as loopback mounts, etc.

---

**Note:** KIWI configuration file name convention

KIWI at first place looks for a configuration file named `config.xml`. If there is no such file,



KIWI looks for files with a `*.kiwi` extension. In that case, the first match is the loaded file.

---

## The Prepare Step

The creation of an image with KIWI is a two step process. The first step is called the `prepare` step and it must complete successfully before the second step, the `create` step can be executed.

During the prepare step, KIWI creates an *unpacked image*, also called “root tree”. The new root tree is created in a directory specified on the command line with the option `--root` argument or the value of the `defaultroot` element in the `config.xml` file. This directory will be the installation target for software packages to be installed during the image creation process.

For package installation, KIWI relies on the package manager specified with the `packagemanager` element in the `config.xml` file. KIWI supports the following package managers: `dnf`, `zypper` (default), `yum` and `apt/dpkg`.

The prepare step consists of the following substeps:

1. **Create Target Root Directory.**

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

2. **Install Packages.**

Initially, KIWI configures the package manager to use the repositories specified in the configuration file and/or the command line. Following the repository setup, the packages specified in the `bootstrap` section of the configuration file are installed in a temporary workspace external to the target root tree. This establishes the initial environment to support the completion of the process in chroot setting. The essential packages to specify as part of the bootstrap environment are the `filesystem` and `glibc-locale` packages. The dependency chain of these two packages is sufficient to populate the bootstrap environment with all required software to support the installation of packages into the new root tree.

The installation of software packages through the selected package manager may install unwanted packages. Removing such packages can be accomplished by marking them for deletion in the configuration file. To do so specify a configuration entry like:

```
<package type="delete">package_to_be_deleted</package>
```

3. **Apply the Overlay Tree.**

After the package installation is complete, KIWI will apply all files and directories present in the overlay directory named `root` to the target root tree. Files already present in the target root directory will be overwritten, others will be added. This allows you to overwrite any file that was installed by one of the packages during the installation phase.

4. **Apply Archives.**

Any archive specified with the `archive` element in the `config.xml` file is applied in the specified order (top to bottom) after the overlay tree copy operation is complete. Files and directories will be 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 Scripts `config.sh`.

At the end of the preparation stage the script named `config.sh` is executed if present. It is executed on the top level of the target root tree. The script's primary function is to complete the system configuration, for example, by activating services. See [Image Customization with `config.sh` Shell Script](#) section for further details.

### 6. Manage The New Root Tree.

The unpacked image directory is a directory, as far as the build system is concerned you can manipulate the content of this directory according to your needs. Since it represents a system installation you can “chroot” into this directory for testing purposes. The file system contains an additional directory named `/image` that is not present in a regular system. It contains information KIWI requires during the create step, including a copy of the `config.xml` file.

Do not make any changes to the system, since they will get lost when re-running the `prepare` step again. Additionally, you may introduce errors that will 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.

## The Create Step

The successful completion of the `prepare` step is a prerequisite for the `create` step. It ensures the unpacked root tree is complete and consistent. Creating the packed, or final, image is done in the `create` step. Multiple images can be created using the same unpacked root tree. It is, for example, possible to create a self installing OEM image and a virtual machine image from a single unpacked root tree. The only prerequisite is that both image types are specified in the `config.xml` before the `prepare` step is executed.

During the `create` step the following major operations are performed by KIWI:

#### 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 executed 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.

#### 2. Create Requested Image Type.

The image types that can be created from a prepared image tree depend on the types specified in the image description `config.xml` file. The configuration file must contain at least one `type` element. see: [Building Images](#)

## 1.2.2 Components of an Image Description

A KIWI image description can be composed by several parts. The main part is the KIWI 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.

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.

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>
```

### Image Customization with `config.sh` Shell Script

The KIWI image description allows to have an optional `config.sh` bash script in place. It can be used for changes appropriate for all images to be created from a given unpacked image (since `config.sh` runs prior to create step). Basically the script should be designed to take over control of adding the image operating system configuration. Configuration in that sense means all tasks which runs once in an OS installation process like activating services, creating configuration files, prepare an environment for a firstboot workflow, etc. The `config.sh` script is called 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 an exit code `!= 0` the kiwi process will exit with an error too.

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

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

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

#=====
# Mount system filesystems
#-----
baseMount

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

#=====
# Umount kernel filesystems
#-----
baseCleanMount

#=====
# Exit safely
#-----
exit 0
```

## Common Functions

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

**baseCleanMount** Umount the system filesystems `/proc`, `/dev/pts`, and `/sys`.

**baseDisableCtrlAltDel** Disable the Ctrl–Alt–Del key sequence setting in `/etc/inittab`.

**baseGetPackagesForDeletion** Return the name(s) of packages which will be deleted.

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

**baseSetRunlevel {value}** Set the default run level.

**baseSetupBoot** Set up the `linuxrc` as `init`.

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

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

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

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

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

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

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

**baseStripDocs {list of docu names to keep}** Remove all documentation, except one given as parameter.

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

**baseStripLocales {list of locales}** Remove all locales, except one given as parameter.

**baseStripMans {list of manpages to keep}** Remove all manual pages, except one given as parameter example:

```
baseStripMans more less
```

**baseStripRPM** Remove rpms defined in `config.xml` in the `image type=delete` section.

**suseRemovePackagesMarkedForDeletion** Remove rpms defined in `config.xml` in the `image type=delete` section. The difference compared to `baseStripRPM` is that the suse variant checks if the package is really installed prior to passing it to rpm to uninstall it. The suse rpm exits with an error exit code while there are other rpm version which just ignore if an uninstall request was set on a package which is not installed.

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

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

**baseUpdateSysConfig {filename} {variable} {value}** Update `sysconfig` variable contents.

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

**Echo {echo commandline}** Helper function to print a message to the controlling terminal.

**Rm {list of files}** Helper function to delete files and announce it to log.

**Rpm {rpm commandline}** Helper function to the RPM function and announce it to log.

**suseConfig** Setup keytable language, timezone and hwclock if specified in `config.xml` and call `SuSEconfig` afterwards `SuSEconfig` is only called on systems which still support it.

**suseInsertService {servicename}** This function calls `baseInsertService` and exists only for compatibility reasons.

**suseRemoveService {servicename}** This function calls `baseRemoveService` and exists only for compatibility reasons.

**baseInsertService {servicename}** Activate the given service by using the `chkconfig` or `systemctl` program. Which init system is in use is auto detected.

**baseRemoveService {servicename}** Deactivate the given service by using the `chkconfig` or `systemctl` program. Which init system is in use is auto detected.

**baseService {servicename} {on|off}** Activate/Deactivate a service by using the `chkconfig` or `systemctl` program. The function requires the service name and the value on or off as parameters. Which init system is in use is auto detected.

**suseActivateDefaultServices** Activates the following `sysVInit` services to be on by default using the `chkconfig` program: `boot.rootfsck`, `boot.cleanup`, `boot.localfs`, `boot.localnet`, `boot.clock`, `policykitd`, `dbus`, `consolekit`, `haldaemon`, `network`, `atd`, `syslog`, `cron`, `kbd`. And the following for `systemd` systems: `network`, `cron`.

**suseSetupProduct** This function creates the `baseproduct` link in `/etc/products.d` pointing to the installed product.

**suseSetupProductInformation** This function will use `zypper` to search for the installed product and install all product specific packages. This function only makes sense if `zypper` is used as package manager.

**suseStripPackager {-a}** Remove smart or `zypper` packages and db files Also remove rpm package and db if -a given.

## Profile Environment Variables

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

**\$kiwi\_compressed** The value of the compressed attribute set in the type element in `config.xml`.

**\$kiwi\_delete** A list of all packages which are part of the packages section with `type="delete"` in `config.xml`.

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

**\$kiwi\_iname** The name of the image as listed in `config.xml`.

**\$kiwi\_iversio**n The image version string major.minor.release.

**\$kiwi\_keytable** The contents of the keytable setup as done in `config.xml`.

**\$kiwi\_language** The contents of the locale setup as done in `config.xml`.

**\$kiwi\_profiles** A list of profiles used to build this image.

**\$kiwi\_size** The predefined size value for this image. This is not the computed size but only the optional size value of the preferences section in `config.xml`.

**\$kiwi\_timezone** The contents of the timezone setup as done in `config.xml`.

**\$kiwi\_type** The basic image type.

## Configuration Tips

In this section some ideas of how `config.sh` file could be used to fine tune the resulting unpacked image are quickly described:

### 1. Stateless systemd UUIDs:

Usually during the image packages installation when *dbus* and/or *systemd* are installed machine ID files are created and set (`/etc/machine-id`, `/var/lib/dbus/machine-id`). Those UUIDs are meant to be unique and set only once in each deployment. In order to ensure that every single box running out from the same image has its own specific systemd UUID, the original image must not include any systemd or dbus ID, this way it is assigned during the first boot. The following bash snippet allows this behavior in `config.sh`:

```
#=====
# Make machine-id stateless
#-----
if [ -e /etc/machine-id ]; then
    > /etc/machine-id
    if [ -e /var/lib/dbus/machine-id ]; then
        rm /var/lib/dbus/machine-id
    fi
    ln -s /etc/machine-id /var/lib/dbus/machine-id
fi
```

---

**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.

---



## Image Customization with `images.sh` Shell Script

The KIWI image description allows to have an optional `images.sh` bash script in place. It can be used for changes appropriate for certain images/image types on case-by-case basis (since it runs at beginning of *create step*). Basically the script should be designed to take over control of handling image type specific tasks. For example if building the `oem` type requires some additional package or config it can be handled in `images.sh`. Please keep in mind there is only one unpacked root tree the script operates in. This means all changes are permanent and will not be automatically restored. It is also the script authors tasks to check if changes done before do not interfere in a negative way if another image type is created from the same unpacked image root tree. If `images.sh` exits with an exit code `!= 0` the kiwi process will exit with an error too.

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

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

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

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

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

## Common Functions

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

**baseCleanMount** Umount the system file systems `/proc`, `/dev/pts`, and `/sys`.

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

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



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

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

**suseStripInitrd** This function removes a whole bunch of tools binaries and libraries which are not required to boot a suse system with KIWI.

**Rm {list of files}** Helper function to delete files and announce it to log.

**Rpm {rpm commandline}** Helper function to the rpm function and announce it to log.

**Echo {echo commandline}** Helper function to print a message to the controlling terminal.

**Debug {message}** Helper function to print a message if the variable *DEBUG* is set to 1.

## Profile environment variables

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

**\$kiwi\_iname** The name of the image as listed in `config.xml`.

**\$kiwi\_iversio**n The image version string major.minor.release.

**\$kiwi\_keytable** The contents of the keytable setup as done in `config.xml`.

**\$kiwi\_language** The contents of the locale setup as done in `config.xml`.

**\$kiwi\_timezone** The contents of the timezone setup as done in `config.xml`.

**\$kiwi\_delete** A list of all packages which are part of the packages section with *type="delete"* in `config.xml`.

**\$kiwi\_profiles** A list of profiles used to build this image.

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

**\$kiwi\_size** The predefined size value for this image. This is not the computed size but only the optional size value of the preferences section in `config.xml`.

**\$kiwi\_compressed** The value of the compressed attribute set in the type element in `config.xml`.

**\$kiwi\_type** The basic image type.

### 1.2.3 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 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 create functionality which is not natively provided by `dracut` itself. In the scope of KIWI the following `dracut` modules are used:

**kiwi-dump** The `dracut` module which serves as an image installer. It provides the required implementation to install a KIWI 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-live** The `dracut` module which boots up a KIWI live image. This module is required if the `iso` image type is selected

**kiwi-overlay** The `dracut` module which 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 `overlayfs` for the root filesystem using an extra partition on the same disk for persistent data.

**kiwi-repart** The `dracut` module which 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** The `dracut` module which provides functions of general use and serves as a library usable by other `dracut` modules. As the name says its main purpose is to function as library for the above mentioned kiwi `dracut` modules.

---

#### Note: Custom Boot Image Support

Apart from the standard `dracut` based creation of the boot image, KIWI 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-leap42.3"/>
```

Such boot descriptions for the `oem` and `pxe` types are currently still provided by the KIWI 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.

---

## Boot Image Hook-Scripts

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

```
$ vi root/usr/lib/dracut/modules.d/90my-module/module-setup.sh

#!/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_
```

(continues on next page)

(continued from previous page)

```
}

# 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 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 at

<http://people.redhat.com/harald/dracut.html>

## Boot Image Parameters

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

**rd.kiwi.debug** This variable activates the debug log file for the kiwi part of the boot process at `/run/initramfs/log/boot.kiwi`

**rd.kiwi.install.pxe** This variable 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** This variable specifies the remote location of the system image in a pxe based oem installation

**rd.live.overlay.persistent** This variable tells a live iso image to prepare a persistent write partition.

**rd.live.overlay.cowfs** This variable tells a live iso image which filesystem should be used to store data on the persistent write partition.

**rd.live.dir** This variable tells a live iso image the directory which contains the live OS root directory. Defaults to `LiveOS`

**rd.live.squashimg** This variable 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
```

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

## 1.4 Terminology

**Appliance** An appliance is a ready to use image of an operating system including a preconfigured 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 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 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 below a directory named `root`. The content of the directory structure is copied over the existing file system (overlayed) of the appliance root. This includes permissions and attributes as a supplement.

**KIWI** 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.

## 1.5 System Requirements

To use and run KIWI, you need:

- A recent Linux distribution, see *Supported Distributions* for details. Alternatively a Linux distribution which supports the docker container system as it would be required when running KIWI in a container, see: *Building in a Self-Contained Environment*
- Enough free disk space to build and store the image. We recommend a minimum of 10GB.
- Python version 2.7, 3.4 or higher; as KIWI supports both Python versions, the information in this guide applies also to both packages, be it `python3-kiwi` or `python2-kiwi`.
- Git (package `git-core`) to clone a repository.
- Virtualization technology to start the image. We recommend QEMU

## CHAPTER 2

---

### Installation

---

---

**Hint:** This document describes how to install KIWI. Apart from the preferred method to install KIWI via rpm, it is also provided on pypi and can be installed via pip.

---

KIWI can be installed with different methods. For this guide, only the installation as a packages through a package manager is described.

Packages for the new KIWI version are provided at the [openSUSE buildservice](#).

To install KIWI, do:

1. Open the URL <http://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 in your shell. The `DIST` placeholder contains the respective distribution. Use **zypper ar** to add it to your list of repositories:

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

4. Install KIWI:

---

**Note:** Multipython packages

This version of KIWI is provided for python 2 and python 3 versions. The following information is based on the installation of the python3-kiwi package

---

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

## 2.1 Example Appliance Descriptions

For use with the next generation KIWI there is also a GitHub project hosting example appliance descriptions. Users who need an example to start with should checkout the project as follows:

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



# CHAPTER 3

---

## Quick Start

---

---

### Hint: Abstract

This document describes how to start with KIWI, an OS appliance builder. This description applies for version 9.16.19.

---

## 3.1 Before you start

Make sure you have installed KIWI and the example collection as explained in [Installation](#)

## 3.2 Build your first image

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

```
$ sudo kiwi-ng --type vmx system build \  
    --description kiwi-descriptions/suse/x86_64/suse-leap-42.3-JeOS_\  
↪ \  
    --target-dir /tmp/myimage
```

Find the image with the suffix `.raw` below `/tmp/myimage`.

## 3.3 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 our case QEMU is used, and boot it as follows:

```
$ qemu \  
  -boot c  
  -drive file=LimeJeOS-Leap-42.3.x86_64-1.42.3.raw,format=raw,  
  ↪if=virtio \  
  -m 4096
```

# CHAPTER 4

---

## Building Images

---

---

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

---

### 4.1 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. A useful pocket system for testing and demo and debugging purposes.

The following example shows how to build a live ISO image based on openSUSE Leap:

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

```
$ sudo kiwi-ng --type iso system build \  
    --description kiwi-descriptions/suse/x86_64/suse-leap-42.3-  
→JeOS \  
    --target-dir /tmp/myimage
```

Find the image with the suffix `.iso` below `/tmp/myimage`.

### 3. Test the live image with QEMU:

```
$ qemu -cdrom LimeJeOS-Leap-42.3.x86_64-1.42.3.iso -m 4096
```

After the test was successful, the image is 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.

## 4.2 Build a Virtual Disk Image

### Abstract

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

- how to build a vmx image
- how to run it with QEMU

A simple disk image represents the system disk, useful for cloud frameworks like Amazon EC2, Google Compute Engine or Microsoft Azure.

The following example shows how to build a simple disk image based on openSUSE Leap and ready to run in QEMU:

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

```
$ sudo kiwi-ng --type vmx system build \  
    --description kiwi-descriptions/suse/x86_64/suse-leap-42.3-  
→JeOS \  
    --target-dir /tmp/myimage
```

Find the image with the suffix `.raw` below `/tmp/myimage`.

### 3. Test the live image with QEMU:

```
$ qemu \  
    -drive file=LimeJeOS-Leap-42.3.x86_64-1.42.3.raw,format=raw,  
→if=virtio \  
    -m 4096
```

After the test was successful, the image is complete. For further information how to setup the image to work within a cloud framework see:

- [KIWI Image Description for Amazon EC2](#)
- [KIWI Image Description for Microsoft Azure](#)
- [KIWI Image Description for Google Compute Engine](#)

## 4.3 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:

```
$ sudo kiwi-ng --type oem system build \  
    --description kiwi-descriptions/suse/x86_64/suse-  
↪leap-42.3-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.

### 4.3.1 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'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's OEM installer deploy the OEM disk image from the network onto the target disk

### 4.3.2 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-42.3.x86_64-1.42.3.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.

### 4.3.3 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 -cdrom LimeJeOS-Leap-42.3.x86_64-1.42.3.install.iso -hda ↵
↵target_disk -boot d -m 4096
```

---

#### Note: USB Stick Deployment

Like any other iso image built with KIWI, 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*

---

### 4.3.4 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-42.3-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-42.3.x86_64-1.42.3.install.tar.xz` file to a temporary directory and copy the `initrd` and kernel images to the PXE server:

```
# Unpack installation tarball
mkdir /tmp/pxe && cd /tmp/pxe
tar -xf LimeJeOS-Leap-42.3.x86_64-1.42.3.install.tar.xz
```

(continues on next page)

(continued from previous page)

```
# Copy kernel and initrd used for pxe boot
scp pxeboot.initrd.xz PXE_SERVER_IP:/srv/tftpboot/boot/initrd
scp pxeboot.kernel PXE_SERVER_IP:/srv/tftpboot/boot/linux
```

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

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

```
# Copy system image and MD5 checksum
scp LimeJeOS-Leap-42.3.xz PXE_SERVER_IP:/srv/tftpboot/image/
scp LimeJeOS-Leap-42.3.md5 PXE_SERVER_IP:/srv/tftpboot/image/

# Copy kernel and initrd used for booting the system via kexec
scp LimeJeOS-Leap-42.3.initrd PXE_SERVER_IP:/srv/tftpboot/image/
scp LimeJeOS-Leap-42.3.kernel PXE_SERVER_IP:/srv/tftpboot/image/
```

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-42.3.xz
```

The location of the image is specified as a source URI which can point to any location supported by the `curl` command. KIWI 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.

---

**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>

---



## 4.4 Build a PXE Root File System Image

### Abstract

This page explains how to build a file system image for use with KIWI'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, 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'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 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:

```
$ sudo kiwi-ng --type pxe system build \  
  --description kiwi-descriptions/suse/x86_64/suse-  
↳ leap-42.3-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-42.3.x86_64-1.42.3 /srv/tftpboot/  
↳ image  
$ cp LimeJeOS-Leap-42.3.x86_64-1.42.3.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 [PXE Client Setup Configuration](#).

## 7. Create the image client configuration file:

```
$ vi /srv/tftpboot/KIWI/config.default  
  
IMAGE=/dev/ram1;LimeJeOS-Leap-42.3.x86_64;1.42.3;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 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
```

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

The Docker configuration metadata is provided to KIWI 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 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` repository to your list:

```
$ zypper ar -f http://download.opensuse.org/repositories/  
→Virtualization:/containers/<DIST>
```

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

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

```
$ zypper in umoci skopeo
```

4. Build the image with KIWI:

```
$ 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
```

## 4.6 Building in a Self-Contained Environment

---

### Hint: Abstract

Users building images with KIWI 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 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 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.

---

### 4.6.1 Requirements

The following components needs to be installed on the build system:

- Dice - a containment build system for KIWI.
- Docker - a container framework based on the Linux container support in the kernel.
- Docker Image - a docker build container for KIWI.
- 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 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 build.

### 4.6.2 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/SUSE/dice>

```
$ sudo zypper in ruby[VERSION]-rubygem-dice
```

### 4.6.3 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
```

### 4.6.4 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 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
```

---

## 4.6.5 Installing and Setting up Vagrant

---

**Note:** Optional step

By default Dice shares the KIWI 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 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
```

## 4.6.6 Configuring Dice

If you build in a contained environment, there is no need to have KIWI installed on the host system. KIWI is part of the container and is only called there. However, a KIWI image description and some metadata defining how to run the container are required as input data.

## 4.6.7 Selecting a KIWI Template

If you don't have a KIWI description select one from the templates provided at the GitHub project hosting example appliance descriptions.

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

The descriptions hosted here also provides a default `Dicefile` as part of each image description.

## 4.6.8 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
```

## 4.6.9 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-42.3-JeOS
$ dice status suse/x86_64/suse-leap-42.3-JeOS
```

## 4.6.10 Buildsystem Backends

Dice currently supports three build system backends:

1. Host buildsystem - Dice builds on the host like if you would call KIWI 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.

## 4.6.11 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. 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 build jobs.

## 4.6.12 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 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.

---

## 4.6.13 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.

### 4.6.14 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
```

### 4.6.15 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.

### 4.6.16 The Vagrantfile

The existence of a Vagrantfile tells Dice to use Vagrant as Buildsysteem. 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 build process. Vagrant will take over the task to run and manage the docker container via the `docker` tool chain.

### 4.6.17 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.

### 4.6.18 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.

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.

### 4.6.19 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 build process. Vagrant will take over the task to run and manage the virtual machines via the libvirt tool chain.

## 4.7 Building in the Open Buildservice

---

#### Hint: Abstract

This document gives a brief overview how to build images with KIWI in version 9.16.19 inside of the Open Buildservice. A tutorial on the Open Buildservice itself can be found here: [https://en.opensuse.org/openSUSE:Build\\_Service\\_Tutorial](https://en.opensuse.org/openSUSE:Build_Service_Tutorial)

---

The next generation KIWI is fully integrated with the 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` at:

<https://build.opensuse.org>

In order to use the next generation KIWI to build an appliance in the build service it is required to add the Builder project as repository to the KIWI XML configuration like in the following example:

```
<repository type="rpm-md" alias="kiwi-next-generation">
  <source path="obs://Virtualization:Appliances:Builder/Factory"/>
</repository>
```

## 4.8 Working with Images

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

### 4.8.1 Deploy ISO Image on an USB Stick

#### Abstract

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

- *Build an ISO Hybrid Live Image*

In KIWI 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-42.3.x86_64-1.42.3.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.

## 4.8.2 Deploy ISO Image as File on a FAT32 Formated USB Stick

### Abstract

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

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

In KIWI, 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 provides this feature upstream called as “iso-scan”. Therefore all KIWI 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](http://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-42.3.x86_64-1.42.3.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.

### 4.8.3 KIWI Image Description for Amazon EC2

#### Abstract

This page provides further information for handling vmx images built with KIWI 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 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"/>
<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 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
```

(continues on next page)

(continued from previous page)

```
cloud_config_modules:
- 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
    gecos: "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](#)

## 4.8.4 KIWI Image Description for Microsoft Azure

### Abstract

This page provides further information for handling vmx images built with KIWI 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 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](#)

## 4.8.5 KIWI Image Description for Google Compute Engine

### Abstract

This page provides further information for handling vmx images built with KIWI 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 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 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'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'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 NON_
→PERSISTENT_DEVICE_NAMES=1"
  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](#)

## 4.8.6 Setting Up a Network Boot Server

**Abstract**

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

- *[Build a PXE Root File System Image](#)*

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

**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
```

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

# The boot filename, Server name, Server Ip Address
```

(continues on next page)

(continued from previous page)

```
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
```

## 4.8.7 Setting Up YaST at First Boot

### Abstract

This page provides information how to setup the KIWI 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 XML description as follows:

1. Edit the KIWI 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 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 `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
```

## 4.8.8 PXE Client Setup Configuration

### Abstract

This page provides further information for handling PXE images built with KIWI 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"
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"
```

(continues on next page)

(continued from previous page)

```
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.

## 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 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.

## 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-42.3-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-leap42.3"/>
```

```
IMAGE="/dev/sda2;LimeJeOS-Leap-42.3.x86_64;1.42.3;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.

## 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-42.3-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-leap42.3"/>
```

```
RAID='1;/dev/sda;/dev/sdb'  
IMAGE="/dev/md1;LimeJeOS-Leap-42.3.x86_64;1.42.3;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.

## 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'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*

### 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

### 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

### 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
```

### 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

---



## 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.

---

## Setup a Different Download Protocol and Server

By default all downloads controlled by the KIWI 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 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

---

### 4.8.9 Gracefully Uninstall System Packages

#### Abstract

This page provides some details about uninstalling packages and how it could be used in order to remove packages once the image configuration, using the `config.sh` script, is done.

Uninstalling packages from the system image that were previously installed during the installation phase is an operation that can be handy under certain circumstances. As an example, someone could be interested in performing some configuration tasks in the `config.sh` script (see [prepare step](#) for further details). That would require to include some extra packages, which are only needed at build time. One example would be compiling some unpacked application sources.

KIWI description file schema defines package requests of type `uninstall` and type `delete`:

- The `uninstall` requests perform a clean packages removal by removing any package dependent on the requested ones and also removing orphan dependencies.
- The `delete` requests perform a hard removal without any dependency check, thus only listed packages are deleted even if it breaks dependencies or compromises any underlying package database.

This page focuses on `uninstall` package requests.

This is an example of the package requests in a description of a Container image that removes user related tools and development tools:

```
<packages type="image">
  <package name="ca-certificates"/>
  <package name="ca-certificates-mozilla"/>
  <package name="coreutils"/>
  <package name="iputils"/>
  <package name="openSUSE-build-key"/>
  <package name="krb5"/>
  <package name="netcfg"/>
  <package name="kubic-locale-archive"/>
  <package name="make"/>
  <package name="llvm-clang"/>
  <archive name="foo_app_sources.tar.gz"/>
</packages>
<!-- These packages will be uninstalled after running config.sh -->
<packages type="uninstall">
  <package name="shadow"/>
  <package name="make"/>
  <package name="llvm-clang"/>
</packages>
```

In the previous example after installing all the packages and archives, image repositories are configured and then the `config.sh` script is executed. In `config.sh` the `foo_app_sources.tar.gz` could be compiled using the `make` and `llvm` packages with something like a `make install` call. It is a common practice to build tiny and single purpose container images, thus makes sense to remove unneeded packages, like `make` and `llvm-clang`. To gracefully remove them, they have been included into the *type="uninstall"* packages list. Those packages will be removed including a dependency cleanup.

**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.

In the above example also the *shadow* package is being removed, again, in this specific case, it

is not expected to be needed in the final image. The *shadow* package mainly provides tools to handle user accounts. In a container image, once everything is installed and configured, it is not expected to require any further user account modification to the image, tools such as *useradd* or *usermod* will not be required.

## 4.8.10 KIWI Image Description for Vagrant

### Abstract

This page provides further information for handling VMX images built with KIWI 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, use **veewee** which builds boxes based on AutoYaST. As an alternative, use **Packer**, which is provided by Vagrant itself.

Both tools are based on the official installation media (DVDs) as shipped by the distribution vendor.

The KIWI way of building images might be helpful, if such a media does not exist. For example, if the distribution is still under development or you want to use a collection of your own repositories.

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

A Vagrant box which is able to work with Vagrant has to comply with the constraints documented in **Vagrant Box Constraints**. Applied to the referenced KIWI image description from [Build a Virtual Disk Image](#), the following changes are required:

1. Add mandatory packages

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

2. 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.

### 3. 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. Change this as you see fit.

### 4. Integrate public SSH key

Reach out to [Insecure Keys](#) for details on this keys. Add the public key to the box as overlay file in your image description at `home/vagrant/.ssh/authorized_keys`

### 5. Setup and start SSH daemon

In `config.sh` add the start of the `sshd` and the initial creation of machine keys as follows:

```
#=====
# Create ssh machine keys
#-----
/usr/sbin/sshd-gen-keys-start

#=====
# Activate services
#-----
suseInsertService sshd
```

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

### 6. Configure sudo for Vagrant user

Add the `etc/sudoers` file to the box as overlay file and make sure the user: **vagrant** is configured to allow passwordless root permissions.

An image built with the above setup creates a box file with the extension `.vagrant.libvirt.box`. That box file can now be added to vagrant with the command:

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

---

**Note:** Using the box requires a correct Vagrant installation on your machine. The libvirt daemon and the libvirt default network need to be running.

---

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
```

---

**Note:** Vagrant Providers

Currently, KIWI only supports the libvirt provider. There are other providers like virtualbox and also vmware available which requires a different box layout currently not supported by KIWI.

---

## 4.8.11 Booting a Live ISO Image from Network

### Abstract

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

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

In KIWI, 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-42.3.x86_64-1.42.3.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. 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-42.3.x86_64-1.42.3.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 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 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 **vbladed** 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
```

- 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'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

## 4.9 Supported Distributions

KIWI can build the above image types for distributions which are **equal** or **newer** compared to the following list:

- CentOS 7
- Fedora 25
- openSUSE Leap 42
- Red Hat Enterprise 7
- SUSE Linux Enterprise 12
- Tumbleweed
- Ubuntu Xenial

For anything older please consider to use the legacy KIWI version *v7.x* For more details on the legacy kiwi, see: *[Legacy KIWI vs. Next Generation](#)*.

# CHAPTER 5

---

## KIWI Commands

---

---

**Hint:** This document provides a list of the existing KIWI commands for version 9.16.19.

---

### 5.1 kiwi

#### 5.1.1 SYNOPSIS

```
kiwi [global options] service <command> [<args>]

kiwi -h | --help
kiwi [--profile=<name>...]
    [--type=<build_type>]
    [--logfile=<filename>]
    [--debug]
    [--color-output]
    image <command> [<args>...]
kiwi [--debug]
    [--color-output]
    result <command> [<args>...]
kiwi [--profile=<name>...]
    [--shared-cache-dir=<directory>]
    [--type=<build_type>]
    [--logfile=<filename>]
    [--debug]
    [--color-output]
    system <command> [<args>...]
```

(continues on next page)



(continued from previous page)

```
kiwi compat <legacy_args>...  
kiwi -v | --version  
kiwi help
```

## 5.1.2 DESCRIPTION

KIWI 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/SUSE/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 operates in two steps. The system build command combines both steps into one to make it easier to start with KIWI. The first step is the preparation step and if that step was successful, a creation step follows which is able to create different image output types.

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 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.

## 5.1.3 GLOBAL OPTIONS

- |                                   |   |
|-----------------------------------|---|
| <b>--color-output</b>             | 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. |
| <b>--debug</b>                    | Print debug information on the commandline.   |
| <b>--logfile=&lt;filename&gt;</b> | 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 /var/cache/kiwi
- type=<build\_type>** Select image build type. The specified build type must be configured as part of the XML description.
- version** Show program version

### 5.1.4 EXAMPLE

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

$ kiwi --type vmx system build \
  --description kiwi-descriptions/suse/x86_64/suse-leap-42.3-JeOS_
↪ \
  --target-dir /tmp/myimage
```

### 5.1.5 RUNTIME CONFIG FILE

To control custom parameters of the tool chain used by KIWI a user specific configuration file can be provided as:

```
~/.config/kiwi/config.yml
```

The contents of the file is in YAML format and supports the following setup parameters:

```
xz:
  - options: -a -b -c

  # Specifies XZ-compression-options
  # For details see man xz

obs:
  - download_url: url

  # Specifies download server url of an open buildservice instance
  # defaults to: http://download.opensuse.org/repositories

  - public: true|false

  # Specifies if the buildservice instance is public or private
  # defaults to: true
```

## 5.1.6 COMPATIBILITY

This version of KIWI uses a different caller syntax compared to former versions. However there is a compatibility mode which allows to use a legacy KIWI commandline as follows:

```
$ kiwi compat \  
    --build kiwi-descriptions/suse/x86_64/suse-leap-42.3-JeOS \  
    --type vmx -d /tmp/myimage
```

## 5.2 kiwi result list

### 5.2.1 SYNOPSIS

```
kiwi [global options] service <command> [<args>]  
  
kiwi result list -h | --help  
kiwi result list --target-dir=<directory>  
kiwi result list help
```

### 5.2.2 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

### 5.2.3 OPTIONS

**--target-dir=<directory>** directory containing the kiwi build results

## 5.3 kiwi result bundle

### 5.3.1 SYNOPSIS

```
kiwi [global options] service <command> [<args>]  
  
kiwi result bundle -h | --help  
kiwi result bundle --target-dir=<directory> --id=<bundle_id> --  
    bundle-dir=<directory>  
    [--zsync_source=<download_location>]  
kiwi result bundle help
```

## 5.3.2 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.

## 5.3.3 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 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

## 5.4 kiwi system prepare

### 5.4.1 SYNOPSIS

```
kiwi [global options] service <command> [<args>]

kiwi system prepare -h | --help
kiwi 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>...]
  [--delete-package=<name>...]
```

(continues on next page)

(continued from previous page)

```
[--signing-key=<key-file>...]
kiwi system prepare help
```

## 5.4.2 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
```

## 5.4.3 OPTIONS

- 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.

## 5.5 kiwi system update

### 5.5.1 SYNOPSIS

```
kiwi [global options] service <command> [<args>]

kiwi system update -h | --help
kiwi system update --root=<directory>
    [--add-package=<name>...]
    [--delete-package=<name>...]
kiwi system update help
```

### 5.5.2 DESCRIPTION

Update a previously prepared 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

### 5.5.3 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.

## 5.6 kiwi system build

### 5.6.1 SYNOPSIS

```
kiwi [global options] service <command> [<args>]

kiwi system build -h | --help
kiwi 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>...]
    [--delete-package=<name>...]
    [--signing-key=<key-file>...]
kiwi system build help
```

### 5.6.2 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.

### 5.6.3 OPTIONS

- 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.

## 5.6.4 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](http://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

## 5.7 kiwi system create

### 5.7.1 SYNOPSIS

```
kiwi [global options] service <command> [<args>]

kiwi system create -h | --help
kiwi system create --root=<directory> --target-dir=<directory>
    [--signing-key=<key-file>...]
kiwi system create help
```

## 5.7.2 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

## 5.7.3 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

## 5.8 kiwi image resize

### 5.8.1 SYNOPSIS

```
kiwi [global options] service <command> [<args>]

kiwi image resize -h | --help
kiwi image resize --target-dir=<directory> --size=<size>
    [--root=<directory>]
kiwi image resize help
```

### 5.8.2 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.

### 5.8.3 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

## 5.9 kiwi image info

### 5.9.1 SYNOPSIS

```
kiwi [global options] service <command> [<args>]

kiwi image info -h | --help
kiwi image info --description=<directory>
    [--resolve-package-list]
    [--ignore-repos]
    [--add-repo=<source,type,alias,priority>...]
kiwi image info help
```

### 5.9.2 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.

### 5.9.3 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.

# CHAPTER 6

---

## Development and Contributing

---

---

### Hint: Abstract

This document describes the development process of KIWI and how you can be part of it. This description applies for version 9.16.19.

---

## 6.1 Using KIWI NG in a Python Project

---

### Hint: Abstract

KIWI is provided as python module under the **kiwi** namespace. It is available for the python 2 and 3 versions. The following description applies for KIWI version 9.16.19.

---

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

# Import of log handler only needed if default logging
# setup is not appropriate for the project
```

(continues on next page)

(continued from previous page)

```
# from kiwi.logger import log

# By default the logging level is set to DEBUG, which
# can be changed by the following call
# log.setLogLevel(logging.INFO)

# Logging can also be disabled completely
# log.disabled = True

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',
    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. . .

## 6.2 API Documentation 9.16.19

### 6.2.1 Subpackages

#### kiwi.archive Package

##### Submodules

#### 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

## 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

## Module Contents

### kiwi.boot Package

### Subpackages

### kiwi.boot.image Package

### Submodules

### 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*)

Implements creation of the initrd

**Parameters**

- **mbrid** (*object*) – instance of ImageIdentifier
- **basename** (*string*) – base initrd file name

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 this boot image

Implementation in specialized boot image class

**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*)

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

**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

**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

## 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*)

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

**get\_boot\_names()**

Provides kernel and initrd names for kiwi boot image

### Returns

Contains boot\_names\_type tuple

```
boot_names_type (
    kernel_name='INSTALLED_KERNEL',
    initrd_name='DRACUT_OUTPUT_NAME'
)
```

### Return type tuple

**include\_file**(*filename*)

Include file to dracut boot image

**Parameters** **filename**(*string*) – file path name

**post\_init()**

Post initialization method

Initialize empty list of dracut caller options

**prepare()**

Prepare kiwi profile environment to be included in dracut initrd

## **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*)

Create initrd from prepared boot system tree and compress the result

#### **Parameters**

- **mbrid** (*object*) – instance of ImageIdentifier
- **basename** (*string*) – base initrd file name

**get\_boot\_names()**

Provides kernel and initrd names for kiwi boot image

#### **Returns**

Contains boot\_names\_type tuple

```
boot_names_type (  
    kernel_name='linux.vmx',  
    initrd_name='initrd.vmx'  
)
```

#### **Return type** tuple

**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

## 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

## Module Contents

**kiwi.bootloader Package**

**Subpackages**

**kiwi.bootloader.config Package**

**Submodules**

**kiwi.bootloader.config.base Module**

**class** `kiwi.bootloader.config.base.BootLoaderConfigBase` (*xml\_state*,  
*root\_dir*,  
*cus-*  
*tom\_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\_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** (*string*) – kernel options as string

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

## **kiwi.bootloader.config.grub2 Module**

```
class kiwi.bootloader.config.grub2.BootLoaderConfigGrub2 (xml_state,  
                                                         root_dir,  
                                                         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, root_uuid, hypervisor='xen.gz',  
                        kernel='linux.vmx',    initrd='initrd.vmx',  
                        boot_options="")
```

Create the grub.cfg in memory from a template suitable to boot from a disk image

#### 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**(*string*) – kernel options as string

```
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',    ker-  
                        nel='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

```
setup_sysconfig_bootloader()
```

Create or update etc/sysconfig/bootloader by the following parameters required according to the grub2 bootloader setup

- **LOADER\_TYPE**
- **LOADER\_LOCATION**
- **DEFAULT\_APPEND**
- **FAILSAFE\_APPEND**

```
write()
```

Write grub.cfg and etc/default/grub file

### kiwi.bootloader.config.isolinux Module

```
class kiwi.bootloader.config.isolinux.BootLoaderConfigIsoLinux(xml_state,  
                                                             root_dir,  
                                                             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

The mbrid parameter is not used, because only isolinux loader binary and possible theming files are copied

#### Parameters

- **mbrid**(*string*) – unused
- **lookup\_path**(*string*) – custom module lookup path

**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

Calls setup\_install\_boot\_images because no different action required

**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

## kiwi.bootloader.config.zipl Module

**class** kiwi.bootloader.config.zipl.BootLoaderConfigZipl(*xml\_state*,  
*root\_dir*,  
*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='linux.vmx', initrd='initrd.vmx', 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** (*string*) – kernel options as string

**write** ()

Write zipl config file

## 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

## kiwi.bootloader.install Package

## Submodules

## `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

## `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 boot-loader arguments

```
{
    'target_removable': bool,
    'system_volumes': list_of_volumes,
    'firmware': FirmWare_instance,
    'efi_device': string,
    'boot_device': string,
    'root_device': string
}
```

## 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 }
```

## 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

## `kiwi.bootloader.template` Package

### Submodules

#### `kiwi.bootloader.template.grub2` Module

**class** `kiwi.bootloader.template.grub2.BootLoaderTemplateGrub2`

Bases: `object`

**grub2 configuraton file templates**

**get\_disk\_template** (*failsafe=True, hybrid=True, terminal='gfxterm'*)

Bootloader configuration template for disk image

**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\_install\_template** (*failsafe=True, hybrid=True, terminal='gfxterm'*)

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\_disk\_template** (*failsafe=True, terminal='gfxterm'*)

Bootloader configuration template for disk image 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\_install\_template** (*failsafe=True, terminal='gfxterm'*)

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`



**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*)

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*)

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, check-iso=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`

## `kiwi.bootloader.template.zipl` Module

**class** `kiwi.bootloader.template.zipl.BootLoaderTemplateZipl`

Bases: `object`

**zipl configuraton file templates**

**get\_template** (*failsafe=True*)

Bootloader configuration template for disk boot

**Parameters** **failsafe** (*bool*) – with failsafe true/false

**Returns** instance of `Template`

**Return type** `Template`

## Module Contents

## Module Contents

## `kiwi.builder` Package

## Submodules

## `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

## 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 tarball including container specific metadata.

Image types which triggers this builder are:

- image="docker"

**Returns** result

**Return type** instance of Result

## 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`

## `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\_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

## 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"`

## **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`

## **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*

## Module Contents

**class** *kiwi.builder.ImageBuilder*

Bases: *object*

image builder factory

## kiwi.container Package

### Subpackages

### kiwi.container.setup Package

### Submodules

### 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

**create\_fstab()**

Container boot mount setup

Initialize an empty fstab file, mount processes in a container are controlled by the container infrastructure

**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

### **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
```

### **Module Contents**

```
class kiwi.container.setup.ContainerSetup
    Bases: object

    container setup factory
```

### **Submodules**

#### **kiwi.container.docker Module**

```
class kiwi.container.docker.ContainerImageDocker (root_dir, cus-
                                                    tom_args=None)

    Bases: kiwi.container.oci.ContainerImageOCI

    Create docker container from a root directory

    pack_image_to_file (filename)
        Packs the given oci image into the given filename.

        Parameters filename (string) – file name of the resulting packed
        image
```

#### **kiwi.container.oci Module**

```
class kiwi.container.oci.ContainerImageOCI (root_dir,          cus-
                                                    tom_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': [
        '-config.entrypoint=/bin/bash',
        '-config.entrypoint=-x'
    ],
    'entry_subcommand': [
        '-config.cmd=ls',
        '-config.cmd=-l'
    ],
    'maintainer': ['-author=tux'],
    'user': ['-config.user=root'],
    'workingdir': ['-config.workingdir=/root'],
    'expose_ports': [
        '-config.exposedports=80',
        '-config.exposedports=42'
    ],
    'volumes': [
        '-config.volume=/var/log',
        '-config.volume=/tmp'
    ],
    'environment': ['-config.env=PATH=/bin'],
    'labels': ['-config.label=name=value']
}
```

**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

**pack\_image\_to\_file** (*filename*)

Packs the oci image into the given filename.

**Parameters filename** (*string*) – file name of the resulting packed image

## 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

## kiwi.filesystem Package

### Submodules

#### 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*)

Store dictionary of custom arguments if not empty. This overrides the default custom argument hash

```
{
    'create_options': ['option'],
    'mount_options': ['option'],
    'meta_data': {
        'key': 'value'
    }
}
```

## Copy root data tree into filesystem

## kiwi.filesystem.btrfs Module

## Implements creation of btrfs filesystem

## Create btrfs filesystem on block device

## kiwi.filesystem.clicfs Module

## Implements creation of clicfs filesystem

## Create clicfs filesystem from data tree

## 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 embeded filesystem

Parameters **custom\_args** (*dict*) – unused

## **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

## **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

## **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

### **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

### **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

### **kiwi.filesystem.iso9660 Module**

```
class kiwi.filesystem.iso9660.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

## 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

## kiwi.filesystem.squashfs Module

**class** `kiwi.filesystem.squashfs.FileSystemSquashFs` (*device\_provider*,  
*root\_dir=None*,  
*cus-*  
*tom\_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



## 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

## 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

## kiwi.iso\_tools Package

### Submodules

## 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

## **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

## **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.

**classmethod 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

**classmethod 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

**classmethod 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

**classmethod 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

## Module Contents

**class** `kiwi.iso_tools.IsoTools`

Bases: `object`

**IsoTools** factory

## `kiwi.package_manager` Package

### Submodules

#### `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
- **root\_bind** (*object*) – instance of `RootBind`
- **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()**

Check if package database is consistent

Implementation in specialized package manager class

**dump\_reload\_package\_database** (*version=45*)

For rpm based package managers, dump and reload the database to match the desired rpm db version

Implementation in specialized package manager class

**Parameters** **version** (*str*) – unused

**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

**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 method, only 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

## **kiwi.package\_manager.yum Module**

```
class kiwi.package_manager.yum.PackageManagerYum (repository,  
                                                    cus-  
                                                    tom_args=None)  
Bases: kiwi.package_manager.base.PackageManagerBase
```

**Implements base class for installation/deletion of packages and collections using yum**

### **Parameters**

- **yum\_args** (*list*) – yum arguments from repository runtime configuration
- **command\_env** (*dict*) – yum command environment from repository runtime configuration

**database\_consistent** ()

Check if rpm package database is consistent

**Returns** True or False

**Return type** bool

**dump\_reload\_package\_database** (*version=45*)

Dump and reload the rpm database to match the desired rpm db version

For the supported RHEL versions there is no dump/reload cycle required

**Parameters** **version** (*str*) – unused

**match\_package\_deleted** (*package\_name*, *yum\_output*)

Match expression to indicate a package has been deleted

**Parameters**

- **package\_list** (*list*) – list of all packages
- **log\_line** (*str*) – yum status line

**Returns** match or None if there isn't any match

**Return type** match object, None

**match\_package\_installed** (*package\_name*, *yum\_output*)

Match expression to indicate a package has been installed

This match for the package to be installed in the output of the yum 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*) – yum 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 yum arguments

**Parameters** **custom\_args** (*list*) – custom yum arguments

**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*) – yum 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 rhel repo data

**Parameters** **name** (*str*) – unused

**update()**

Process package update requests (chroot)

**Returns** process results in command type

**Return type** namedtuple

## **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

**database\_consistent** ()

Check if rpm package database is consistent

**Returns** True or False

**Return type** bool

**dump\_reload\_package\_database** (*version=45*)

Dump and reload the rpm database to match the desired rpm db version Supported target rpm database db versions are

- 45 (db45\_load)
- 48 (db48\_load)

**Parameters** **version** (*str*) – target rpm db version

**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

**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

## 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

## kiwi.partitionner Package

### Submodules

#### 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

## **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

## **kiwi.partitioners.gpt Module**

**class** `kiwi.partitioners.gpt.PartitionerGpt` (*disk\_provider*,  
*start\_sector=None*)

Bases: `kiwi.partitioners.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)

## **kiwi.partitioners.msdos Module**

**class** `kiwi.partitioners.msdos.PartitionerMsDos` (*disk\_provider*,  
*start\_sector=None*)  
Bases: `kiwi.partitioners.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

## **Module Contents**

**class** `kiwi.partitioners.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

## kiwi.repository Package

### Subpackages

## kiwi.repository.template Package

### Submodules

#### kiwi.repository.template.apt Module

**class** `kiwi.repository.template.apt.PackageManagerTemplateAptGet`

Bases: `object`

apt-get configuration file template

**get\_host\_template** (*exclude\_docs=False*)

apt-get package manager template for apt-get called outside of the image, not chrooted

**Return type** `Template`

**get\_image\_template** (*exclude\_docs=False*)

apt-get package manager template for apt-get called inside of the image, chrooted

**Return type** `Template`

## Module Contents

### Submodules

#### 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*)

Add repository

Implementation in specialized repository class

#### Parameters

- **name** (*str*) – unused
- **uri** (*str*) – unused
- **repo\_type** – unused
- **prio** (*int*) – unused
- **dist** – unused
- **components** – unused
- **user** – unused
- **secret** – unused
- **credentials\_file** – unused
- **repo\_gpgcheck** – unused
- **pkg\_gpgcheck** – 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

**use\_default\_location** ()

Call repository operations with default repository manager setup

Implementation in specialized repository class

## **kiwi.repository.yum Module**

**class** `kiwi.repository.yum.RepositoryYum` (*root\_bind*, *custom\_args=None*)

Bases: `kiwi.repository.base.RepositoryBase`

**Implements repository handling for yum package manager**

### **Parameters**

- **shared\_yum\_dir** (*str*) – shared directory between image root and build system root
- **runtime\_yum\_config\_file** (*str*) – yum runtime config file name
- **command\_env** (*dict*) – customized os.environ for yum
- **runtime\_yum\_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*)

Add yum repository

### **Parameters**

- **name** (*str*) – repository base file name
- **uri** (*str*) – repository URI
- **repo\_type** – repository type name

- **prio** (*int*) – yum repostory priority
- **dist** – unused
- **components** – unused
- **user** – unused
- **secret** – unused
- **credentials\_file** – unused
- **repo\_gpgcheck** (*bool*) – enable repository signature validation
- **pkg\_gpgcheck** (*bool*) – enable package signature validation

**cleanup\_unused\_repos** ()

Delete unused yum 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 yum repositories

**delete\_repo** (*name*)

Delete yum repository

**Parameters** **name** (*str*) – repository base file name

**delete\_repo\_cache** (*name*)

Delete yum repository cache

The cache data for each repository is stored in a directory of the same name as the repository name. The method deletes this directory 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 yum arguments and create runtime configuration and environment

**Parameters** **custom\_args** (*list*) – yum arguments

**runtime\_config** ()

yum runtime configuration and environment

**Returns** yum\_args:list, command\_env:dict

**Return type** dict

**use\_default\_location** ()

Setup yum repository operations to store all data in the default places

```
class kiwi.repository.yum.RepositoryYumSpaceRemover (file_object)
    Bases: object
```

Yum is not able to deal with key = value pairs if there are spaces around the '=' delimiter.

This is a helper class to eliminate spaces around delimiters when writing out the data of a ConfigParser instance. In Python 3 ConfigParser.write() supports the parameter:

```
ConfigParser.write(file_object, space_around_delimiters=False)
```

Unfortunately, in Python 2 the spaces are hard coded and can't be configured. In order to still support both Python versions this helper class provides a custom `write()` method which deletes the unwanted spaces.

Instead of passing an instance of a file\_object to ConfigParser we pass an instance of a `RepositoryYumSpaceRemover` object:

```
ConfigParser.write(RepositoryYumSpaceRemover(file_object))
```

It is expected that ConfigParser.write() only calls the `write()` method of the usually provided file object. Thus this helper class only provides a custom version of this `write()` method.

```
write(data)
```

## kiwi.repository.zypper Module

```
class kiwi.repository.zypper.RepositoryZypper (root_bind,      cus-
                                             tom_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)  
Add zypper repository

**Parameters**

- **name** (*str*) – repository name
- **uri** (*str*) – repository URI
- **repo\_type** – repository type name
- **prio** (*int*) – zypper repository priority
- **dist** – unused
- **components** – unused
- **user** – credentials username
- **secret** – credentials password
- **credentials\_file** – zypper credentials file
- **repo\_gpgcheck** (*bool*) – enable repository signature validation
- **pkg\_gpgcheck** (*bool*) – enable package signature validation

**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

**use\_default\_location** ()

Setup zypper repository operations to store all data in the default places

## 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

## kiwi.storage Package

### Subpackages

#### kiwi.storage.subformat Package

### Subpackages

#### kiwi.storage.subformat.template Package

### Submodules

#### `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

## Module contents

### Submodules

#### `kiwi.storage.subformat.base` Module

**class** `kiwi.storage.subformat.base.DiskFormatBase` (*xml\_state, root\_dir, target\_dir, custom\_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. Subformats which creates additional metadata files needs to overwrite this method

**Parameters** **result** (*object*) – Instance of Result



## 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' }

## 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

## **kiwi.storage.subformat.qcow2 Module**

```
class kiwi.storage.subformat.qcow2.DiskFormatQcow2 (xml_state,  
                                                    root_dir,  
                                                    target_dir,  
                                                    cus-  
                                                    tom_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

## **kiwi.storage.subformat.vagrant\_libvirt Module**

```
class kiwi.storage.subformat.vagrant_libvirt.DiskFormatVagrantLibVirt (xml_  
                                                                           root_  
                                                                           tar_  
                                                                           get_  
                                                                           cus-  
                                                                           tom_
```

Bases: *kiwi.storage.subformat.base.DiskFormatBase*

**Create vagrant box for libvirt provider**

**create\_image\_format** ()

Create vagrant box for libvirt provider. This includes

- creation of qcow2 disk image format as box.img
- creation of box metadata.json
- creation of box Vagrantfile
- creation of result format tarball from the files created above

**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 }
```

### **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

### **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

## `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

## `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

## 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 dictio-  
        nary  
  
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
```

## 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
```

## Submodules

### 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

## 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

**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

## **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

**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*)

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

**create\_crypttab** (*filename*)

Create crypttab, setting the UUID of the storage device

**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

### **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

### **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

## **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

## Module Contents

### kiwi.system Package

#### Submodules

#### 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

## **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

## **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*)

Install system software using the package manager from the host, also known as bootstrapping

**Parameters** **manager** (*object*) – instance of a `PackageManager` subclass

**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

## **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** ()

Create bash quoted profile

**Returns** profile dump for bash

**Return type** `str`

**delete** (*key*)

## **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

**classmethod 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

**compress**

Alias for field number 2

**filename**

Alias for field number 0

**shasum**

Alias for field number 3

**use\_for\_bundle**

Alias for field number 1

**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 buildsystem 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**move\_to\_root** (*elements*)

Change the given path elements to a new root directory

**Parameters** **elements** (*list*) – list of path names**Returns** changed elements**Return type** `list`**setup\_intermediate\_config** ()

Create intermediate config files

Some config files e.g etc/hosts needs to be temporarily copied from the buildsystem host to the image root system in order to allow e.g DNS resolution in the way as it is configured on the buildsystem 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

### **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 static core device nodes
- create core system paths

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

### **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** (*entries*)

Create etc/fstab from given list of entries

Also lookup 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

**Parameters** **entries** (*list*) – list of line entries for 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*, *pre-serve\_owner\_group=False*)

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

**import\_shell\_environment** (*profile*)

Create profile environment to let scripts consume information from the XML description.

**Parameters** **profile** (*object*) – instance of `Profile`

**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\_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

## **kiwi.system.shell Module**

**class** `kiwi.system.shell.Shell`

Bases: `object`

### **Special character handling for shell evaluated code**

**classmethod** **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`

**classmethod** `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*

**classmethod** `run_common_function(name, parameters)`

Run a function implemented in config/functions.sh

**Parameters**

- **name** (*str*) – function name
- **parameters** (*list*) – function arguments

## **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

## `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, None otherwise

**Return type** str, None

**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

## **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

## Module Contents

### kiwi.solver Package

### Subpackages

### kiwi.solver.repository Package

### Submodules

#### kiwi.solver.repository.base Module

```
class kiwi.solver.repository.base.SolverRepositoryBase (uri,
                                                         user=None,
                                                         se-
                                                         cret=None)
```

Bases: object

**Base class interface for SAT solvable creation.**

- **param object uri** Instance of Uri

**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.

## Module Contents

```
class kiwi.solver.repository.SolverRepository  
Bases: object
```

Repository factory for creation of SAT solvables

- **param object uri** Instance of Uri

## Submodules

### 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

## Module Contents

### kiwi.tasks package

#### Submodules

#### 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

**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

### `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

### `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

### `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

## `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

## `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

## `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

## Module Contents

### `kiwi.utils` Package

### Submodules

## 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

## 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

## 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

**Return type** 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

## **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

## **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

**write** ()

Write back source file with changed content but in same order

## Module Contents

### kiwi.volume\_manager Package

#### Submodules

#### kiwi.volume\_manager.base Module

```
class kiwi.volume_manager.base.VolumeManagerBase (device_provider,  
                                                    root_dir,  
                                                    volumes, cus-  
                                                    tom_args=None)
```

Bases: *kiwi.storage.device\_provider.DeviceProvider*

Implements base class for volume management interface

##### Parameters

- **mountpoint** (*str*) – root mountpoint for volumes
- **device\_provider** (*object*) – instance of a *DeviceProvider* subclass
- **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** ()

Dictionary with instance of MappedDevice for the root device node

**Returns** root 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

## **kiwi.volume\_manager.btrfs Module**

```
class kiwi.volume_manager.btrfs.VolumeManagerBtrfs (device_provider,  
                                                    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** (*persistence\_type*='by-label', *filesystem\_name*=None)

Implements creation of the fstab entries. The method returns a list of fstab compatible entries

**Parameters**

- **persistence\_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

## **kiwi.volume\_manager.lvm Module**

```
class kiwi.volume_manager.lvm.VolumeManagerLVM(device_provider,  
                                              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**

- **persistence\_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

## Module Contents

**class** `kiwi.volume_manager.VolumeManager`

Bases: object

**VolumeManager factory**

**Parameters**

- **name** (*str*) – volume management name
- **device\_provider** (*object*) – instance of a class based on DeviceProvider
- **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

## 6.2.2 Submodules

### 6.2.3 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

## 6.2.4 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** importlib loaded module

**Return type** object

**show\_and\_exit\_on\_help\_request()**

Execute man to show the selected manual page

## 6.2.5 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

**classmethod** `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

**classmethod** `run` (*command*, *custom\_env=None*, *raise\_on\_error=True*)

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

#### Returns

Contains call results in command type

```
command(output='string', error='string',   
↪returncode=int)
```

**Return type** namedtuple

## 6.2.6 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

## 6.2.7 kiwi.defaults Module

**class** `kiwi.defaults.Defaults`

Bases: object

**Implements default values**

Provides class 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

**classmethod** **get\_archive\_image\_types**()

Provides list of supported archive image types

**Returns** archive names

**Return type** list

**classmethod** **get\_boot\_image\_description\_path**()

Provides the path to find custom kiwi boot descriptions

**Returns** directory path

**Return type** str

**classmethod** **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

**classmethod** **get\_buildservice\_env\_name**()

Provides the base name of the environment file in a buildservice worker

**Returns** file basename

**Return type** str

**classmethod** **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

**classmethod** **get\_container\_compression**()

Provides default container compression algorithm

**Returns** name

**Return type** str

**classmethod** **get\_container\_image\_types**()

Provides list of supported container image types

**Returns** container names

**Return type** list

**classmethod** `get_default_boot_mbytes()`

Provides default boot partition size in mbytes

**Returns** mbsize value

**Return type** int

**classmethod** `get_default_boot_timeout_seconds()`

Provides default boot timeout in seconds

**Returns** seconds

**Return type** int

**classmethod** `get_default_container_name()`

Provides the default container name.

**Returns** name

**Return type** str

**classmethod** `get_default_container_tag()`

Provides the default container tag.

**Returns** tag

**Return type** str

**classmethod** `get_default_disk_start_sector()`

Provides the default initial disk sector for the first disk partition.

**Returns** sector value

**Return type** int

**classmethod** `get_default_efi_boot_mbytes()`

Provides default EFI partition size in mbytes

**Returns** mbsize value

**Return type** int

**classmethod** `get_default_efi_partition_table_type()`

Provides the default partition table type for efi firmwares.

**Returns** partition table type name

**Return type** str

**classmethod** `get_default_firmware(arch)`

Provides default firmware for specified architecture

**Parameters** **arch** (*string*) – platform.machine

**Returns** firmware name

**Return type** str

**classmethod** `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

**classmethod** `get_default_legacy_bios_mbytes()`

Provides default size of bios\_grub partition in mbytes

**Returns** mbsize value

**Return type** int

**classmethod** `get_default_live_iso_root_filesystem()`

Provides default live iso root filesystem type

**Returns** filesystem name

**Return type** str

**classmethod** `get_default_live_iso_type()`

Provides default live iso union type

**Returns** live iso type

**Return type** str

**classmethod** `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

**classmethod** `get_default_prep_mbytes()`

Provides default size of prep partition in mbytes

**Returns** mbsize value

**Return type** int

**classmethod** `get_default_video_mode()`

Provides 800x600 default video mode as hex value for the kernel

**Returns** vesa video kernel hex value

**Return type** str

**classmethod** `get_default_volume_group_name()`

Provides default LVM volume group name

**Returns** name

**Return type** str

**classmethod** `get_disk_format_types()`

Provides supported disk format types

**Returns** disk types

**Return type** list

**classmethod** `get_disk_image_types()`

Provides supported disk image types

**Returns** disk image type names

**Return type** list

**classmethod** `get_dracut_conf_name()`

Provides file path of dracut config file to be used with KIWI

**Returns** file path name

**Return type** str

**classmethod** `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

**classmethod** `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

**classmethod** `get_efi_image_name(arch)`

Provides architecture specific EFI boot binary name

**Parameters** `arch` (*string*) – platform.machine

**Returns** name

**Return type** str

**classmethod** `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

**classmethod** `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

**classmethod** `get_failsafe_kernel_options()`

Provides failsafe boot kernel options

**Returns**

list of kernel options

`['option=value', 'option']`

**Return type** list

**classmethod** `get_filesystem_image_types()`

Provides list of supported filesystem image types

**Returns** filesystem names

**Return type** list

**classmethod** `get_firmware_types()`

Provides supported architecture specific firmware types

**Returns** firmware types per architecture

**Return type** dict

**classmethod** `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

**classmethod** `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

**classmethod** `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

**classmethod** `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

**classmethod** `get_grub_ofw_modules()`

Provides list of grub ofw modules (ppc)

**Returns** list of module names

**Return type** list

**classmethod** `get_grub_path(lookup_path)`

Provides grub install path

Depending on the distribution grub could be installed below a grub2 or grub directory. Therefore this information needs to be dynamically looked up

**Parameters** `lookup_path` (*string*) – path name

**Returns** grub2 install directory path name

**Return type** str

**classmethod** `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

**classmethod** `get_install_volume_id()`

Provides default value for ISO volume ID for install media

**Returns** name

**Return type** str

**classmethod** `get_iso_boot_path()`

Provides arch specific relative path to boot files on kiwi iso filesystems

**Returns** relative path name

**Return type** str

**classmethod** `get_iso_tool_category()`

Provides default iso tool category

**Returns** name

**Return type** str

**classmethod** `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

**classmethod** `get_live_image_types()`

Provides supported live image types

**Returns** live image type names

**Return type** list

**classmethod** `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

**classmethod** `get_lvm_overhead_mbytes()`

Provides empiric LVM overhead size in mbytes

**Returns** mbsize value

**Return type** int

**classmethod** `get_min_volume_mbytes()`

Provides default minimum LVM volume size in mbytes

**Returns** mbsize value

**Return type** int

**classmethod** `get_network_image_types()`

Provides supported pxe image types

**Returns** pxe image type names

**Return type** list

**classmethod** `get_obs_download_server_url()`

Provides the default download server url hosting the public open builds service repositories

**Returns** url path

**Return type** str

**classmethod** `get_preparer()`

Provides ISO preparer name

**Returns** name

**Return type** str

**classmethod** `get_publisher()`

Provides ISO publisher name

**Returns** name

**Return type** str

**classmethod** `get_recovery_spare_mbytes()`

Provides spare size of recovery partition in mbytes

**Returns** mbsize value

**Return type** int

**classmethod** `get_s390_disk_block_size()`

Provides the default block size for s390 storage disks

**Returns** blocksize value

**Return type** int

**classmethod** `get_s390_disk_type()`

Provides the default disk type for s390 storage disks

**Returns** type name

**Return type** str

**classmethod** `get_schema_file()`

Provides file path to kiwi RelaxNG schema

**Returns** file path

**Return type** str

**classmethod** `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

**classmethod** `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

**classmethod** `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

**classmethod** `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

**classmethod** `get_snapper_config_template_file` ()

Provides the default configuration template file for snapper

**Returns** file

**Return type** str

**classmethod** `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

**classmethod** `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

**classmethod** `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

**classmethod** `get_volume_id` ()

Provides default value for ISO volume ID

**Returns** name

**Return type** str

**classmethod** `get_xsl_stylesheet_file()`

Provides the file path to the KIWI XSLT style sheets

**Returns** file path

**Return type** str

**classmethod** `get_xz_compression_options()`

Provides compression options for the xz compressor

**Returns**

Contains list of options

[ '-option=value' ]

**Return type** list

**classmethod** `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

**classmethod** `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

**classmethod** `set_python_default_encoding_to_utf8()`

Set python default encoding to utf-8 if not already done

This is not a safe operation since `sys.setdefaultencoding()` was removed from `sys` on purpose when Python starts. Reenabling it and changing the default encoding can break code that relies on `ascii` being the default. Within the scope of `kiwi` the operation is safe because all data is expected to be utf-8 everywhere and considered a bug if this is not the case

**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

## 6.2.8 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.KiwiBootLoaderIsoLinuxPlatformError` (*message*)  
Bases: `kiwi.exceptions.KiwiError`

Exception raised if an attempt was made to use isolinux on an unsupported platform.

**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.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.KiwiFileNotFound` (*message*)  
Bases: `kiwi.exceptions.KiwiError`

Exception raised if the requested file could not be found.

**exception** `kiwi.exceptions.KiwiFileSystemServiceSetupError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if an attempt was made to build an unsupported or unspecified filesystem.

**exception** `kiwi.exceptions.KiwiFileSystemServiceSyncError` (*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.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.KiwiRepoTypeUnknown` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised if an unsupported repository type is specified for the corresponding package manager.

**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.KiwiRpmDatabaseReloadError` (*message*)

Bases: `kiwi.exceptions.KiwiError`

Exception raised on error of an rpm DB dump -> reload process.

**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.

## 6.2.9 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** True or False

**Return type** bool

**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

## 6.2.10 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

## 6.2.11 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')`]
- **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 [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

## 6.2.12 kiwi.logger Module

**class** `kiwi.logger.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.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`

**class** `kiwi.logger.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.**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.**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.**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

**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

**class** `kiwi.logger.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.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`

## 6.2.13 `kiwi.mount_manager` Module

**class** `kiwi.mount_manager.MountManager` (*device, mountpoint=None*)

Bases: `object`

**Implements methods for mounting, umounting 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

## 6.2.14 kiwi.path Module

**class** kiwi.path.Path

Bases: object

### Directory path helpers

**classmethod** **create** (*path*)

Create path and all sub directories to target

**Parameters** **path** (*string*) – path name

**classmethod** **remove** (*path*)

Delete empty path, causes an error if target is not empty

**Parameters** **path** (*string*) – path name

**classmethod** `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

**classmethod** `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

**classmethod** `which(filename, alternative_lookup_paths=None, custom_env=None, access_mode=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

**Returns** absolute path to file or None

**Return type** str

**classmethod** `wipe(path)`

Delete path and all contents

**Parameters** `path` (*string*) – path name

## 6.2.15 kiwi.privileges Module

**class** `kiwi.privileges.Privileges`

Bases: object

**Implements** check for root privileges

**classmethod** `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

## 6.2.16 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_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_docker_tool_chain_installed()`**

When creating docker images the tools `umoci` and `skopeo` are used in order to create 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\_grub\_efi\_installed\_for\_efi\_firmware()**

If the image is being built with efi or uefi firmware setting we need a grub(2)-...-efi package installed. The check is not 100% as every distribution has different names and different requirement but it is a reasonable approximation on the safe side meaning the user may still get an error but should not receive a false positive

**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\_only\_for\_x86\_arch()**

If the current architecture is not from the x86 family the 'mediacheck' feature available for iso images is not supported. Checkmedia tool and its related boot code are only available for x86 platforms.

**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\_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

## 6.2.17 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\_container\_compression()**

Return compression algorithm to use for compression of container images

**container:**

- `compress: xzlnone`

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\_iso\_tool\_category()**

Return tool category which should be used to build iso images

**iso:**

- `tool_category: cdrtools|xorriso`

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\_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

## 6.2.18 kiwi.version Module

Global version information used in kiwi and the package

## 6.2.19 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

## 6.2.20 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*) – instance of XMLDescription
- **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()**

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

**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\_pxedeploy\_section()**

First pxedeploy section from the build type section

**Returns** <pxedeploy> section reference

**Return type** xml\_parse::pxedeploy

**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\_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\_vmnice\_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\_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\_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\_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\_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\_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\_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\_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 chattr

**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 firmware and xen\_loader configuration values:

- 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

## 6.2.21 Module Contents

# 6.3 Schema Documentation 6.9

## 6.4 Extending KIWI with Custom Operations

---

### Hint: Abstract

Users building images with KIWI 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 today.

This document describes how to create an extension plugin for the KIWI schema to add and validate additional information in the KIWI image description.

Such a schema extension can be used in an additional KIWI task plugin to provide a new subcommand for KIWI. As of today there is no other plugin interface except for providing additional KIWI commands implemented.

Depending on the demand for custom plugins, the interface to hook in code into other parts of the KIWI processing needs to be extended.

This description applies for version 9.16.19.

---

### 6.4.1 The <extension> Section

The main KIWI 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

## 6.4.2 RELAX NG Schema for the Extension

If an extension section is found, KIWI 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
```

## 6.4.3 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"
```

(continues on next page)

(continued from previous page)

```
uri="file:///usr/share/xml/kiwi/my_plugin.rng"/>
</catalog>
```

For resolving the catalog KIWI 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 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
```

### 6.4.4 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'))
```

The core appliance builder is developed in Python and follows the test driven development rules.

## 6.5 Fork the upstream KIWI repository

1. On GitHub, navigate to: <https://github.com/SUSE/kiwi>
2. In the top-right corner of the page, click **Fork**.

## 6.6 Create a local clone of the forked KIWI repository

```
$ git clone https://github.com/YOUR-USERNAME/kiwi
$ git remote add upstream https://github.com/SUSE/kiwi.git
```

## 6.7 Create a python virtual development environment

The Python project uses **pyvenv** to setup a development environment for the desired Python version. The script **pyvenv** is already installed when using Python 3.3 and higher (see <https://docs.python.org/3.3/whatsnew/3.3.html#pep-405-virtual-environments> for details). For Python 2.7 use **virtualenv**, which is provided via pip or as an extra package in your favourite Linux distribution.

However, for setting up a Python virtual development environment the following additional LaTeX, include, header files and compilers are required in order to allow for compiling the C parts of the runtime required Python modules:

- XML processing with libxml2 and libxslt (for lxml)
- Foreign function interface library (libffi48)
- Python header files (for xattr)
- GCC compiler and glibc-devel header files
- LaTeX packages for building PDF documentation

---

**Note:** On SUSE systems the required components can be installed with the following command. Package names and install command are different on other systems.

---

```
$ zypper install \
  python3-devel libxml2-devel libxslt-devel libffi48-devel \
  glibc-devel gcc texlive-fncychap texlive-wrapfig \
  texlive-capt-of trang
```

Once the basic python module requirements are installed on your system, the next step is to create the virtual development environment. The following procedure describes how to create a Python3 virtual development environment:

1. Create the virtual environment:

```
$ python3 -m venv .env3
```

2. Activate the virtual environment:

```
$ source .env3/bin/activate
```

3. Install KIWI requirements inside the virtual environment:

```
$ pip install -r .virtualenv.dev-requirements.txt
```

4. Install KIWI in Development Mode:

```
$ ./setup.py develop
```

Once the development environment is activated and initialized with the project required Python modules, you are ready to work.

The **develop** target of the **setup.py** script automatically creates the application entry point called **kiwi-ng-3**, which allows to simply call the application from the code in the virtual environment:

```
$ kiwi-ng-3 --help
```

In order to leave the development mode just call:

```
$ deactivate
```

To resume your work, change into your local Git repository and recall:

```
$ source .env3/bin/activate
```

If the version has changed by **bumpversion**, this causes the current entry point to become invalid. Reconstruct the entry point after a version change by recalling:

```
$ ./setup.py develop
```

## 6.8 Running Test Cases

For running test cases, the preferred method is to use Tox. The Tox execution environment can be used to run any kind of target, tests are just one, documentation is another one. Refer to `tox.ini` for more details. Tox itself creates a python virtual environment for each tox target below `./.tox`.

Before you start to contribute code make sure all tests pass by calling the following command:

```
$ tox
```

The previous call would run **tox** for different Python versions, checks the source code for errors, and builds the documentation.

If you want to see the target, use the option `-l` to print a list:

```
$ tox -l
```

To only run a special target, use the `-e` option. The following example runs the test cases for the 3.6 interpreter only:

```
$ tox -e 3.6
```

## 6.9 Create a branch for each feature or bugfix

Congratulation ! you successfully created a KIWI development environment and all tests passed. Now it's time to hack on KIWI. Code changes should be done in an extra git branch. This allows for creating GitHub pull requests in a clean way. Also See [Github 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 effort.

---

---

**Note:** Your work is important and should 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 tests and code style checks. All of these are also performed by the travis integration test system at the time when a pull request will be created.

```
$ tox
```

Once all is done push your local branch to the forked repository and head out to GitHub for creating a pull request into the upstream repository.

```
$ git push origin my-topic-branch
```

Thanks much for contributing to KIWI. Your time and work effort is very much appreciated.

## 6.10 Good to know

The following sections provides further information about repository integrity, version, package and documentation management and are a good read to complete the picture of how the KIWI project works.

## 6.10.1 Signing Git Patches

With ssh keys being widely available and the increasing compute power available to many people refactoring of SSH keys is in the range of possibilities. Therefore SSH keys as used by GitHub as a “login/authentication” mechanism no longer provide the security they once did. In an effort to ensure the integrity of the repository and the code base patches sent for inclusion must be GPG signed.

To prepare Git to sign commits, follow these one-time instructions:

1. Create a key suitable for signing (its not recommended to use existing keys to not mix it up with your email environment etc):

```
$ gpg --gen-key
```

2. Choose a DSA key (3) with a key size of 2048 bits (default) and a validation of 3 years (3y). Enter your name/email and GPG will generate a DSA key for you.

You can also choose to use an empty passphrase, despite GPG’s warning, because you are only going to sign your public git commits with it and don’t need it for protecting any of your secrets. That might ease later use if you are not using an **gpg-agent** that caches your passphrase between multiple signed Git commits.

3. Add the key ID to your git config

In above case, the ID is 11223344 so you add it to either your global `~/.gitconfig` or even better to your `.git/config` inside your repo:

```
[user]
name = Joe Developer
email = developer@foo.bar
signingkey = 11223344
```

## 6.10.2 Raising Versions

The KIWI project follows the [Semantic Versioning](#) method. To make it easier to follow this method, **bumpversion** is used as a tool.

Follow these instructions to raise the major, minor, or patch part of a version:

- For backwards-compatible bug fixes:

```
$ bumpversion patch
```

- For additional functionality in a backwards-compatible manner. When changed, the patch level is set back to zero:

```
$ bumpversion minor
```

- For incompatible API changes. When changed, the patch and minor levels are set back to zero:



```
$ bumpversion major
```

### 6.10.3 Creating a Package

The creation of RPM package sources has to be done by calling the following make target:

```
$ make build
```

The sources are collected below the `dist/` directory. In there you will find all required files to submit a package to the Open Build Service or just build it with **rpmbuild**.

### 6.10.4 Building Documentation

The documentation is implemented using Sphinx with the ReST markup. In order to build the documentation just call:

```
tox -e doc
```

Whenever a change in the documentation is pushed to GitHub, it will be automatically updated via **travis-sphinx** and is available at:

<https://opensource.suse.com/kiwi>

## CHAPTER 7

---

### Appliance ?

---

An appliance is a ready to use image of an operating system including a preconfigured 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.

In KIWI the appliance is specified by a collection of human readable files in a directory, also called as the `image description`. 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.

## CHAPTER 8

---

### Contact

---

- [Mailing list](#)

The `kiwi-images` group is an open group and anyone can [subscribe](#), even if you do not have a Google account.

- IRC:

`#opensuse-kiwi` on `irc.freenode.net`

---

## Python Module Index

---

### k

- kiwi, 224
- kiwi.app, 174
- kiwi.archive, 78
- kiwi.archive.cpio, 76
- kiwi.archive.tar, 77
- kiwi.boot, 82
- kiwi.boot.image, 82
- kiwi.boot.image.base, 78
- kiwi.boot.image.builtin\_kiwi, 81
- kiwi.boot.image.dracut, 80
- kiwi.bootloader, 96
- kiwi.bootloader.config, 90
- kiwi.bootloader.config.base, 82
- kiwi.bootloader.config.grub2, 86
- kiwi.bootloader.config.isolinux, 88
- kiwi.bootloader.config.zipl, 89
- kiwi.bootloader.install, 93
- kiwi.bootloader.install.base, 91
- kiwi.bootloader.install.grub2, 91
- kiwi.bootloader.install.zipl, 92
- kiwi.bootloader.template, 96
- kiwi.bootloader.template.grub2, 93
- kiwi.bootloader.template.isolinux, 95
- kiwi.bootloader.template.zipl, 96
- kiwi.builder, 102
- kiwi.builder.archive, 96
- kiwi.builder.container, 97
- kiwi.builder.disk, 98
- kiwi.builder.filesystem, 99
- kiwi.builder.install, 100
- kiwi.builder.live, 101
- kiwi.builder.pxe, 101
- kiwi.cli, 175
- kiwi.command, 176
- kiwi.command\_process, 177
- kiwi.container, 105
- kiwi.container.docker, 104
- kiwi.container.oci, 104
- kiwi.container.setup, 104
- kiwi.container.setup.base, 102
- kiwi.container.setup.docker, 104
- kiwi.defaults, 178
- kiwi.exceptions, 189
- kiwi.filesystem, 111
- kiwi.filesystem.base, 106
- kiwi.filesystem.btrfs, 107
- kiwi.filesystem.clicfs, 107
- kiwi.filesystem.ext2, 108
- kiwi.filesystem.ext3, 108
- kiwi.filesystem.ext4, 108
- kiwi.filesystem.fat16, 109
- kiwi.filesystem.fat32, 109
- kiwi.filesystem.isofs, 109
- kiwi.filesystem.setup, 110
- kiwi.filesystem.squashfs, 110
- kiwi.filesystem.xfs, 111
- kiwi.firmware, 199
- kiwi.help, 201
- kiwi.iso\_tools, 115
- kiwi.iso\_tools.base, 111
- kiwi.iso\_tools.cdrtools, 112
- kiwi.iso\_tools.iso, 113
- kiwi.kiwi, 201
- kiwi.logger, 202
- kiwi.mount\_manager, 204

---

- kiwi.package\_manager, 122
- kiwi.package\_manager.base, 115
- kiwi.package\_manager.yum, 117
- kiwi.package\_manager.zypper, 119
- kiwi.partitioner, 125
- kiwi.partitioner.base, 122
- kiwi.partitioner.dasd, 123
- kiwi.partitioner.gpt, 124
- kiwi.partitioner.msdos, 125
- kiwi.path, 205
- kiwi.privileges, 206
- kiwi.repository, 132
- kiwi.repository.base, 126
- kiwi.repository.template, 126
- kiwi.repository.template.apt, 126
- kiwi.repository.yum, 128
- kiwi.repository.zypper, 130
- kiwi.runtime\_checker, 207
- kiwi.runtime\_config, 209
- kiwi.solver, 163
- kiwi.solver.repository, 162
- kiwi.solver.repository.base, 160
- kiwi.solver.repository.rpm\_dir, 161
- kiwi.solver.repository.rpm\_md, 161
- kiwi.solver.repository.suse, 161
- kiwi.solver.sat, 162
- kiwi.storage, 146
- kiwi.storage.device\_provider, 139
- kiwi.storage.disk, 140
- kiwi.storage.loop\_device, 142
- kiwi.storage.luks\_device, 143
- kiwi.storage.mapped\_device, 144
- kiwi.storage.raid\_device, 144
- kiwi.storage.setup, 145
- kiwi.storage.subformat, 139
- kiwi.storage.subformat.base, 133
- kiwi.storage.subformat.gce, 135
- kiwi.storage.subformat.ova, 135
- kiwi.storage.subformat.qcow2, 136
- kiwi.storage.subformat.template, 133
- kiwi.storage.subformat.template.vmware\_settings, 132
- kiwi.storage.subformat.vagrant\_libvirt, 136
- kiwi.storage.subformat.vdi, 137
- kiwi.storage.subformat.vhd, 137
- kiwi.storage.subformat.vhdfixed, 138
- kiwi.storage.subformat.vhdx, 138
- kiwi.storage.subformat.vmdk, 139
- kiwi.system, 160
- kiwi.system.identifier, 146
- kiwi.system.kernel, 147
- kiwi.system.prepare, 148
- kiwi.system.profile, 150
- kiwi.system.result, 150
- kiwi.system.root\_bind, 152
- kiwi.system.root\_init, 153
- kiwi.system.setup, 153
- kiwi.system.shell, 156
- kiwi.system.size, 157
- kiwi.system.uri, 158
- kiwi.system.users, 159
- kiwi.tasks, 165
- kiwi.tasks.base, 163
- kiwi.tasks.result\_bundle, 164
- kiwi.tasks.result\_list, 164
- kiwi.tasks.system\_build, 164
- kiwi.tasks.system\_create, 165
- kiwi.tasks.system\_prepare, 165
- kiwi.tasks.system\_update, 165
- kiwi.utils, 169
- kiwi.utils.block, 166
- kiwi.utils.checksum, 166
- kiwi.utils.compress, 167
- kiwi.utils.sync, 168
- kiwi.utils.sysconfig, 168
- kiwi.version, 210
- kiwi.volume\_manager, 174
- kiwi.volume\_manager.base, 169
- kiwi.volume\_manager.btrfs, 171
- kiwi.volume\_manager.lvm, 173
- kiwi.xml\_description, 210
- kiwi.xml\_state, 211

## A

- `accumulate_files()`  
(*kiwi.system.size.SystemSize* method), 157
  - `accumulate_mbyte_file_sizes()`  
(*kiwi.system.size.SystemSize* method), 157
  - `activate_boot_partition()`  
(*kiwi.storage.disk.Disk* method), 140
  - `add()` (*kiwi.system.profile.Profile* method), 150
  - `add()` (*kiwi.system.result.Result* method), 150
  - `add_container_config_label()`  
(*kiwi.xml\_state.XMLState* method), 211
  - `add_efi_loader_parameters()`  
(*kiwi.iso\_tools.base.IsoToolsBase* method), 111
  - `add_efi_loader_parameters()`  
(*kiwi.iso\_tools.cdrtools.IsoToolsCdrTools* method), 112
  - `add_repo()`  
(*kiwi.repository.base.RepositoryBase* method), 127
  - `add_repo()`  
(*kiwi.repository.yum.RepositoryYum* method), 128
  - `add_repo()`  
(*kiwi.repository.zypper.RepositoryZypper* method), 130
  - `add_repository()` (*kiwi.solver.sat.Sat* method), 162
  - `add_repository()`  
(*kiwi.xml\_state.XMLState* method), 211
  - `alias()` (*kiwi.system.uri.Uri* method), 158
  - `App` (class in *kiwi.app*), 174
  - `append_files()`  
(*kiwi.archive.tar.ArchiveTar* method), 77
  - `append_unpartitioned_space()`  
(*kiwi.builder.disk.DiskBuilder* method), 98
  - `apply_attributes_on_volume()`  
(*kiwi.volume\_manager.base.VolumeManagerBase* method), 169
  - `ArchiveBuilder` (class in *kiwi.builder.archive*), 96
  - `ArchiveCpio` (class in *kiwi.archive.cpio*), 76
  - `ArchiveTar` (class in *kiwi.archive.tar*), 77
- ## B
- `bind_mount()`  
(*kiwi.mount\_manager.MountManager* method), 205
  - `bios_mode()` (*kiwi.firmware.FirmWare* method), 199
  - `BlockID` (class in *kiwi.utils.block*), 166
  - `boot_partition_size()`  
(*kiwi.storage.setup.DiskSetup* method), 145
  - `BootImage` (class in *kiwi.boot.image*), 82
  - `BootImageBase` (class in *kiwi.boot.image.base*), 78
  - `BootImageDracut` (class in *kiwi.boot.image.dracut*), 80
  - `BootImageKiwi` (class in *kiwi.boot.image.builtin\_kiwi*), 81

BootLoaderConfig	(class in <i>kiwi.bootloader.config</i> ), 90	( <i>kiwi.runtime_checker.RuntimeChecker</i> method), 207
BootLoaderConfigBase	(class in <i>kiwi.bootloader.config.base</i> ), 82	<i>check_docker_tool_chain_installed()</i> ( <i>kiwi.runtime_checker.RuntimeChecker</i> method), 207
BootLoaderConfigGrub2	(class in <i>kiwi.bootloader.config.grub2</i> ), 86	<i>check_dracut_module_for_disk_oem_in_package()</i> ( <i>kiwi.runtime_checker.RuntimeChecker</i> method), 207
BootLoaderConfigIsoLinux	(class in <i>kiwi.bootloader.config.isolinux</i> ), 88	<i>check_dracut_module_for_disk_overlay_in_package()</i> ( <i>kiwi.runtime_checker.RuntimeChecker</i> method), 207
BootLoaderConfigZipl	(class in <i>kiwi.bootloader.config.zipl</i> ), 89	<i>check_dracut_module_for_live_iso_in_package()</i> ( <i>kiwi.runtime_checker.RuntimeChecker</i> method), 207
BootLoaderInstall	(class in <i>kiwi.bootloader.install</i> ), 93	<i>check_dracut_module_for_oem_install_in_package()</i> ( <i>kiwi.runtime_checker.RuntimeChecker</i> method), 207
BootLoaderInstallBase	(class in <i>kiwi.bootloader.install.base</i> ), 91	<i>check_efi_mode_for_disk_overlay_correctly()</i> ( <i>kiwi.runtime_checker.RuntimeChecker</i> method), 207
BootLoaderInstallGrub2	(class in <i>kiwi.bootloader.install.grub2</i> ), 91	<i>check_for_root_permissions()</i> ( <i>kiwi.privileges.Privileges</i> class method), 206
BootLoaderInstallZipl	(class in <i>kiwi.bootloader.install.zipl</i> ), 92	<i>check_grub_efi_installed_for_efi_firmware()</i> ( <i>kiwi.runtime_checker.RuntimeChecker</i> method), 208
BootLoaderTemplateGrub2	(class in <i>kiwi.bootloader.template.grub2</i> ), 93	<i>check_image_include_repos_publicly_resolvable()</i> ( <i>kiwi.runtime_checker.RuntimeChecker</i> method), 208
BootLoaderTemplateIsoLinux	(class in <i>kiwi.bootloader.template.isolinux</i> ), 95	<i>check_mediacheck_only_for_x86_arch()</i> ( <i>kiwi.runtime_checker.RuntimeChecker</i> method), 208
BootLoaderTemplateZipl	(class in <i>kiwi.bootloader.template.zipl</i> ), 96	<i>check_repositories_configured()</i> ( <i>kiwi.runtime_checker.RuntimeChecker</i> method), 208
<b>C</b>		<i>check_target_directory_not_in_shared_cache()</i> ( <i>kiwi.runtime_checker.RuntimeChecker</i> method), 208
calculate_id()	( <i>kiwi.system.identifier.SystemIdentifier</i> method), 146	<i>check_volume_label_used_with_lvm()</i> ( <i>kiwi.runtime_checker.RuntimeChecker</i> method), 208
call()	( <i>kiwi.command.Command</i> class method), 176	<i>check_volume_setup_has_no_root_definition()</i> ( <i>kiwi.runtime_checker.RuntimeChecker</i> method), 208
call_config_script()	( <i>kiwi.system.setup.SystemSetup</i> method), 154	<i>check_xen_uniquely_setup_as_server_or_guest()</i> ( <i>kiwi.runtime_checker.RuntimeChecker</i> method), 208
call_edit_boot_config_script()	( <i>kiwi.system.setup.SystemSetup</i> method), 154	
call_edit_boot_install_script()	( <i>kiwi.system.setup.SystemSetup</i> method), 154	
call_image_script()	( <i>kiwi.system.setup.SystemSetup</i> method), 154	
check_boot_description_exists()	( <i>kiwi.runtime_checker.RuntimeChecker</i> method), 207	
check_consistent_kernel_in_boot_image	( <i>kiwi.system.checksum</i> class method), 207	

166	( <i>kiwi.xml_state.XMLState</i> method),
<code>cleanup()</code> ( <i>kiwi.system.root_bind.RootBind</i>	212
method), 152	<code>copy_bootincluded_packages()</code>
<code>cleanup()</code> ( <i>kiwi.system.setup.SystemSetup</i>	( <i>kiwi.xml_state.XMLState</i> method),
method), 154	212
<code>cleanup_requests()</code>	<code>copy_build_type_attributes()</code>
( <i>kiwi.package_manager.base.PackageManagerBase</i>	( <i>kiwi.xml_state.XMLState</i> method),
method), 115	212
<code>cleanup_unused_repos()</code>	<code>copy_displayname()</code>
( <i>kiwi.repository.base.RepositoryBase</i>	( <i>kiwi.xml_state.XMLState</i> method),
method), 127	212
<code>cleanup_unused_repos()</code>	<code>copy_drivers_sections()</code>
( <i>kiwi.repository.yum.RepositoryYum</i>	( <i>kiwi.xml_state.XMLState</i> method),
method), 129	212
<code>cleanup_unused_repos()</code>	<code>copy_kernel()</code>
( <i>kiwi.repository.zypper.RepositoryZypper</i>	( <i>kiwi.system.kernel.Kernel</i> method),
method), 131	147
<code>Cli</code> (class in <i>kiwi.cli</i> ), 175	<code>copy_machine_section()</code>
<code>CliTask</code> (class in <i>kiwi.tasks.base</i> ), 163	( <i>kiwi.xml_state.XMLState</i> method),
<code>ColorFormatter</code> (class in <i>kiwi.logger</i> ),	213
202	<code>copy_name()</code> ( <i>kiwi.xml_state.XMLState</i>
<code>ColorMessage</code> (class in <i>kiwi.logger</i> ), 202	method), 213
<code>Command</code> (class in <i>kiwi.command</i> ), 176	<code>copy_oemconfig_section()</code>
<code>CommandIterator</code> (class in	( <i>kiwi.xml_state.XMLState</i> method),
<i>kiwi.command_process</i> ), 177	213
<code>CommandProcess</code> (class in	<code>copy_preferences_subsections()</code>
<i>kiwi.command_process</i> ), 178	( <i>kiwi.xml_state.XMLState</i> method),
<code>Compress</code> (class in <i>kiwi.utils.compress</i> ), 167	213
<code>compress</code> ( <i>kiwi.system.result.result_file_type</i>	<code>copy_repository_sections()</code>
attribute), 151	( <i>kiwi.xml_state.XMLState</i> method),
<code>ContainerBuilder</code> (class in	213
<i>kiwi.builder.container</i> ), 97	<code>copy_strip_sections()</code>
<code>ContainerImage</code> (class in <i>kiwi.container</i> ),	( <i>kiwi.xml_state.XMLState</i> method),
105	213
<code>ContainerImageDocker</code> (class in	<code>copy_systemdisk_section()</code>
<i>kiwi.container.docker</i> ), 104	( <i>kiwi.xml_state.XMLState</i> method),
<code>ContainerImageOCI</code> (class in	213
<i>kiwi.container.oci</i> ), 104	<code>copy_xen_hypervisor()</code>
<code>ContainerSetup</code> (class in	( <i>kiwi.system.kernel.Kernel</i> method),
<i>kiwi.container.setup</i> ), 104	147
<code>ContainerSetupBase</code> (class in	<code>create()</code> ( <i>kiwi.archive.cpio.ArchiveCpio</i>
<i>kiwi.container.setup.base</i> ), 102	method), 77
<code>ContainerSetupDocker</code> (class in	<code>create()</code> ( <i>kiwi.archive.tar.ArchiveTar</i>
<i>kiwi.container.setup.docker</i> ), 104	method), 77
<code>copy_bootdelete_packages()</code>	<code>create()</code> ( <i>kiwi.builder.archive.ArchiveBuilder</i>
( <i>kiwi.xml_state.XMLState</i> method),	method), 97
212	<code>create()</code> ( <i>kiwi.builder.container.ContainerBuilder</i>
<code>copy_bootincluded_archives()</code>	method), 97



```

create() (kiwi.builder.disk.DiskBuilder method), 98
create_efi_path()
create() (kiwi.builder.filesystem.FileSystemBuilder (kiwi.bootloader.config.base.BootLoaderConfigBase method), 99
create() (kiwi.builder.live.LiveImageBuilder create_fstab() (kiwi.container.setup.base.ContainerSetupBase method), 101
create() (kiwi.builder.pxe.PxeBuilder method), 103
create_fstab()
create() (kiwi.container.oci.ContainerImageOCI (kiwi.system.setup.SystemSetup method), 105
create() (kiwi.partitioner.base.PartitionerBase create_gnu_gzip_compressed() (kiwi.archive.tar.ArchiveTar method), 122
create() (kiwi.partitioner.dasd.PartitionerDasd 78
create() (kiwi.partitioner.gpt.PartitionerGpt (kiwi.iso_tools.iso.Iso method), 124
create() (kiwi.partitioner.msdos.PartitionerMsdos create_header_end_block() (kiwi.iso_tools.iso.Iso method), 125
create() (kiwi.path.Path class method), create_hybrid() (kiwi.iso_tools.iso.Iso class method), 114
create_hybrid_mbr()
create() (kiwi.storage.loop_device.LoopDevice create_hybrid_mbr() (kiwi.storage.disk.Disk method), 143
create() (kiwi.system.profile.Profile 141
create_image_format()
create() (kiwi.system.root_init.RootInit (kiwi.storage.subformat.base.DiskFormatBase method), 153
create_boot_partition() create_image_format() (kiwi.storage.subformat.gce.DiskFormatGce method), 135
create_crypto_luks() create_image_format() (kiwi.storage.subformat.ova.DiskFormatOva method), 135
create_crypttab() create_image_format() (kiwi.storage.subformat.qcow2.DiskFormatQcow2 method), 136
create_degraded_raid() create_image_format() (kiwi.storage.subformat.vagrant_libvirt.DiskFormatVagrant method), 136
create_disk() create_image_format() (kiwi.storage.subformat.vdi.DiskFormatVdi method), 137
create_disk_format() create_image_format() (kiwi.storage.subformat.vhd.DiskFormatVhd method), 137
create_efi_csm_partition() create_image_format() (kiwi.storage.subformat.vhdfixed.DiskFormatVhdFixed method), 138
create_efi_partition() create_image_format() (kiwi.storage.disk.Disk method),

```

(*kiwi.storage.subformat.vhdx.DiskFormatVhdx* method), 108  
method), 138 create\_on\_device()  
create\_image\_format() (*kiwi.filesystem.ext4.FileSystemExt4*  
(*kiwi.storage.subformat.vmdk.DiskFormatVmdk* method), 108  
method), 139 create\_on\_device()  
create\_init\_link\_from\_linuxrc() (*kiwi.filesystem.fat16.FileSystemFat16*  
(*kiwi.system.setup.SystemSetup* method), 109  
method), 155 create\_on\_device()  
create\_initrd() (*kiwi.filesystem.fat32.FileSystemFat32*  
(*kiwi.boot.image.base.BootImageBase* method), 109  
method), 79 create\_on\_device()  
create\_initrd() (*kiwi.filesystem.xfs.FileSystemXfs*  
(*kiwi.boot.image.builtin\_kiwi.BootImageKiwi* method), 111  
method), 81 create\_on\_file()  
create\_initrd() (*kiwi.filesystem.base.FileSystemBase*  
(*kiwi.boot.image.dracut.BootImageDracut* method), 106  
method), 80 create\_on\_file()  
create\_install\_iso() (*kiwi.filesystem.clicfs.FileSystemClicFs*  
(*kiwi.builder.install.InstallImageBuilder* method), 107  
method), 100 create\_on\_file()  
create\_install\_media() (*kiwi.filesystem.iso9660.FileSystemIso9660*  
(*kiwi.builder.disk.DiskBuilder* method), 109  
method), 99 create\_on\_file()  
create\_install\_pxe\_archive() (*kiwi.filesystem.squashfs.FileSystemSquashFs*  
(*kiwi.builder.install.InstallImageBuilder* method), 110  
method), 100 create\_prep\_partition()  
create\_iso() (*kiwi.storage.disk.Disk* method),  
(*kiwi.iso\_tools.base.IsoToolsBase* method), 112 141  
create\_iso() create\_raid\_config()  
(*kiwi.iso\_tools.cdrtools.IsoToolsCdrTools* method), 145  
(*kiwi.iso\_tools.cdrtools.IsoToolsCdrTools* method), 113 create\_recovery\_archive()  
create\_match\_method() (*kiwi.system.setup.SystemSetup*  
(*kiwi.command\_process.CommandProcess* method), 155  
method), 178 create\_repository\_solvable()  
create\_mbr() (*kiwi.storage.disk.Disk* method), 141  
(*kiwi.storage.disk.Disk* method), 160  
create\_on\_device() create\_root\_lvm\_partition()  
(*kiwi.filesystem.base.FileSystemBase* method), 106  
(*kiwi.storage.disk.Disk* method), 141  
create\_on\_device() create\_root\_partition()  
(*kiwi.filesystem.btrfs.FileSystemBtrfs* method), 107  
(*kiwi.storage.disk.Disk* method), 141  
create\_on\_device() create\_root\_raid\_partition()  
(*kiwi.filesystem.ext2.FileSystemExt2* method), 108  
(*kiwi.storage.disk.Disk* method), 141  
create\_on\_device() create\_root\_readonly\_partition()  
(*kiwi.filesystem.ext3.FileSystemExt3* method), 108  
(*kiwi.storage.disk.Disk* method), 141  
create\_spare\_partition()  
(*kiwi.storage.disk.Disk* method), 141

142  
 create\_volume\_paths\_in\_root\_dir() (kiwi.volume\_manager.base.VolumeManagerBase method), 169  
 create\_volumes() (kiwi.volume\_manager.base.VolumeManagerBase method), 169  
 create\_volumes() (kiwi.volume\_manager.btrfs.VolumeManagerBtrfs method), 171  
 create\_volumes() (kiwi.volume\_manager.lvm.VolumeManagerLvm method), 173  
 create\_xz\_compressed() (kiwi.archive.tar.ArchiveTar method), 78  
 credentials\_file\_name() (kiwi.system.uri.Uri method), 158  
 customize() (kiwi.system.size.SystemSize method), 157

**D**

database\_consistent() (kiwi.package\_manager.base.PackageManagerBase method), 115  
 database\_consistent() (kiwi.package\_manager.yum.PackageManagerYum method), 117  
 database\_consistent() (kiwi.package\_manager.zypper.PackageManagerZypper method), 120  
 DataSync (class in kiwi.utils.sync), 168  
 deactivate\_bootloader\_setup() (kiwi.container.setup.base.ContainerSetupBase method), 103  
 deactivate\_root\_filesystem\_check() (kiwi.container.setup.base.ContainerSetupBase method), 103  
 deactivate\_systemd\_service() (kiwi.container.setup.base.ContainerSetupBase method), 103  
 DebugFilter (class in kiwi.logger), 202  
 Defaults (class in kiwi.defaults), 178  
 delete() (kiwi.system.profile.Profile method), 150  
 delete() (kiwi.system.root\_init.RootInit method), 153  
 delete\_all\_repos() (kiwi.repository.base.RepositoryBase method), 127  
 delete\_all\_repos() (kiwi.repository.yum.RepositoryYum method), 129  
 delete\_all\_repos() (kiwi.repository.zypper.RepositoryZypper method), 131  
 delete\_btrfs\_packages() (kiwi.system.prepare.SystemPrepare method), 148  
 delete\_lvm\_repo() (kiwi.repository.base.RepositoryBase method), 127  
 delete\_repo() (kiwi.repository.yum.RepositoryYum method), 129  
 delete\_repo() (kiwi.repository.zypper.RepositoryZypper method), 131  
 delete\_repo\_cache() (kiwi.repository.base.RepositoryBase method), 127  
 delete\_repo\_cache() (kiwi.repository.yum.RepositoryYum method), 129  
 delete\_repo\_cache() (kiwi.repository.zypper.RepositoryZypper method), 131  
 delete\_repository\_sections() (kiwi.xml\_state.XMLState method), 213  
 delete\_repository\_sections\_used\_for\_build() (kiwi.xml\_state.XMLState method), 213  
 DeviceProvider (class in kiwi.storage.device\_provider), 139  
 disable\_cleanup() (kiwi.boot.image.base.BootImageBase method), 79  
 Disk (class in kiwi.storage.disk), 140  
 DiskBuilder (class in kiwi.builder.disk), 98  
 DiskFormat (class in kiwi.storage.subformat), 139  
 DiskFormatBase (class in kiwi.storage.subformat.base), 133  
 DiskFormatGce (class in

<i>kiwi.storage.subformat.gce</i> ), 135	<i>export_modprobe_setup()</i>
DiskFormatOva (class in <i>kiwi.storage.subformat.ova</i> ), 135	( <i>kiwi.system.setup.SystemSetup</i> method), 155
DiskFormatQcow2 (class in <i>kiwi.storage.subformat.qcow2</i> ), 136	<i>export_package_list()</i> ( <i>kiwi.system.setup.SystemSetup</i> method), 155
DiskFormatVagrantLibVirt (class in <i>kiwi.storage.subformat.vagrant_libvirt</i> ), 136	<i>export_package_verification()</i> ( <i>kiwi.system.setup.SystemSetup</i> method), 155
DiskFormatVdi (class in <i>kiwi.storage.subformat.vdi</i> ), 137	<i>extract()</i> ( <i>kiwi.archive.cpio.ArchiveCpio</i> method), 77
DiskFormatVhd (class in <i>kiwi.storage.subformat.vhd</i> ), 137	<i>extract()</i> ( <i>kiwi.archive.tar.ArchiveTar</i> method), 78
DiskFormatVhdFixed (class in <i>kiwi.storage.subformat.vhdfixed</i> ), 138	<i>extras()</i> (in module <i>kiwi.kiwi</i> ), 201
<b>F</b>	
DiskFormatVhdx (class in <i>kiwi.storage.subformat.vhdx</i> ), 138	<i>failsafe_boot_entry_requested()</i> ( <i>kiwi.bootloader.config.base.BootLoaderConfigBase</i> method), 82
DiskFormatVmdk (class in <i>kiwi.storage.subformat.vmdk</i> ), 139	<i>filename()</i> ( <i>kiwi.system.result.result_file_type</i> attribute), 151
DiskSetup (class in <i>kiwi.storage.setup</i> ), 145	<i>FileSystem</i> (class in <i>kiwi.filesystem</i> ), 111
<i>download_from_repository()</i> ( <i>kiwi.solver.repository.base.SolverRepositoryBase</i> method), 161	<i>FileSystemBase</i> (class in <i>kiwi.filesystem.base</i> ), 106
<i>dump()</i> ( <i>kiwi.boot.image.base.BootImageBase</i> method), 79	<i>FileSystemBtrfs</i> (class in <i>kiwi.filesystem.btrfs</i> ), 107
<i>dump()</i> ( <i>kiwi.system.result.Result</i> method), 151	<i>FileSystemBuilder</i> (class in <i>kiwi.builder.filesystem</i> ), 99
<i>dump_reload_package_database()</i> ( <i>kiwi.package_manager.base.PackageManagerBase</i> method), 115	<i>FileSystemClicFs</i> (class in <i>kiwi.filesystem.clicfs</i> ), 107
<i>dump_reload_package_database()</i> ( <i>kiwi.package_manager.yum.PackageManagerYum</i> method), 118	<i>FileSystemExt2</i> (class in <i>kiwi.filesystem.ext2</i> ), 108
<i>dump_reload_package_database()</i> ( <i>kiwi.package_manager.zypper.PackageManagerZypper</i> method), 120	<i>FileSystemExt3</i> (class in <i>kiwi.filesystem.ext3</i> ), 108
<b>E</b>	
<i>ec2_mode()</i> ( <i>kiwi.firmware.FirmWare</i> method), 199	<i>FileSystemExt4</i> (class in <i>kiwi.filesystem.ext4</i> ), 108
<i>efi_mode()</i> ( <i>kiwi.firmware.FirmWare</i> method), 200	<i>FileSytemFat16</i> (class in <i>kiwi.filesystem.fat16</i> ), 109
<i>enable_cleanup()</i> ( <i>kiwi.boot.image.base.BootImageBase</i> method), 79	<i>FileSystemFat32</i> (class in <i>kiwi.filesystem.fat32</i> ), 109
<i>ErrorFilter</i> (class in <i>kiwi.logger</i> ), 203	<i>FileSystemIsoFs</i> (class in <i>kiwi.filesystem.isoFs</i> ), 109
	<i>FileSystemSetup</i> (class in <i>kiwi.filesystem.setup</i> ), 110
	<i>FileSystemSquashFs</i> (class in <i>kiwi.filesystem.squashfs</i> ), 110
	<i>FileSystemXfs</i> (class in <i>kiwi.filesystem.xfs</i> ), 111

[filter\(\)](#) (*kiwi.logger.DebugFilter method*), 202  
[filter\(\)](#) (*kiwi.logger.ErrorFilter method*), 203  
[filter\(\)](#) (*kiwi.logger.InfoFilter method*), 203  
[filter\(\)](#) (*kiwi.logger.LoggerSchedulerFilter method*), 204  
[filter\(\)](#) (*kiwi.logger.WarningFilter method*), 204  
[FirmWare](#) (class in *kiwi.firmware*), 199  
[fix\\_boot\\_catalog\(\)](#) (*kiwi.iso\_tools.iso.Iso class method*), 114  
[format\(\)](#) (*kiwi.logger.ColorFormatter method*), 202  
[format\\_message\(\)](#) (*kiwi.logger.ColorMessage method*), 202  
**G**  
[get\(\)](#) (*kiwi.defaults.Defaults method*), 178  
[get\\_archive\\_image\\_types\(\)](#) (*kiwi.defaults.Defaults class method*), 179  
[get\\_blkid\(\)](#) (*kiwi.utils.block.BlockID method*), 166  
[get\\_boot\\_cmdline\(\)](#) (*kiwi.bootloader.config.base.BootLoaderConfigBase method*), 83  
[get\\_boot\\_description\\_directory\(\)](#) (*kiwi.boot.image.base.BootImageBase method*), 79  
[get\\_boot\\_image\\_description\\_path\(\)](#) (*kiwi.defaults.Defaults class method*), 179  
[get\\_boot\\_image\\_strip\\_file\(\)](#) (*kiwi.defaults.Defaults class method*), 179  
[get\\_boot\\_label\(\)](#) (*kiwi.storage.setup.DiskSetup method*), 145  
[get\\_boot\\_names\(\)](#) (*kiwi.boot.image.base.BootImageBase method*), 79  
[get\\_boot\\_names\(\)](#) (*kiwi.boot.image.builtin\_kiwi.BootImageKiwi method*), 81  
[get\\_boot\\_names\(\)](#) (*kiwi.boot.image.dracut.BootImageDracut method*), 80  
[get\\_boot\\_path\(\)](#) (*kiwi.bootloader.config.base.BootLoaderConfigBase method*), 83  
[get\\_boot\\_theme\(\)](#) (*kiwi.bootloader.config.base.BootLoaderConfigBase method*), 83  
[get\\_boot\\_timeout\\_seconds\(\)](#) (*kiwi.bootloader.config.base.BootLoaderConfigBase method*), 83  
[get\\_bootstrap\\_archives\(\)](#) (*kiwi.xml\_state.XMLState method*), 213  
[get\\_bootstrap\\_collection\\_type\(\)](#) (*kiwi.xml\_state.XMLState method*), 214  
[get\\_bootstrap\\_collections\(\)](#) (*kiwi.xml\_state.XMLState method*), 214  
[get\\_bootstrap\\_packages\(\)](#) (*kiwi.xml\_state.XMLState method*), 214  
[get\\_bootstrap\\_packages\\_sections\(\)](#) (*kiwi.xml\_state.XMLState method*), 214  
[get\\_bootstrap\\_products\(\)](#) (*kiwi.xml\_state.XMLState method*), 214  
[get\\_build\\_type\\_containerconfig\\_section\(\)](#) (*kiwi.xml\_state.XMLState method*), 214  
[get\\_build\\_type\\_format\\_options\(\)](#) (*kiwi.xml\_state.XMLState method*), 214  
[get\\_build\\_type\\_machine\\_section\(\)](#) (*kiwi.xml\_state.XMLState method*), 214  
[get\\_build\\_type\\_name\(\)](#) (*kiwi.xml\_state.XMLState method*), 214  
[get\\_build\\_type\\_oemconfig\\_section\(\)](#) (*kiwi.xml\_state.XMLState method*), 215  
[get\\_build\\_type\\_pxedeploy\\_section\(\)](#) (*kiwi.xml\_state.XMLState method*), 215



<code>get_build_type_size()</code> ( <i>kiwi.xml_state.XMLState</i> method), 215	<code>get_container_compression()</code> ( <i>kiwi.defaults.Defaults</i> class method), 179
<code>get_build_type_spare_part_size()</code> ( <i>kiwi.xml_state.XMLState</i> method), 215	<code>get_container_compression()</code> ( <i>kiwi.runtime_config.RuntimeConfig</i> method), 209
<code>get_build_type_system_disk_section()</code> ( <i>kiwi.xml_state.XMLState</i> method), 215	<code>get_container_config()</code> ( <i>kiwi.xml_state.XMLState</i> method), 216
<code>get_build_type_unpartitioned_bytes()</code> ( <i>kiwi.xml_state.XMLState</i> method), 215	<code>get_container_image_types()</code> ( <i>kiwi.defaults.Defaults</i> class method), 179
<code>get_build_type_vagrant_config_section()</code> ( <i>kiwi.xml_state.XMLState</i> method), 215	<code>get_container_name()</code> ( <i>kiwi.container.setup.base.ContainerSetupBase</i> method), 103
<code>get_build_type_vmconfig_entries()</code> ( <i>kiwi.xml_state.XMLState</i> method), 215	<code>get_default_boot_mbytes()</code> ( <i>kiwi.defaults.Defaults</i> class method), 179
<code>get_build_type_vmdisk_section()</code> ( <i>kiwi.xml_state.XMLState</i> method), 216	<code>get_default_boot_timeout_seconds()</code> ( <i>kiwi.defaults.Defaults</i> class method), 180
<code>get_build_type_vmdvd_section()</code> ( <i>kiwi.xml_state.XMLState</i> method), 216	<code>get_default_container_name()</code> ( <i>kiwi.defaults.Defaults</i> class method), 180
<code>get_build_type_vmnics_entries()</code> ( <i>kiwi.xml_state.XMLState</i> method), 216	<code>get_default_container_tag()</code> ( <i>kiwi.defaults.Defaults</i> class method), 180
<code>get_buildservice_env_name()</code> ( <i>kiwi.defaults.Defaults</i> class method), 179	<code>get_default_disk_start_sector()</code> ( <i>kiwi.defaults.Defaults</i> class method), 180
<code>get_byte_size()</code> ( <i>kiwi.storage.device_provider.DeviceProvider</i> method), 139	<code>get_default_efi_boot_mbytes()</code> ( <i>kiwi.defaults.Defaults</i> class method), 180
<code>get_canonical_volume_list()</code> ( <i>kiwi.volume_manager.base.VolumeManagerBase</i> method), 170	<code>get_default_efi_partition_table_type()</code> ( <i>kiwi.defaults.Defaults</i> class method), 180
<code>get_collection_type()</code> ( <i>kiwi.xml_state.XMLState</i> method), 216	<code>get_default_firmware()</code> ( <i>kiwi.defaults.Defaults</i> class method), 180
<code>get_collections()</code> ( <i>kiwi.xml_state.XMLState</i> method), 216	<code>get_default_inode_size()</code> ( <i>kiwi.defaults.Defaults</i> class method), 180
<code>get_command()</code> ( <i>kiwi.cli.Cli</i> method), 175	<code>get_default_legacy_bios_mbytes()</code> ( <i>kiwi.defaults.Defaults</i> class method), 181
<code>get_command_args()</code> ( <i>kiwi.cli.Cli</i> method), 175	
<code>get_common_functions_file()</code> ( <i>kiwi.defaults.Defaults</i> class method), 179	<code>get_default_live_iso_root_filesystem()</code> ( <i>kiwi.defaults.Defaults</i> class method), 181

<code>get_default_live_iso_type()</code> ( <i>kiwi.defaults.Defaults class method</i> ), 181	( <i>kiwi.xml_state.XMLState method</i> ), 217
<code>get_default_packager_tool()</code> ( <i>kiwi.defaults.Defaults class method</i> ), 181	<code>get_disk_template()</code> ( <i>kiwi.bootloader.template.grub2.BootLoaderTemplate method</i> ), 93
<code>get_default_prep_mbytes()</code> ( <i>kiwi.defaults.Defaults class method</i> ), 181	<code>get_disksize_mbytes()</code> ( <i>kiwi.storage.setup.DiskSetup method</i> ), 145
<code>get_default_video_mode()</code> ( <i>kiwi.defaults.Defaults class method</i> ), 181	<code>get_distribution_name_from_boot_attribute()</code> ( <i>kiwi.xml_state.XMLState method</i> ), 217
<code>get_default_volume_group_name()</code> ( <i>kiwi.defaults.Defaults class method</i> ), 181	<code>get_dracut_conf_name()</code> ( <i>kiwi.defaults.Defaults class method</i> ), 182
<code>get_derived_from_image_uri()</code> ( <i>kiwi.xml_state.XMLState method</i> ), 216	<code>get_drivers_list()</code> ( <i>kiwi.xml_state.XMLState method</i> ), 217
<code>get_device()</code> ( <i>kiwi.storage.device_provider.DeviceProvider method</i> ), 140	<code>get_ec2_capable_firmware_names()</code> ( <i>kiwi.defaults.Defaults class method</i> ), 182
<code>get_device()</code> ( <i>kiwi.storage.disk.Disk method</i> ), 142	<code>get_efi_capable_firmware_names()</code> ( <i>kiwi.defaults.Defaults class method</i> ), 182
<code>get_device()</code> ( <i>kiwi.storage.loop_device.LoopDevice method</i> ), 143	<code>get_efi_image_name()</code> ( <i>kiwi.defaults.Defaults class method</i> ), 182
<code>get_device()</code> ( <i>kiwi.storage.luks_device.LuksDevice method</i> ), 143	<code>get_efi_label()</code> ( <i>kiwi.storage.setup.DiskSetup method</i> ), 146
<code>get_device()</code> ( <i>kiwi.storage.mapped_device.MappedDevice method</i> ), 144	<code>get_efi_module_directory_name()</code> ( <i>kiwi.defaults.Defaults class method</i> ), 182
<code>get_device()</code> ( <i>kiwi.storage.raid_device.RaidDevice method</i> ), 145	<code>get_efi_partition_size()</code> ( <i>kiwi.firmware.FirmWare method</i> ), 200
<code>get_device()</code> ( <i>kiwi.volume_manager.base.VolumeManagerBase method</i> ), 170	<code>get_error_code()</code> ( <i>kiwi.command_process.CommandIterator method</i> ), 177
<code>get_device()</code> ( <i>kiwi.volume_manager.lvm.VolumeManagerLVM method</i> ), 173	<code>get_error_output()</code> ( <i>kiwi.command_process.CommandIterator method</i> ), 177
<code>get_disk_format_types()</code> ( <i>kiwi.defaults.Defaults class method</i> ), 181	<code>get_exclude_list_for_root_data_sync()</code> ( <i>kiwi.defaults.Defaults class method</i> ), 182
<code>get_disk_image_types()</code> ( <i>kiwi.defaults.Defaults class method</i> ), 182	<code>get_extension_xml_data()</code> ( <i>kiwi.xml_description.XMLDescription method</i> ), 211
<code>get_disk_start_sector()</code>	<code>get_failsafe_kernel_options()</code>

(*kiwi.defaults.Defaults class method*), 183

`get_filesystem()` (*kiwi.utils.block.BlockID method*), 166

`get_filesystem_image_types()` (*kiwi.defaults.Defaults class method*), 183

`get_firmware_types()` (*kiwi.defaults.Defaults class method*), 183

`get_format()` (*kiwi.utils.compress.Compress method*), 167

`get_fragment()` (*kiwi.system.uri.Uri method*), 158

`get_fs_mount_option_list()` (*kiwi.xml\_state.XMLState method*), 217

`get_fstab()` (*kiwi.volume\_manager.base.VolumeManagerBase method*), 170

`get_fstab()` (*kiwi.volume\_manager.btrfs.VolumeManagerBtrfs method*), 172

`get_fstab()` (*kiwi.volume\_manager.lvm.VolumeManagerLVM method*), 173

`get_gfxmode()` (*kiwi.bootloader.config.base.BootLoaderConfigBase method*), 83

`get_global_args()` (*kiwi.cli.Cli method*), 175

`get_grub_basic_modules()` (*kiwi.defaults.Defaults class method*), 183

`get_grub_bios_modules()` (*kiwi.defaults.Defaults class method*), 183

`get_grub_boot_directory_name()` (*kiwi.defaults.Defaults class method*), 183

`get_grub_efi_modules()` (*kiwi.defaults.Defaults class method*), 183

`get_grub_ofw_modules()` (*kiwi.defaults.Defaults class method*), 184

`get_grub_path()` (*kiwi.defaults.Defaults class method*), 184

`get_host_template()` (*kiwi.repository.template.apt.PackageManagerTemplate method*), 126

`get_id()` (*kiwi.partitionner.base.PartitionerBase method*), 123

`get_id()` (*kiwi.system.identifier.SystemIdentifier method*), 146

`get_image_packages_sections()` (*kiwi.xml\_state.XMLState method*), 217

`get_image_template()` (*kiwi.repository.template.apt.PackageManagerTemplate method*), 126

`get_image_version()` (*kiwi.xml\_state.XMLState method*), 217

`get_imported_root_image()` (*kiwi.defaults.Defaults class method*), 184

`get_initrd_system()` (*kiwi.xml\_state.XMLState method*), 217

`get_install_image_boot_default()` (*kiwi.bootloader.config.base.BootLoaderConfigBase method*), 83

`get_install_message_template()` (*kiwi.bootloader.template.isolinux.BootLoaderTemplate method*), 95

`get_install_template()` (*kiwi.bootloader.template.grub2.BootLoaderTemplate method*), 93

`get_install_template()` (*kiwi.bootloader.template.isolinux.BootLoaderTemplate method*), 95

`get_install_volume_id()` (*kiwi.defaults.Defaults class method*), 184

`get_iso_boot_path()` (*kiwi.defaults.Defaults class method*), 184

`get_iso_template()` (*kiwi.bootloader.template.grub2.BootLoaderTemplate method*), 94

`get_iso_tool_category()` (*kiwi.defaults.Defaults class method*), 184



<code>get_iso_tool_category()</code> ( <i>kiwi.runtime_config.RuntimeConfig</i> <i>method</i> ), 209	<code>method</code> ), 94
<code>get_kernel()</code> ( <i>kiwi.system.kernel.Kernel</i> <i>method</i> ), 147	<code>get_multiboot_template()</code> ( <i>kiwi.bootloader.template.isolinux.BootLoaderTemplate</i> <i>method</i> ), 95
<code>get_label()</code> ( <i>kiwi.utils.block.BlockID</i> <i>method</i> ), 166	<code>get_network_image_types()</code> ( <i>kiwi.defaults.Defaults class method</i> ), 185
<code>get_legacy_bios_partition_size()</code> ( <i>kiwi.firmware.FirmWare</i> <i>method</i> ), 200	<code>get_obs_download_server_url()</code> ( <i>kiwi.defaults.Defaults class method</i> ), 185
<code>get_live_dracut_module_from_flags()</code> ( <i>kiwi.defaults.Defaults class method</i> ), 184	<code>get_obs_download_server_url()</code> ( <i>kiwi.runtime_config.RuntimeConfig</i> <i>method</i> ), 209
<code>get_live_image_types()</code> ( <i>kiwi.defaults.Defaults class method</i> ), 185	<code>get_oemconfig_oem_multipath_scan()</code> ( <i>kiwi.xml_state.XMLState</i> <i>method</i> ), 218
<code>get_live_iso_persistent_boot_options()</code> ( <i>kiwi.defaults.Defaults class method</i> ), 185	<code>get_package_manager()</code> ( <i>kiwi.xml_state.XMLState</i> <i>method</i> ), 218
<code>get_lvm_overhead_mbytes()</code> ( <i>kiwi.defaults.Defaults class method</i> ), 185	<code>get_package_sections()</code> ( <i>kiwi.xml_state.XMLState</i> <i>method</i> ), 218
<code>get_max_size_constraint()</code> ( <i>kiwi.runtime_config.RuntimeConfig</i> <i>method</i> ), 209	<code>get_packages_sections()</code> ( <i>kiwi.xml_state.XMLState</i> <i>method</i> ), 218
<code>get_menu_entry_install_title()</code> ( <i>kiwi.bootloader.config.base.BootLoaderConfigBase</i> <i>method</i> ), 84	<code>get_partition_table_type()</code> ( <i>kiwi.firmware.FirmWare</i> <i>method</i> ), 200
<code>get_menu_entry_title()</code> ( <i>kiwi.bootloader.config.base.BootLoaderConfigBase</i> <i>method</i> ), 84	<code>get_pid()</code> ( <i>kiwi.command_process.CommandIterator</i> <i>method</i> ), 177
<code>get_message_template()</code> ( <i>kiwi.bootloader.template.isolinux.BootLoaderTemplateIsoLinux</i> <i>method</i> ), 95	<code>get_preferences_sections()</code> ( <i>kiwi.xml_state.XMLState</i> <i>method</i> ), 218
<code>get_min_volume_mbytes()</code> ( <i>kiwi.defaults.Defaults class method</i> ), 185	<code>get_prep_partition_size()</code> ( <i>kiwi.firmware.FirmWare</i> <i>method</i> ), 200
<code>get_multiboot_disk_template()</code> ( <i>kiwi.bootloader.template.grub2.BootLoaderTemplateGrub2</i> <i>method</i> ), 94	<code>get_preparer()</code> ( <i>kiwi.defaults.Defaults</i> <i>class method</i> ), 185
<code>get_multiboot_install_template()</code> ( <i>kiwi.bootloader.template.grub2.BootLoaderTemplateGrub2</i> <i>method</i> ), 94	<code>get_release()</code> ( <i>kiwi.xml_state.XMLState</i> <i>method</i> ), 218
<code>get_multiboot_install_template()</code> ( <i>kiwi.bootloader.template.isolinux.BootLoaderTemplateIsoLinux</i> <i>method</i> ), 95	<code>get_partition_id_map()</code> ( <i>kiwi.storage.disk.Disk</i> <i>method</i> ), 142
<code>get_multiboot_iso_template()</code> ( <i>kiwi.bootloader.template.grub2.BootLoaderTemplateGrub2</i> <i>method</i> ), 94	<code>get_publisher()</code> ( <i>kiwi.defaults.Defaults</i> <i>class method</i> ), 185
	<code>get_qemu_option_list()</code> ( <i>kiwi.storage.subformat.base.DiskFormatBase</i> <i>method</i> ), 214

`get_recovery_spare_mbytes()`  
(*kiwi.defaults.Defaults* class method), 186

`get_repository_sections()`  
(*kiwi.xml\_state.XMLState* method), 219

`get_repository_sections_used_for_build()`  
(*kiwi.xml\_state.XMLState* method), 219

`get_repository_sections_used_in_image()`  
(*kiwi.xml\_state.XMLState* method), 219

`get_results()` (*kiwi.system.result.Result* method), 151

`get_root_label()`  
(*kiwi.storage.setup.DiskSetup* method), 146

`get_rpm_check_signatures()`  
(*kiwi.xml\_state.XMLState* method), 219

`get_rpm_excludedocs()`  
(*kiwi.xml\_state.XMLState* method), 219

`get_s390_disk_block_size()`  
(*kiwi.defaults.Defaults* class method), 186

`get_s390_disk_type()`  
(*kiwi.defaults.Defaults* class method), 186

`get_schema_file()`  
(*kiwi.defaults.Defaults* class method), 186

`get_servicename()` (*kiwi.cli.Cli* method), 175

`get_shared_cache_location()`  
(*kiwi.defaults.Defaults* class method), 186

`get_shim_loader()`  
(*kiwi.defaults.Defaults* class method), 186

`get_shim_vendor_directory()`  
(*kiwi.defaults.Defaults* class method), 186

`get_signed_grub_loader()`  
(*kiwi.defaults.Defaults* class method), 187

`get_size_mbytes()`  
(*kiwi.filesystem.setup.FileSystemSetup* method), 110

`get_snapper_config_template_file()`  
(*kiwi.defaults.Defaults* class method), 187

`get_solvable_location()`  
(*kiwi.defaults.Defaults* class method), 187

`get_strip_files_to_delete()`  
(*kiwi.xml\_state.XMLState* method), 219

`get_strip_libraries_to_keep()`  
(*kiwi.xml\_state.XMLState* method), 219

`get_strip_list()`  
(*kiwi.xml\_state.XMLState* method), 220

`get_strip_tools_to_keep()`  
(*kiwi.xml\_state.XMLState* method), 220

`get_system_archives()`  
(*kiwi.xml\_state.XMLState* method), 220

`get_system_collection_type()`  
(*kiwi.xml\_state.XMLState* method), 220

`get_system_collections()`  
(*kiwi.xml\_state.XMLState* method), 220

`get_system_ignore_packages()`  
(*kiwi.xml\_state.XMLState* method), 220

`get_system_packages()`  
(*kiwi.xml\_state.XMLState* method), 220

`get_system_products()`  
(*kiwi.xml\_state.XMLState* method), 220

`get_target_file_path_for_format()`  
(*kiwi.storage.subformat.base.DiskFormatBase* method), 134

`get_target_file_path_for_format()`  
(*kiwi.storage.subformat.gce.DiskFormatGce* method), 135

`get_template()`  
(*kiwi.bootloader.template.isolinux.BootLoaderTemplate* method), 96

`get_template()`  
(*kiwi.bootloader.template.zipl.BootLoaderTemplate* method), 96

*method*), 96  
 get\_template() (kiwi.storage.subformat.template.vmware\_settings.Template method), 132  
 get\_to\_become\_deleted\_packages() (kiwi.xml\_state.XMLState method), 221  
 get\_tool\_name() (kiwi.iso\_tools.base.IsoToolsBase method), 112  
 get\_tool\_name() (kiwi.iso\_tools.cdrtools.IsoToolsCdrTools method), 113  
 get\_unsigned\_grub\_loader() (kiwi.defaults.Defaults class method), 187  
 get\_user\_groups() (kiwi.xml\_state.XMLState method), 221  
 get\_users() (kiwi.xml\_state.XMLState method), 221  
 get\_users\_sections() (kiwi.xml\_state.XMLState method), 221  
 get\_uuid() (kiwi.storage.device\_provider.DeviceProvider method), 140  
 get\_uuid() (kiwi.utils.block.BlockID method), 166  
 get\_video\_mode\_map() (kiwi.defaults.Defaults class method), 187  
 get\_volume\_group\_name() (kiwi.xml\_state.XMLState method), 221  
 get\_volume\_id() (kiwi.defaults.Defaults class method), 187  
 get\_volume\_management() (kiwi.xml\_state.XMLState method), 221  
 get\_volume\_mbsize() (kiwi.volume\_manager.base.VolumeManagerBase method), 170  
 get\_volumes() (kiwi.volume\_manager.base.VolumeManagerBase method), 170  
 get\_volumes() (kiwi.volume\_manager.btrfs.VolumeManagerBtrfs method), 172  
 get\_volumes() (kiwi.volume\_manager.lvm.VolumeManagerLVM method), 173  
 get\_volumes() (kiwi.xml\_state.XMLState method), 221  
 get\_xen\_hypervisor() (kiwi.system.kernel.Kernel method), 148  
 get\_xsl\_stylesheet\_file() (kiwi.defaults.Defaults class method), 188  
 get\_xz\_compression\_options() (kiwi.defaults.Defaults class method), 188  
 get\_xz\_options() (kiwi.runtime\_config.RuntimeConfig method), 210  
 getLogLevel() (kiwi.logger.Logger method), 203  
 group\_add() (kiwi.system.users.Users method), 159  
 group\_exists() (kiwi.system.users.Users method), 159  
 gzip() (kiwi.utils.compress.Compress method), 167

## H

has\_iso\_hybrid\_capability() (kiwi.iso\_tools.base.IsoToolsBase method), 112  
 has\_iso\_hybrid\_capability() (kiwi.iso\_tools.cdrtools.IsoToolsCdrTools method), 113  
 has\_raw\_disk() (kiwi.storage.subformat.base.DiskFormatBase method), 134  
 Help (class in kiwi.help), 201

## I

ImageBuilder (class in kiwi.builder), 102  
 import\_cdroot\_files() (kiwi.system.setup.SystemSetup method), 155  
 import\_description() (kiwi.system.setup.SystemSetup method), 155  
 image\_identifier()

```

    (kiwi.system.setup.SystemSetup
    method), 155
import_overlay_files()
    (kiwi.system.setup.SystemSetup
    method), 155
import_repositories_marked_as_imageinstallable()
    (kiwi.system.setup.SystemSetup
    method), 155
import_shell_environment()
    (kiwi.system.setup.SystemSetup
    method), 155
import_system_description_elements()
    (kiwi.boot.image.base.BootImageBase
    method), 79
import_trusted_keys()
    (kiwi.repository.base.RepositoryBase
    method), 127
import_trusted_keys()
    (kiwi.repository.yum.RepositoryYum
    method), 129
import_trusted_keys()
    (kiwi.repository.zypper.RepositoryZypper
    method), 131
include_file()
    (kiwi.boot.image.base.BootImageBase
    method), 79
include_file()
    (kiwi.boot.image.dracut.BootImageDracut
    method), 80
InfoFilter (class in kiwi.logger), 203
init_iso_creation_parameters()
    (kiwi.iso_tools.base.IsoToolsBase
    method), 112
init_iso_creation_parameters()
    (kiwi.iso_tools.cdrtools.IsoToolsCdrTools
    method), 113
install() (kiwi.bootloader.install.base.BootLoaderInstallBase
    method), 91
install() (kiwi.bootloader.install.grub2.BootLoaderInstallGrub2
    method), 91
install() (kiwi.bootloader.install.zipl.BootLoaderInstallZipl
    method), 92
install_bootstrap()
    (kiwi.system.prepare.SystemPrepare
    method), 148
install_packages()
    (kiwi.system.prepare.SystemPrepare
    method), 148
install_required()
    (kiwi.bootloader.install.base.BootLoaderInstallBase
    method), 91
install_required()
    (kiwi.bootloader.install.grub2.BootLoaderInstallGrub2
    method), 91
install_required()
    (kiwi.bootloader.install.zipl.BootLoaderInstallZipl
    method), 92
install_system()
    (kiwi.system.prepare.SystemPrepare
    method), 149
InstallImageBuilder (class in
    kiwi.builder.install), 100
invoke_kiwicompat() (kiwi.cli.Cli
    method), 175
is_buildservice_worker()
    (kiwi.defaults.Defaults class method),
    188
is_loop() (kiwi.storage.device_provider.DeviceProvider
    method), 140
is_loop() (kiwi.storage.disk.Disk
    method), 142
is_loop() (kiwi.storage.loop_device.LoopDevice
    method), 143
is_loop() (kiwi.storage.luks_device.LuksDevice
    method), 144
is_loop() (kiwi.storage.mapped_device.MappedDevice
    method), 144
is_loop() (kiwi.storage.raid_device.RaidDevice
    method), 145
is_loop() (kiwi.volume_manager.base.VolumeManagerBase
    method), 170
is_mounted()
    (kiwi.mount_manager.MountManager
    method), 205
is_public() (kiwi.runtime_config.RuntimeConfig
    method), 212
is_prepared()
    (kiwi.boot.image.base.BootImageBase
    method), 79
is_public() (kiwi.system.uri.Uri
    method), 158
is_remote() (kiwi.system.uri.Uri
    method), 158
is_uptodate()
    (kiwi.solver.repository.base.SolverRepositoryBase
    method), 158

```

- method*), 161
- `is_xen_guest()`
  - (*kiwi.xml\_state.XMLState* *method*), 222
- `is_xen_server()`
  - (*kiwi.xml\_state.XMLState* *method*), 222
- Iso* (*class in kiwi.iso\_tools.iso*), 113
- IsoTools* (*class in kiwi.iso\_tools*), 115
- IsoToolsBase* (*class in kiwi.iso\_tools.base*), 111
- IsoToolsCdrTools* (*class in kiwi.iso\_tools.cdrtools*), 112
- K**
- Kernel* (*class in kiwi.system.kernel*), 147
- `kill()` (*kiwi.command\_process.CommandIterator* *method*), 178
- kiwi* (*module*), 224
- kiwi.app* (*module*), 174
- kiwi.archive* (*module*), 78
- kiwi.archive.cpio* (*module*), 76
- kiwi.archive.tar* (*module*), 77
- kiwi.boot* (*module*), 82
- kiwi.boot.image* (*module*), 82
- kiwi.boot.image.base* (*module*), 78
- kiwi.boot.image.builtin\_kiwi* (*module*), 81
- kiwi.boot.image.dracut* (*module*), 80
- kiwi.bootloader* (*module*), 96
- kiwi.bootloader.config* (*module*), 90
- kiwi.bootloader.config.base* (*module*), 82
- kiwi.bootloader.config.grub2* (*module*), 86
- kiwi.bootloader.config.isolinux* (*module*), 88
- kiwi.bootloader.config.zipl* (*module*), 89
- kiwi.bootloader.install* (*module*), 93
- kiwi.bootloader.install.base* (*module*), 91
- kiwi.bootloader.install.grub2* (*module*), 91
- kiwi.bootloader.install.zipl* (*module*), 92
- kiwi.bootloader.template* (*module*), 96
- kiwi.bootloader.template.grub2* (*module*), 93
- kiwi.bootloader.template.isolinux* (*module*), 95
- kiwi.bootloader.template.zipl* (*module*), 96
- kiwi.builder* (*module*), 102
- kiwi.builder.archive* (*module*), 96
- kiwi.builder.container* (*module*), 97
- kiwi.builder.disk* (*module*), 98
- kiwi.builder.filesystem* (*module*), 99
- kiwi.builder.install* (*module*), 100
- kiwi.builder.live* (*module*), 101
- kiwi.builder.pxe* (*module*), 101
- kiwi.cli* (*module*), 175
- kiwi.command* (*module*), 176
- kiwi.command\_process* (*module*), 177
- kiwi.container* (*module*), 105
- kiwi.container.docker* (*module*), 104
- kiwi.container.oci* (*module*), 104
- kiwi.container.setup* (*module*), 104
- kiwi.container.setup.base* (*module*), 102
- kiwi.container.setup.docker* (*module*), 104
- kiwi.defaults* (*module*), 178
- kiwi.exceptions* (*module*), 189
- kiwi.filesystem* (*module*), 111
- kiwi.filesystem.base* (*module*), 106
- kiwi.filesystem.btrfs* (*module*), 107
- kiwi.filesystem.clicfs* (*module*), 107
- kiwi.filesystem.ext2* (*module*), 108
- kiwi.filesystem.ext3* (*module*), 108
- kiwi.filesystem.ext4* (*module*), 108
- kiwi.filesystem.fat16* (*module*), 109
- kiwi.filesystem.fat32* (*module*), 109
- kiwi.filesystem.isofs* (*module*), 109
- kiwi.filesystem.setup* (*module*), 110
- kiwi.filesystem.squashfs* (*module*), 110
- kiwi.filesystem.xfs* (*module*), 111
- kiwi.firmware* (*module*), 199
- kiwi.help* (*module*), 201
- kiwi.iso\_tools* (*module*), 115
- kiwi.iso\_tools.base* (*module*), 111



`kiwi.iso_tools.cdrtools` (*module*), 112

`kiwi.iso_tools.iso` (*module*), 113

`kiwi.kiwi` (*module*), 201

`kiwi.logger` (*module*), 202

`kiwi.mount_manager` (*module*), 204

`kiwi.package_manager` (*module*), 122

`kiwi.package_manager.base` (*module*), 115

`kiwi.package_manager.yum` (*module*), 117

`kiwi.package_manager.zypper` (*module*), 119

`kiwi.partitioner` (*module*), 125

`kiwi.partitioner.base` (*module*), 122

`kiwi.partitioner.dasd` (*module*), 123

`kiwi.partitioner.gpt` (*module*), 124

`kiwi.partitioner.msdos` (*module*), 125

`kiwi.path` (*module*), 205

`kiwi.privileges` (*module*), 206

`kiwi.repository` (*module*), 132

`kiwi.repository.base` (*module*), 126

`kiwi.repository.template` (*module*), 126

`kiwi.repository.template.apt` (*module*), 126

`kiwi.repository.yum` (*module*), 128

`kiwi.repository.zypper` (*module*), 130

`kiwi.runtime_checker` (*module*), 207

`kiwi.runtime_config` (*module*), 209

`kiwi.solver` (*module*), 163

`kiwi.solver.repository` (*module*), 162

`kiwi.solver.repository.base` (*module*), 160

`kiwi.solver.repository.rpm_dir` (*module*), 161

`kiwi.solver.repository.rpm_md` (*module*), 161

`kiwi.solver.repository.suse` (*module*), 161

`kiwi.solver.sat` (*module*), 162

`kiwi.storage` (*module*), 146

`kiwi.storage.device_provider` (*module*), 139

`kiwi.storage.disk` (*module*), 140

`kiwi.storage.loop_device` (*module*), 142

`kiwi.storage.luks_device` (*module*), 143

`kiwi.storage.mapped_device` (*module*), 144

`kiwi.storage.raid_device` (*module*), 144

`kiwi.storage.setup` (*module*), 145

`kiwi.storage.subformat` (*module*), 139

`kiwi.storage.subformat.base` (*module*), 133

`kiwi.storage.subformat.gce` (*module*), 135

`kiwi.storage.subformat.ova` (*module*), 135

`kiwi.storage.subformat.qcow2` (*module*), 136

`kiwi.storage.subformat.template` (*module*), 133

`kiwi.storage.subformat.template.vmware_se` (*module*), 132

`kiwi.storage.subformat.vagrant_libvirt` (*module*), 136

`kiwi.storage.subformat.vdi` (*module*), 137

`kiwi.storage.subformat.vhd` (*module*), 137

`kiwi.storage.subformat.vhdfixed` (*module*), 138

`kiwi.storage.subformat.vhdx` (*module*), 138

`kiwi.storage.subformat.vmdk` (*module*), 139

`kiwi.system` (*module*), 160

`kiwi.system.identifier` (*module*), 146

`kiwi.system.kernel` (*module*), 147

`kiwi.system.prepare` (*module*), 148

`kiwi.system.profile` (*module*), 150

`kiwi.system.result` (*module*), 150

`kiwi.system.root_bind` (*module*), 152

`kiwi.system.root_init` (*module*), 153

`kiwi.system.setup` (*module*), 153

`kiwi.system.shell` (*module*), 156

`kiwi.system.size` (*module*), 157

`kiwi.system.uri` (*module*), 158

- `kiwi.system.users (module)`, 159
- `kiwi.tasks (module)`, 165
- `kiwi.tasks.base (module)`, 163
- `kiwi.tasks.result_bundle (module)`, 164
- `kiwi.tasks.result_list (module)`, 164
- `kiwi.tasks.system_build (module)`, 164
- `kiwi.tasks.system_create (module)`, 165
- `kiwi.tasks.system_prepare (module)`, 165
- `kiwi.tasks.system_update (module)`, 165
- `kiwi.utils (module)`, 169
- `kiwi.utils.block (module)`, 166
- `kiwi.utils.checksum (module)`, 166
- `kiwi.utils.compress (module)`, 167
- `kiwi.utils.sync (module)`, 168
- `kiwi.utils.sysconfig (module)`, 168
- `kiwi.version (module)`, 210
- `kiwi.volume_manager (module)`, 174
- `kiwi.volume_manager.base (module)`, 169
- `kiwi.volume_manager.btrfs (module)`, 171
- `kiwi.volume_manager.lvm (module)`, 173
- `kiwi.xml_description (module)`, 210
- `kiwi.xml_state (module)`, 211
- `KiwiArchiveSetupError`, 189
- `KiwiArchiveTarError`, 189
- `KiwiBootImageDumpError`, 189
- `KiwiBootImageSetupError`, 189
- `KiwiBootLoaderConfigSetupError`, 189
- `KiwiBootLoaderGrubDataError`, 189
- `KiwiBootLoaderGrubFontError`, 189
- `KiwiBootLoaderGrubInstallError`, 189
- `KiwiBootLoaderGrubModulesError`, 189
- `KiwiBootLoaderGrubPlatformError`, 189
- `KiwiBootLoaderGrubSecureBootError`, 189
- `KiwiBootLoaderInstallSetupError`, 190
- `KiwiBootLoaderIsoLinuxPlatformError`, 190
- `KiwiBootLoaderTargetError`, 190
- `KiwiBootLoaderZiplInstallError`, 190
- `KiwiBootLoaderZiplPlatformError`, 190
- `KiwiBootLoaderZiplSetupError`, 190
- `KiwiBootStrapPhaseFailed`, 190
- `KiwiBundleError`, 190
- `KiwiCommandCapabilitiesError`, 190
- `KiwiCommandError`, 190
- `KiwiCommandNotFound`, 190
- `KiwiCommandNotLoaded`, 191
- `KiwiCompatError`, 191
- `KiwiCompressionFormatUnknown`, 191
- `KiwiConfigFileNotFound`, 191
- `KiwiContainerBuilderError`, 191
- `KiwiContainerImageSetupError`, 191
- `KiwiContainerSetupError`, 191
- `KiwiDataStructureError`, 191
- `KiwiDebootstrapError`, 191
- `KiwiDecodingError`, 191
- `KiwiDescriptionConflict`, 191
- `KiwiDescriptionInvalid`, 192
- `KiwiDeviceProviderError`, 192
- `KiwiDiskBootImageError`, 192
- `KiwiDiskFormatSetupError`, 192
- `KiwiDiskGeometryError`, 192
- `KiwiDistributionNameError`, 192
- `KiwiError`, 192
- `KiwiExtensionError`, 192
- `KiwiFileNotFound`, 192
- `KiwiFileSystemSetupError`, 193
- `KiwiFileSystemSyncError`, 193
- `KiwiFormatSetupError`, 193
- `KiwiHelpNoCommandGiven`, 193
- `KiwiImageResizeError`, 193
- `KiwiImportDescriptionError`, 193
- `KiwiInstallBootImageError`, 193
- `KiwiInstallMediaError`, 193
- `KiwiInstallPhaseFailed`, 193
- `KiwiIsoLoaderError`, 193

- KiwiIsoMetaDataError, [193](#)
  - KiwiIsoToolError, [194](#)
  - KiwiKernelLookupError, [194](#)
  - KiwiLiveBootImageError, [194](#)
  - KiwiLoadCommandUndefined, [194](#)
  - KiwiLogFileSetupFailed, [194](#)
  - KiwiLoopSetupError, [194](#)
  - KiwiLuksSetupError, [194](#)
  - KiwiMappedDeviceError, [194](#)
  - KiwiMountKernelFileSystemsError, [194](#)
  - KiwiMountSharedDirectoryError, [194](#)
  - KiwiNotImplementedError, [194](#)
  - KiwiPackageManagerSetupError, [195](#)
  - KiwiPackagesDeletePhaseFailed, [195](#)
  - KiwiPartitionerGptFlagError, [195](#)
  - KiwiPartitionerMsDosFlagError, [195](#)
  - KiwiPartitionerSetupError, [195](#)
  - KiwiPrivilegesError, [195](#)
  - KiwiProfileNotFound, [195](#)
  - KiwiPxeBootImageError, [195](#)
  - KiwiRaidSetupError, [195](#)
  - KiwiRepositorySetupError, [196](#)
  - KiwiRepoTypeUnknown, [195](#)
  - KiwiRequestedTypeError, [196](#)
  - KiwiRequestError, [196](#)
  - KiwiResizeRawDiskError, [196](#)
  - KiwiResultError, [196](#)
  - KiwiRootDirExists, [196](#)
  - KiwiRootImportError, [196](#)
  - KiwiRootInitCreationError, [196](#)
  - KiwiRpmDatabaseReloadError, [196](#)
  - KiwiRpmDirNotRemoteError, [197](#)
  - KiwiRuntimeConfigFormatError, [197](#)
  - KiwiRuntimeError, [197](#)
  - KiwiSatSolverJobError, [197](#)
  - KiwiSatSolverJobProblems, [197](#)
  - KiwiSatSolverPluginError, [197](#)
  - KiwiSchemaImportError, [197](#)
  - KiwiScriptFailed, [197](#)
  - KiwiSetupIntermediateConfigError, [197](#)
  - KiwiSizeError, [197](#)
  - KiwiSolverRepositorySetupError, [198](#)
  - KiwiSystemDeletePackagesFailed, [198](#)
  - KiwiSystemInstallPackagesFailed, [198](#)
  - KiwiSystemUpdateFailed, [198](#)
  - KiwiTargetDirectoryNotFound, [198](#)
  - KiwiTemplateError, [198](#)
  - KiwiTypeNotFound, [198](#)
  - KiwiUnknownServiceName, [198](#)
  - KiwiUriOpenError, [198](#)
  - KiwiUriStyleUnknown, [198](#)
  - KiwiUriTypeUnknown, [198](#)
  - KiwiValidationError, [199](#)
  - KiwiVhdTagError, [199](#)
  - KiwiVolumeGroupConflict, [199](#)
  - KiwiVolumeManagerSetupError, [199](#)
  - KiwiVolumeRootIDError, [199](#)
- L**
- `legacy_bios_mode()`  
(*kiwi.firmware.FirmWare* method), [200](#)
  - `list_iso()`  
(*kiwi.iso\_tools.base.IsoToolsBase* method), [112](#)
  - `list_iso()`  
(*kiwi.iso\_tools.cdrtools.IsoToolsCdrTools* method), [113](#)
  - `LiveImageBuilder` (class in *kiwi.builder.live*), [101](#)
  - `load()` (*kiwi.system.result.Result* class method), [151](#)
  - `load()` (*kiwi.xml\_description.XMLDescription* method), [211](#)
  - `load_boot_xml_description()`  
(*kiwi.boot.image.base.BootImageBase* method), [79](#)
  - `load_command()` (*kiwi.cli.Cli* method), [176](#)
  - `load_xml_description()`  
(*kiwi.tasks.base.CliTask* method), [163](#)
  - `Logger` (class in *kiwi.logger*), [203](#)
  - `LoggerSchedulerFilter` (class in *kiwi.logger*), [204](#)
  - `LoopDevice` (class in



*kiwi.storage.loop\_device*), 142  
 LuksDevice (class in *kiwi.storage.luks\_device*), 143

## M

*main()* (in module *kiwi.kiwi*), 201  
*map\_partitions()* (*kiwi.storage.disk.Disk* method), 142  
*MappedDevice* (class in *kiwi.storage.mapped\_device*), 144  
*match\_package\_deleted()* (*kiwi.package\_manager.base.PackageManagerBase* method), 116  
*match\_package\_deleted()* (*kiwi.package\_manager.yum.PackageManagerYum* method), 118  
*match\_package\_deleted()* (*kiwi.package\_manager.zypper.PackageManagerZypper* method), 120  
*match\_package\_installed()* (*kiwi.package\_manager.base.PackageManagerBase* method), 116  
*match\_package\_installed()* (*kiwi.package\_manager.yum.PackageManagerYum* method), 118  
*match\_package\_installed()* (*kiwi.package\_manager.zypper.PackageManagerZypper* method), 120  
*matches()* (*kiwi.utils.checksum.Checksum* method), 166  
*md5()* (*kiwi.utils.checksum.Checksum* method), 167  
*mount()* (*kiwi.mount\_manager.MountManager* method), 205  
*mount\_kernel\_file\_systems()* (*kiwi.system.root\_bind.RootBind* method), 152  
*mount\_shared\_directory()* (*kiwi.system.root\_bind.RootBind* method), 152  
*mount\_volumes()* (*kiwi.volume\_manager.base.VolumeManagerBase* method), 170  
*mount\_volumes()* (*kiwi.volume\_manager.btrfs.VolumeManagerBtrfs* method), 172  
*mount\_volumes()* (*kiwi.volume\_manager.lvm.VolumeManagerLVM* method), 173  
*MountManager* (class in *kiwi.mount\_manager*), 204  
*move\_to\_root()* (*kiwi.system.root\_bind.RootBind* method), 152

## N

*need\_boot\_partition()* (*kiwi.storage.setup.DiskSetup* method), 146

## O

*ofw\_mode()* (*kiwi.firmware.FirmWare* method), 200  
*opal\_mode()* (*kiwi.firmware.FirmWare* method), 200

## P

*pack\_image\_to\_file()* (*kiwi.container.docker.ContainerImageDocker* method), 104  
*pack\_image\_to\_file()* (*kiwi.container.oci.ContainerImageOCI* method), 105  
*package\_matches\_host\_architecture()* (*kiwi.xml\_state.XMLState* method), 222  
*PackageManager* (class in *kiwi.package\_manager*), 122  
*PackageManagerBase* (class in *kiwi.package\_manager.base*), 115  
*PackageManagerTemplateAptGet* (class in *kiwi.repository.template.appt*), 126  
*PackageManagerYum* (class in *kiwi.package\_manager.yum*), 117  
*PackageManagerZypper* (class in *kiwi.package\_manager.zypper*), 119  
*Partitioner* (class in *kiwi.partitioner*), 125  
*PartitionerBase* (class in *kiwi.partitioner.base*), 122  
*PartitionerDasd* (class in *kiwi.partitioner.dasd*), 123  
*PartitionerGpt* (class in *kiwi.partitioner.gpt*), 124

PartitionerMsDos	(class in	(kiwi.filesystem.base.FileSystemBase
	kiwi.partitioners.msdos), 125	method), 106
Path	(class in kiwi.path), 205	post_init()
pinch_system()		(kiwi.filesystem.clicfs.FileSystemClicFs
	(kiwi.system.prepare.SystemPrepare	method), 108
	method), 149	post_init()
poll()	(kiwi.command_process.CommandProcess	(kiwi.package_manager.base.PackageManagerBase
	method), 178	method), 116
poll_and_watch()		post_init()
	(kiwi.command_process.CommandProcess	(kiwi.package_manager.yum.PackageManagerYum
	method), 178	method), 118
poll_show_progress()		post_init()
	(kiwi.command_process.CommandProcess	(kiwi.package_manager.zypper.PackageManagerZypper
	method), 178	method), 120
post_init()		post_init()
	(kiwi.boot.image.base.BootImageBase	(kiwi.partitioners.base.PartitionerBase
	method), 80	method), 123
post_init()		post_init()
	(kiwi.boot.image.builtin_kiwi.BootImageKiwi	(kiwi.partitioners.dasd.PartitionerDasd
	method), 81	method), 124
post_init()		post_init()
	(kiwi.boot.image.dracut.BootImageDracut	(kiwi.partitioners.gpt.PartitionerGpt
	method), 80	method), 124
post_init()		post_init()
	(kiwi.bootloader.config.base.BootLoaderConfigBase	(kiwi.partitioners.msdos.PartitionerMsDos
	method), 84	method), 125
post_init()		post_init()
	(kiwi.bootloader.config.grub2.BootLoaderConfigGrub2	(kiwi.repository.base.RepositoryBase
	method), 86	method), 128
post_init()		post_init()
	(kiwi.bootloader.config.isolinux.BootLoaderConfigIsolinux	(kiwi.repository.yum.RepositoryYum
	method), 88	method), 129
post_init()		post_init()
	(kiwi.bootloader.config.zipl.BootLoaderConfigZipl	(kiwi.repository.zypper.RepositoryZypper
	method), 89	method), 131
post_init()		post_init()
	(kiwi.bootloader.install.base.BootLoaderInstallBase	(kiwi.storage.subformat.base.DiskFormatBase
	method), 91	method), 134
post_init()		post_init()
	(kiwi.bootloader.install.grub2.BootLoaderInstallGrub2	(kiwi.storage.subformat.gce.DiskFormatGce
	method), 92	method), 135
post_init()		post_init()
	(kiwi.bootloader.install.zipl.BootLoaderInstallZipl	(kiwi.storage.subformat.ova.DiskFormatOva
	method), 92	method), 135
post_init()		post_init()
	(kiwi.container.setup.base.ContainerSetupBase	(kiwi.storage.subformat.qcow2.DiskFormatQcow2
	method), 103	method), 136
post_init()		post_init()

(*kiwi.storage.subformat.vagrant\_libvirt.DiskFormatVagrantLibvirt* requests()  
*method*), 136 (*kiwi.package\_manager.base.PackageManagerBase*  
*method*), 116  
post\_init() (*kiwi.storage.subformat.vdi.DiskFormatVdi* process\_delete\_requests()  
*method*), 137 (*kiwi.package\_manager.yum.PackageManagerYum*  
*method*), 118  
post\_init() (*kiwi.storage.subformat.vhd.DiskFormatVhd* process\_delete\_requests()  
*method*), 137 (*kiwi.package\_manager.zypper.PackageManagerZypper*  
*method*), 121  
post\_init() (*kiwi.storage.subformat.vhdfixed.DiskFormatVhdFixed* install\_requests()  
*method*), 138 (*kiwi.package\_manager.base.PackageManagerBase*  
*method*), 116  
post\_init() (*kiwi.storage.subformat.vhdx.DiskFormatVhdx* process\_install\_requests()  
*method*), 138 (*kiwi.package\_manager.yum.PackageManagerYum*  
*method*), 119  
post\_init() (*kiwi.storage.subformat.vmdk.DiskFormatVmdk* process\_install\_requests()  
*method*), 139 (*kiwi.package\_manager.zypper.PackageManagerZypper*  
*method*), 121  
post\_init() (*kiwi.volume\_manager.base.VolumeManagerBase* process\_install\_requests\_bootstrap()  
*method*), 171 (*kiwi.package\_manager.base.PackageManagerBase*  
*method*), 116  
post\_init() (*kiwi.volume\_manager.btrfs.VolumeManagerBtrfs* process\_install\_requests\_bootstrap()  
*method*), 172 (*kiwi.package\_manager.yum.PackageManagerYum*  
*method*), 119  
post\_init() (*kiwi.volume\_manager.lvm.VolumeManagerLvm* process\_install\_requests\_bootstrap()  
*method*), 173 (*kiwi.package\_manager.zypper.PackageManagerZypper*  
*method*), 121  
prepare() (*kiwi.boot.image.base.BootImageBase* *method*), 80 process\_only\_required()  
prepare() (*kiwi.boot.image.builtin\_kiwi.BootImageKiwi* *method*), 81 (*kiwi.package\_manager.base.PackageManagerBase*  
*method*), 116  
prepare() (*kiwi.boot.image.dracut.BootImageDracut* process\_only\_required()  
*method*), 81 (*kiwi.package\_manager.yum.PackageManagerYum*  
*method*), 119  
print\_results() (*kiwi.system.result.Result* *method*), process\_only\_required()  
151 (*kiwi.package\_manager.zypper.PackageManagerZypper*  
*method*), 121  
Privileges (class in *kiwi.privileges*), 206  
process() (*kiwi.tasks.result\_bundle.ResultBundleTask* process\_plus\_recommended()  
*method*), 164 (*kiwi.package\_manager.base.PackageManagerBase*  
*method*), 116  
process() (*kiwi.tasks.result\_list.ResultListTask* *method*), 164 process\_plus\_recommended()  
(*kiwi.package\_manager.yum.PackageManagerYum*  
*method*), 119  
process() (*kiwi.tasks.system\_build.SystemBuildTask* process\_plus\_recommended()  
*method*), 165 (*kiwi.package\_manager.zypper.PackageManagerZypper*  
*method*), 121  
process() (*kiwi.tasks.system\_prepare.SystemPrepareTask* *method*), 165 Profile (class in *kiwi.system.profile*), 150  
process() (*kiwi.tasks.system\_update.SystemUpdateTask* update\_matches\_host\_architecture()  
*method*), 165 (*kiwi.xml\_state.XMLState* *method*),

223  
progress() (*kiwi.logger.Logger* method),  
203  
project\_file() (*kiwi.defaults.Defaults*  
class method), 188  
PxeBuilder (class in *kiwi.builder.pxe*), 101

## Q

quadruple\_token()  
(*kiwi.tasks.base.CliTask* method),  
163  
quote() (*kiwi.system.shell.Shell* class  
method), 156  
quote\_key\_value\_file()  
(*kiwi.system.shell.Shell* class  
method), 156  
quote\_title()  
(*kiwi.bootloader.config.base.BootLoaderConfigBase*  
method), 84

## R

RaidDevice (class in  
*kiwi.storage.raid\_device*), 144  
relocate\_boot\_catalog()  
(*kiwi.iso\_tools.iso.Iso* class method),  
114  
remove() (*kiwi.path.Path* class method),  
205  
remove\_hierarchy() (*kiwi.path.Path*  
class method), 205  
Repository (class in *kiwi.repository*), 132  
RepositoryBase (class in  
*kiwi.repository.base*), 126  
RepositoryYum (class in  
*kiwi.repository.yum*), 128  
RepositoryYumSpaceRemover (class  
in *kiwi.repository.yum*), 129  
RepositoryZypper (class in  
*kiwi.repository.zypper*), 130  
request\_collection()  
(*kiwi.package\_manager.base.PackageManagerBase*  
method), 116  
request\_collection()  
(*kiwi.package\_manager.yum.PackageManagerYum*  
method), 119  
request\_collection()  
(*kiwi.package\_manager.zypper.PackageManagerZypper*  
method), 121

request\_package()  
(*kiwi.package\_manager.base.PackageManagerBase*  
method), 117  
request\_package()  
(*kiwi.package\_manager.yum.PackageManagerYum*  
method), 119  
request\_package()  
(*kiwi.package\_manager.zypper.PackageManagerZypper*  
method), 121  
request\_package\_exclusion()  
(*kiwi.package\_manager.base.PackageManagerBase*  
method), 117  
request\_package\_exclusion()  
(*kiwi.package\_manager.yum.PackageManagerYum*  
method), 119  
request\_package\_exclusion()  
(*kiwi.package\_manager.zypper.PackageManagerZypper*  
method), 121  
request\_package\_lock()  
(*kiwi.package\_manager.base.PackageManagerBase*  
method), 117  
request\_product()  
(*kiwi.package\_manager.base.PackageManagerBase*  
method), 117  
request\_product()  
(*kiwi.package\_manager.yum.PackageManagerYum*  
method), 119  
request\_product()  
(*kiwi.package\_manager.zypper.PackageManagerZypper*  
method), 121  
resize\_raw\_disk()  
(*kiwi.storage.subformat.base.DiskFormatBase*  
method), 134  
resize\_table()  
(*kiwi.partitionner.base.PartitionerBase*  
method), 123  
resize\_table()  
(*kiwi.partitionner.dasd.PartitionerDasd*  
method), 124  
resize\_table()  
(*kiwi.partitionner.gpt.PartitionerGpt*  
method), 124  
resize\_table()  
(*kiwi.partitionner.msdos.PartitionerMsDos*  
method), 125  
Result (class in *kiwi.system.result*), 150  
Result\_type (class in  
*kiwi.system.result*), 151

ResultBundleTask (class in *kiwi.tasks.result\_bundle*), 164

ResultListTask (class in *kiwi.tasks.result\_list*), 164

RootBind (class in *kiwi.system.root\_bind*), 152

RootInit (class in *kiwi.system.root\_init*), 153

run() (*kiwi.command.Command* class method), 177

run\_common\_function() (*kiwi.system.shell.Shell* class method), 157

runtime\_config() (*kiwi.repository.base.RepositoryBase* method), 128

runtime\_config() (*kiwi.repository.yum.RepositoryYum* method), 129

runtime\_config() (*kiwi.repository.zypper.RepositoryZypper* method), 132

RuntimeChecker (class in *kiwi.runtime\_checker*), 207

RuntimeConfig (class in *kiwi.runtime\_config*), 209

## S

Sat (class in *kiwi.solver.sat*), 162

set\_color\_format() (*kiwi.logger.Logger* method), 204

set\_container\_config\_tag() (*kiwi.xml\_state.XMLState* method), 223

set\_derived\_from\_image\_uri() (*kiwi.xml\_state.XMLState* method), 223

set\_flag() (*kiwi.partitioner.base.PartitionerBase* method), 123

set\_flag() (*kiwi.partitioner.gpt.PartitionerGpt* method), 124

set\_flag() (*kiwi.partitioner.msdos.PartitionerMsDos* method), 125

set\_hybrid\_mbr() (*kiwi.partitioner.base.PartitionerBase* method), 123

set\_hybrid\_mbr() (*kiwi.partitioner.gpt.PartitionerGpt* method), 124

set\_logfile() (*kiwi.logger.Logger* method), 204

set\_mbr() (*kiwi.partitioner.base.PartitionerBase* method), 123

set\_mbr() (*kiwi.partitioner.gpt.PartitionerGpt* method), 125

set\_media\_tag() (*kiwi.iso\_tools.iso.Iso* class method), 114

set\_property\_readonly\_root() (*kiwi.volume\_manager.base.VolumeManagerBase* method), 171

set\_property\_readonly\_root() (*kiwi.volume\_manager.btrfs.VolumeManagerBtrfs* method), 172

set\_python\_default\_encoding\_to\_utf8() (*kiwi.defaults.Defaults* class method), 188

set\_repository() (*kiwi.xml\_state.XMLState* method), 223

set\_selinux\_file\_contexts() (*kiwi.system.setup.SystemSetup* method), 156

setLogLevel() (*kiwi.logger.Logger* method), 204

setup() (*kiwi.container.setup.base.ContainerSetupBase* method), 103

setup() (*kiwi.volume\_manager.base.VolumeManagerBase* method), 171

setup() (*kiwi.volume\_manager.btrfs.VolumeManagerBtrfs* method), 172

setup() (*kiwi.volume\_manager.lvm.VolumeManagerLVM* method), 174

setup\_disk\_boot\_images() (*kiwi.bootloader.config.base.BootLoaderConfigBase* method), 85

setup\_disk\_boot\_images() (*kiwi.bootloader.config.grub2.BootLoaderConfigGrub2* method), 86

setup\_disk\_boot\_images() (*kiwi.bootloader.config.zipl.BootLoaderConfigZipl* method), 89

setup\_disk\_image\_config() (*kiwi.bootloader.config.base.BootLoaderConfigBase* method), 89



[method](#)), [85](#) [setup\\_live\\_image\\_config\(\)](#)  
([kiwi.bootloader.config.grub2.BootLoaderConfigGrub2](#)), [86](#)  
[method](#)), [87](#) [setup\\_live\\_image\\_config\(\)](#)  
[setup\\_disk\\_image\\_config\(\)](#) ([kiwi.bootloader.config.grub2.BootLoaderConfigGrub2](#)), [87](#)  
([kiwi.bootloader.config.zipl.BootLoaderConfigZipl](#)), [87](#)  
[method](#)), [90](#) [setup\\_live\\_image\\_config\(\)](#)  
[setup\\_groups\(\)](#) ([kiwi.bootloader.config.isolinux.BootLoaderConfigIsolinux](#)), [89](#)  
([kiwi.system.setup.SystemSetup](#) [method](#)), [89](#)  
[method](#)), [156](#) [setup\\_locale\(\)](#)  
[setup\\_home\\_for\\_user\(\)](#) ([kiwi.system.setup.SystemSetup](#) [method](#)), [156](#)  
([kiwi.system.users.Users](#) [method](#)), [159](#) [setup\\_mountpoint\(\)](#)  
[setup\\_install\\_boot\\_images\(\)](#) ([kiwi.volume\\_manager.base.VolumeManagerBase](#) [method](#)), [171](#)  
([kiwi.bootloader.config.base.BootLoaderConfigBase](#) [method](#)), [85](#) [setup\\_plymouth\\_splash\(\)](#)  
[setup\\_install\\_boot\\_images\(\)](#) ([kiwi.system.setup.SystemSetup](#) [method](#)), [156](#)  
([kiwi.bootloader.config.grub2.BootLoaderConfigGrub2](#)), [87](#) [setup\\_repositories\(\)](#)  
[setup\\_install\\_boot\\_images\(\)](#) ([kiwi.system.prepare.SystemPrepare](#) [method](#)), [149](#)  
([kiwi.bootloader.config.isolinux.BootLoaderConfigIsolinux](#)), [88](#) [setup\\_root\\_console\(\)](#)  
[setup\\_install\\_image\\_config\(\)](#) ([kiwi.container.setup.base.ContainerSetupBase](#) [method](#)), [103](#)  
([kiwi.bootloader.config.base.BootLoaderConfigBase](#) [method](#)), [85](#) [setup\\_static\\_device\\_nodes\(\)](#)  
[setup\\_install\\_image\\_config\(\)](#) ([kiwi.container.setup.base.ContainerSetupBase](#) [method](#)), [103](#)  
([kiwi.bootloader.config.grub2.BootLoaderConfigGrub2](#)), [87](#) [setup\\_sysconfig\\_bootloader\(\)](#)  
[setup\\_install\\_image\\_config\(\)](#) ([kiwi.bootloader.config.base.BootLoaderConfigBase](#) [method](#)), [86](#)  
([kiwi.bootloader.config.isolinux.BootLoaderConfigIsolinux](#)), [88](#) [setup\\_sysconfig\\_bootloader\(\)](#)  
[setup\\_intermediate\\_config\(\)](#) ([kiwi.bootloader.config.grub2.BootLoaderConfigGrub2](#) [method](#)), [88](#)  
([kiwi.system.root\\_bind.RootBind](#) [method](#)), [152](#) [setup\\_timezone\(\)](#)  
[setup\\_isolinux\\_boot\\_path\(\)](#) ([kiwi.system.setup.SystemSetup](#) [method](#)), [156](#)  
([kiwi.iso\\_tools.iso.Iso](#) [method](#)), [115](#) [setup\\_users\(\)](#)  
[setup\\_keyboard\\_map\(\)](#) ([kiwi.system.setup.SystemSetup](#) [method](#)), [156](#) [method](#)), [156](#)  
([kiwi.system.setup.SystemSetup](#) [method](#)), [156](#)  
[setup\\_live\\_boot\\_images\(\)](#) [sextuple\\_token\(\)](#)  
([kiwi.bootloader.config.base.BootLoaderConfigBase](#) [method](#)), [163](#)  
[method](#)), [85](#) [sha256\(\)](#) ([kiwi.utils.checksum.Checksum](#) [method](#)), [167](#)  
[setup\\_live\\_boot\\_images\(\)](#) ([kiwi.bootloader.config.grub2.BootLoaderConfigGrub2](#)), [87](#)  
([kiwi.bootloader.config.grub2.BootLoaderConfigGrub2](#)), [87](#) [shasum](#) ([kiwi.system.result.result\\_file\\_type](#) [attribute](#)), [151](#)  
[setup\\_live\\_boot\\_images\(\)](#) ([kiwi.bootloader.config.isolinux.BootLoaderConfigIsolinux](#) [method](#)), [89](#)  
([kiwi.bootloader.config.isolinux.BootLoaderConfigIsolinux](#) [method](#)), [89](#) [show\(\)](#) ([kiwi.help.Help](#) [method](#)), [201](#)

[show\\_and\\_exit\\_on\\_help\\_request\(\)](#) (*kiwi.cli.Cli* method), 176  
[solve\(\)](#) (*kiwi.solver.sat.Sat* method), 162  
[SolverRepository](#) (class in *kiwi.solver.repository*), 162  
[SolverRepositoryBase](#) (class in *kiwi.solver.repository.base*), 160  
[SolverRepositoryRpmDir](#) (class in *kiwi.solver.repository.rpm\_dir*), 161  
[SolverRepositoryRpmMd](#) (class in *kiwi.solver.repository.rpm\_md*), 161  
[SolverRepositorySUSE](#) (class in *kiwi.solver.repository.suse*), 161  
[sort\\_by\\_hierarchy\(\)](#) (*kiwi.path.Path* class method), 206  
[store\\_to\\_result\(\)](#) (*kiwi.storage.subformat.base.DiskFormatBase* method), 134  
[store\\_to\\_result\(\)](#) (*kiwi.storage.subformat.ova.DiskFormatOva* method), 135  
[store\\_to\\_result\(\)](#) (*kiwi.storage.subformat.vhdfixed.DiskFormatVhdFixed* method), 138  
[store\\_to\\_result\(\)](#) (*kiwi.storage.subformat.vmdk.DiskFormatVmdk* method), 139  
[sync\\_data\(\)](#) (*kiwi.filesystem.base.FileSystemBase* method), 107  
[sync\\_data\(\)](#) (*kiwi.utils.sync.DataSync* method), 168  
[sync\\_data\(\)](#) (*kiwi.volume\_manager.base.VolumeManagerBase* method), 171  
[sync\\_data\(\)](#) (*kiwi.volume\_manager.btrfs.VolumeManagerBtrfs* method), 172  
[sync\\_data\(\)](#) (*kiwi.volume\_manager.btrfs.VolumeManagerBtrfs* method), 172  
[sync\\_data\(\)](#) (*kiwi.volume\_manager.lvm.VolumeManagerLVM* method), 174  
[SysConfig](#) (class in *kiwi.utils.sysconfig*), 168  
[SystemBuildTask](#) (class in *kiwi.tasks.system\_build*), 164  
[SystemCreateTask](#) (class in *kiwi.tasks.system\_create*), 165  
[SystemIdentifier](#) (class in *kiwi.system.identifier*), 146  
[SystemPrepare](#) (class in *kiwi.system.prepare*), 148  
[SystemPrepareTask](#) (class in *kiwi.tasks.system\_prepare*), 165  
[SystemSetup](#) (class in *kiwi.system.setup*), 153  
[SystemSize](#) (class in *kiwi.system.size*), 157  
[SystemUpdateTask](#) (class in *kiwi.tasks.system\_update*), 165

## T

[target\\_supports\\_extended\\_attributes\(\)](#) (*kiwi.utils.sync.DataSync* method), 168  
[timestamp\(\)](#) (*kiwi.solver.repository.base.SolverRepositoryBase* method), 161  
[timestamp\(\)](#) (*kiwi.solver.repository.rpm\_md.SolverRepositoryRpmMd* method), 161  
[to\\_profile\(\)](#) (*kiwi.defaults.Defaults* method), 188  
[translate\(\)](#) (*kiwi.system.uri.Uri* method), 159

## U

[umount\(\)](#) (*kiwi.mount\_manager.MountManager* method), 205  
[umount\\_lazy\(\)](#) (*kiwi.mount\_manager.MountManager* method), 205  
[umount\\_volumes\(\)](#) (*kiwi.volume\_manager.base.VolumeManagerBase* method), 171  
[umount\\_volumes\(\)](#) (*kiwi.volume\_manager.btrfs.VolumeManagerBtrfs* method), 172  
[umount\\_volumes\(\)](#) (*kiwi.volume\_manager.lvm.VolumeManagerLVM* method), 174  
[uncompress\(\)](#) (*kiwi.utils.compress.Compress* method), 168  
[update\(\)](#) (*kiwi.package\_manager.base.PackageManagerBase* method), 117  
[update\(\)](#) (*kiwi.package\_manager.yum.PackageManagerYum* method), 119  
[update\(\)](#) (*kiwi.package\_manager.zypper.PackageManagerZypper* method), 121  
[update\\_system\(\)](#)

*(kiwi.system.prepare.SystemPrepare method)*, 149

*Uri (class in kiwi.system.uri)*, 158

*usage() (in module kiwi.kiwi)*, 201

*use\_default\_location() (kiwi.repository.base.RepositoryBase method)*, 128

*use\_default\_location() (kiwi.repository.yum.RepositoryYum method)*, 129

*use\_default\_location() (kiwi.repository.zypper.RepositoryZypper method)*, 132

*use\_for\_bundle (kiwi.system.result.result\_file\_type attribute)*, 151

*user\_add() (kiwi.system.users.Users method)*, 160

*user\_exists() (kiwi.system.users.Users method)*, 160

*user\_modify() (kiwi.system.users.Users method)*, 160

*Users (class in kiwi.system.users)*, 159

*write() (kiwi.bootloader.config.grub2.BootLoaderConfigG method)*, 88

*write() (kiwi.bootloader.config.isolinux.BootLoaderConfig method)*, 89

*write() (kiwi.bootloader.config.zipl.BootLoaderConfigZipl method)*, 90

*write() (kiwi.repository.yum.RepositoryYumSpaceRemove method)*, 130

*write() (kiwi.system.identifier.SystemIdentifier method)*, 147

*write() (kiwi.utils.sysconfig.SysConfig method)*, 168

*write\_to\_disk() (kiwi.system.identifier.SystemIdentifier method)*, 147

## X

*XMLDescription (class in kiwi.xml\_description)*, 210

*XMLState (class in kiwi.xml\_state)*, 211

*xz() (kiwi.utils.compress.Compress method)*, 168

## V

*verify\_image\_size() (kiwi.system.result.Result method)*, 151

*VmwareSettingsTemplate (class in kiwi.storage.subformat.template.vmware\_settings)*, 132

*VolumeManager (class in kiwi.volume\_manager)*, 174

*VolumeManagerBase (class in kiwi.volume\_manager.base)*, 169

*VolumeManagerBtrfs (class in kiwi.volume\_manager.btrfs)*, 171

*VolumeManagerLVM (class in kiwi.volume\_manager.lvm)*, 173

## W

*WarningFilter (class in kiwi.logger)*, 204

*which() (kiwi.path.Path class method)*, 206

*wipe() (kiwi.path.Path class method)*, 206

*wipe() (kiwi.storage.disk.Disk method)*, 142

*write() (kiwi.bootloader.config.base.BootLoaderConfigBase method)*, 86