Building Linux System Appliances

Building Linux System Appliances

Publication Date: 03 Jun 2026

https://documentation.suse.com

Contents

Preface xiv

1 The Appliance Concept xv

2 Use Cases xv

3 Contact xvi

1 Overview 1

1.1 Basic Workflow 1

Components of an Image Description 2

1.2 Conceptual Overview 3

1.3 Terminology 4

1.4 System Requirements 5

2 Installation 6

2.1 Installation from OBS 6

2.2 Installation from Distribution Repositories 7

2.3 Installation for SUSE Linux Enterprise 8

2.4 Example Appliance Descriptions 8

3 Quick Start 10

3.1 Before you start 10

3.2 Choose a First Image 10

3.3 Build your First Image 11

3.4 Run your Image 11

3.5 Tweak and Customize your Image 11

4 Working from the Command Line 12

4.1 kiwi-ng 12

SYNOPSIS 12 • DESCRIPTION 13 • GLOBAL OPTIONS 14 • EXAMPLE 15

4.2 kiwi-ng result list 16

SYNOPSIS 16 • DESCRIPTION 16 • OPTIONS 16

4.3 kiwi-ng result bundle 16

SYNOPSIS 16 • DESCRIPTION 17 • OPTIONS 17

4.4 kiwi-ng system prepare 18

SYNOPSIS 18 • DESCRIPTION 19 • OPTIONS 19

4.5 kiwi-ng system update 23

SYNOPSIS 23 • DESCRIPTION 23 • OPTIONS 23

4.6 kiwi-ng system build 23

SYNOPSIS 23 • DESCRIPTION 24 • OPTIONS 24 • URI_TYPES 28

4.7 kiwi-ng system create 28

SYNOPSIS 28 • DESCRIPTION 29 • OPTIONS 29

4.8 kiwi-ng image resize 29

SYNOPSIS 29 • DESCRIPTION 30 • OPTIONS 30

4.9 kiwi-ng image info 30

SYNOPSIS 30 • DESCRIPTION 31 • OPTIONS 31

5 Troubleshooting 33

5.1 Build Host Constraints 33

Package Manager Behavior 34

5.2 Architectures 35

5.3 Host Security Settings Conflicts with KIWI 36

5.4 Incompatible Filesystem Settings on Host vs. Image 38

5.5 Boxbuild Tweaks 39

Increase Box Build Image Size 39

6 KIWI Plugins 41

6.1 Building in a Self-Contained Environment 41

Requirements 42 • Building with the boxbuild command 42 • Sharing Backends 43 • Building in Container 44

6.2 Building based on Containers 45

Installation 46 • Concept 46 • Create a stash 46 • Rebuild from a stash 47 • Turn a container into a VM image 48

7 Concept and Workflow 51

7.1 Host Requirements To Build Images 51

7.2 Setting up Repositories 53

Adding repositories 53

7.3 Adding and Removing Packages 56

The `package` element 57 • The `archive` element 58 • Uninstall System Packages 58 • The `product` and `namedCollection` element 60 • The `ignore` element 61

7.4 Image Profiles 61

7.5 Adding Users 62

7.6 User-Defined Scripts 63

Developing/Debugging Scripts 65

7.7 The Runtime Configuration File 71

7.8 Customizing the Boot Process 72

Boot Image Hook-Scripts 73 • Boot Image Parameters 75

7.9 Overview 80

7.10 Image Building Process 81

The Prepare Step 82 • The Create Step 85

8 Image Description 86

8.1 Image Description Elements 86

<image> 86 • <include> 87 • <description> 88 • <certificates> 88 • <preferences> 89 •

8.2 Main Root 134

8.3 Image Identity 134

8.4 Image Preferences 134

8.5 Image Software Sources 134

8.6 Image Content Setup 134

8.7 Image Users 135

8.8 Image Namespace 135

8.9 Image Includes 135

9 Image Types 136

9.1 Image Results 137

9.2 Image Bundle Format 140

10 Building Images for Supported Types 143

10.1 Build an ISO Hybrid Live Image 143

`overlay` or `dmsquash` 144

10.2 Build a Virtual Disk Image 146

Setting up the Bootloader in the Image 147 • Modifying the Size of the Image 148 • Customizing the Virtual Machine 148

10.3 Build an Expandable Disk Image 152

Deployment Methods 153 • Manual Deployment 153 • CD/DVD Deployment 154 • Network Deployment 155 • OEM Customization 158 • Installation Media Customization 160

10.4 Build a Container Image 161

10.5 Build a WSL Container Image 163

Setup of the WSL-DistroLauncher 164 • Testing the WSL image 167

10.6 Build KIS Image (Kernel, Initrd, System) 167

10.7 Build an AWS Nitro Enclave 169

11 Working with Images 171

11.1 Deploy ISO Image on an USB Stick 171

11.2 Deploy ISO Image as File on a FAT32 Formated USB Stick 172

11.3 Booting a Live ISO Images from Grub2 173

11.4 Image Description for Amazon EC2 174

11.5 Image Description for Microsoft Azure 177

11.6 Image Description for Google Compute Engine 178

11.7 Image Description for Vagrant 179

Customizing the embedded Vagrantfile 183

11.8 Image Description Encrypted Disk 184

11.9 Deploy and Run System in a RamDisk 186

11.10 Custom Disk Partitions 188

11.11 Custom Disk Volumes 191

11.12 Partition Clones 193

Use Case 195

11.13 Setting Up a Network Boot Server 196

Installing and Configuring DHCP and TFTP with dnsmasq 197

11.14 Build PXE Root File System Image for the legacy netboot infrastructure 199

PXE Client Setup Configuration 201

11.15 Booting a Root Filesystem from Network 206

11.16 Booting a Live ISO Image from Network 209

Available Remote Boot Options 211 • Persistent Live System 211

11.17 Setting Up YaST at First Boot 212

11.18 Add or Update the Fstab File 213

11.19 Building Images with Profiles 215

Local Builds 215 • Building with the Open Build Service 215

11.20 Building in the Open Build Service 216
Advantages of using the Open Build Service (OBS) 216 • Differences Between Building Locally and on OBS 217 • Recommendations 219

11.21 Using SUSE Product ISO To Build 222

11.22 Circumvent Debian Bootstrap 223
How to Create a bootstrap_package 224

12 Contributing 226

12.1 Using KIWI NG in a Python Project 226

12.2 Plugin Architecture 227
Naming conventions 228 • Example plugin 228

12.3 Write Integration Tests for the Scripts 230
Test setup 230

12.4 Extending KIWI NG with Custom Operations 231
The <extension> Section 232 • RELAX NG Schema for the Extension 232 • Extension schema in XML catalog 233 • Using the extension 234

12.5 The Basics 234

12.6 Fork the upstream repository 234

12.7 Create a local clone of the forked repository 234

12.8 Install Required Operating System Packages 235

12.9 Create a Python Virtual Development Environment 235

12.10 Running the Unit Tests 235

12.11 Create a Branch for each Feature or Bugfix 236

12.12 Coding Style 237
Documentation 237

12.13 Additional Information 238
Signing Git Patches 238 • Bumping the Version 239 • Creating a RPM Package 239

13 Building Build Tests 241

14 Python API 243

14.1 kiwi Package 243

Submodules 243 • kiwi.app Module 243 • kiwi.cli Module 243 • kiwi.command Module 245 • kiwi.command_process Module 247 • kiwi.defaults Module 249 • kiwi.exceptions Module 276 • kiwi.firmware Module 290 • kiwi.help Module 293 • kiwi.kiwi Module 293 • kiwi.logger Module 294 • kiwi.logger_color_formatter Module 296 • kiwi.logger_filter Module 297 • kiwi.mount_manager Module 299 • kiwi.path Module 301 • kiwi.privileges Module 304 • kiwi.runtime_checker Module 304 • kiwi.runtime_config Module 308 • kiwi.version Module 314 • kiwi.xml_description Module 314 • kiwi.xml_state Module 315 • Module Contents 350

14.2 kiwi.archive Package 350

Submodules 350 • kiwi.archive.cpio Module 350 • kiwi.archive.tar Module 351 • Module Contents 353

14.3 kiwi.boot.image Package 353

Submodules 353 • kiwi.boot.image.base Module 353 • kiwi.boot.image.dracut Module 357 • kiwi.boot.image.builtin_kiwi Module 360 • Module Contents 360

14.4 kiwi.bootloader.config Package 361

Submodules 361 • kiwi.bootloader.config.base Module 361 • kiwi.bootloader.config.grub2 Module 368 • kiwi.bootloader.config.systemd_boot Module 371 • kiwi.bootloader.config.zipl Module 371 • Module Contents 372

14.5 kiwi.bootloader.install Package 372

Submodules 372 • kiwi.bootloader.install.base Module 372 • kiwi.bootloader.install.grub2 Module 373 • kiwi.bootloader.install.systemd_boot

Module 374 • kiwi.bootloader.install.zipl Module 375 • Module Contents 376

14.6 kiwi.bootloader Package 376

Module Contents 376

14.7 kiwi.bootloader.template Package 376

Submodules 376 • kiwi.bootloader.template.grub2 Module 376 • Module Contents 378

14.8 kiwi.boot Package 378

Module Contents 378

14.9 kiwi.builder Package 378

Submodules 378 • kiwi.builder.archive Module 378 • kiwi.builder.container Module 379 • kiwi.builder.disk Module 380 • kiwi.builder.filesystem Module 382 • kiwi.builder.install Module 383 • kiwi.builder.live Module 384 • kiwi.builder.kis Module 385 • Module Contents 386

14.10 kiwi.container Package 386

Submodules 386 • kiwi.container.oci Module 386 • Module Contents 388

14.11 kiwi.container.setup Package 388

Submodules 388 • kiwi.container.setup.base Module 388 • kiwi.container.setup.docker Module 390 • Module Contents 390

14.12 kiwi.filesystem Package 390

Submodules 390 • kiwi.filesystem.base Module 390 • kiwi.filesystem.btrfs Module 394 • kiwi.filesystem.ext2 Module 394 • kiwi.filesystem.ext3 Module 395 • kiwi.filesystem.ext4 Module 395 • kiwi.filesystem.fat16 Module 396 • kiwi.filesystem.fat32

Module 396 • kiwi.filesystem.isofs
Module 397 • kiwi.filesystem.setup
Module 397 • kiwi.filesystem.squashfs
Module 398 • kiwi.filesystem.xfs Module 399 • Module Contents 399

14.13 kiwi.iso_tools Package 400
Submodules 400 • kiwi.iso_tools.base
Module 400 • kiwi.iso_tools.xorriso
Module 401 • kiwi.iso_tools.iso Module 402 • Module Contents 403

14.14 kiwi.package_manager Package 403
Submodules 403 • kiwi.package_manager.base
Module 403 • kiwi.package_manager.dnf4
Module 407 • kiwi.package_manager.zypper Module 410 • Module Contents 414

14.15 kiwi.partitioner Package 415
Submodules 415 • kiwi.partitioner.base
Module 415 • kiwi.partitioner.dasd
Module 417 • kiwi.partitioner.gpt
Module 418 • kiwi.partitioner.msdos Module 419 • Module Contents 421

14.16 kiwi.repository Package 421
Submodules 421 • kiwi.repository.base
Module 421 • kiwi.repository.dnf4
Module 424 • kiwi.repository.zypper Module 426 • Module Contents 429

14.17 kiwi.repository.template Package 430
Submodules 430 • kiwi.repository.template.apt
Module 430 • Module Contents 430

14.18 kiwi.solver.repository Package 430
Submodules 430 • kiwi.solver.repository.base
Module 430 • Module Contents 432

14.19 kiwi.solver Package 433
Submodules 433 • kiwi.solver.sat Module 433 • Module Contents 434
14.20 kiwi.storage Package 434
Submodules 434 • kiwi.storage.device_provider Module 434 • kiwi.storage.disk Module 435 • kiwi.storage.loop_device Module 441 • kiwi.storage.luks_device Module 442 • kiwi.storage.mapped_device Module 443 • kiwi.storage.raid_device Module 444 • kiwi.storage.clone_device Module 445 • kiwi.storage.setup Module 445 • Module Contents 447
14.21 kiwi.storage.subformat Package 447
Submodules 447 • kiwi.storage.subformat.base Module 447 • kiwi.storage.subformat.gce Module 449 • kiwi.storage.subformat.ova Module 450 • kiwi.storage.subformat.qcow2 Module 451 • kiwi.storage.subformat.vagrant_base Module 452 • kiwi.storage.subformat.vagrant_libvirt Module 454 • kiwi.storage.subformat.vagrant_virtualbox Module 455 • kiwi.storage.subformat.vdi Module 456 • kiwi.storage.subformat.vhd Module 457 • kiwi.storage.subformat.vhdfixed Module 457 • kiwi.storage.subformat.vhdx Module 458 • kiwi.storage.subformat.vmdk Module 458 • Module Contents 459
14.22 kiwi.storage.subformat.template Package 459
Submodules 459 • kiwi.storage.subformat.template.vmware_settings Module 459 • kiwi.storage.subformat.template.vagrant_config Module 460 • kiwi.storage.subformat.template.virtualbox_ovf Module 461 • Module contents 462

14.23 kiwi.system Package 462
Submodules 462 • kiwi.system.identifier
Module 462 • kiwi.system.kernel
Module 463 • kiwi.system.prepare
Module 465 • kiwi.system.profile
Module 468 • kiwi.system.result
Module 469 • kiwi.system.root_bind
Module 471 • kiwi.system.root_init
Module 473 • kiwi.system.setup Module 474 • kiwi.system.shell
Module 479 • kiwi.system.size Module 481 • kiwi.system.uri
Module 482 • kiwi.system.users Module 484 • Module
Contents 485

14.24 kiwi.tasks package 485
Submodules 485 • kiwi.tasks.base
Module 485 • kiwi.tasks.result_bundle
Module 487 • kiwi.tasks.result_list
Module 488 • kiwi.tasks.system_build
Module 488 • kiwi.tasks.system_create
Module 488 • kiwi.tasks.system_prepare
Module 489 • kiwi.tasks.system_update Module 489 • Module
Contents 490

14.25 kiwi.utils Package 490
Submodules 490 • kiwi.utils.checksum
Module 490 • kiwi.utils.block Module 491 • kiwi.utils.compress
Module 492 • kiwi.utils.sync Module 493 • kiwi.utils.sysconfig
Module 494 • Module Contents 495

14.26 kiwi.volume_manager Package 495
Submodules 495 • kiwi.volume_manager.base
Module 495 • kiwi.volume_manager.btrfs
Module 500 • kiwi.volume_manager.lvm Module 502 • Module
Contents 504

Preface

Note

This documentation covers KIWI Next Generation (KIWI NG) 10.2.35- the command line utility to build Linux system appliances. If you are using a KIWI NG schema version older than v74, upgrade the kiwi file as follows:

```
$ xsltproc /usr/share/kiwi/xsl_to_v74/update.xsl config.xml|*.kiwi
```

- GitHub Sources (https://github.com/OSInside/kiwi)
- GitHub Releases (https://github.com/OSInside/kiwi/releases)
- RPM Packages (http://download.opensuse.org/repositories/Virtualization:/Appliances:/Builder)
- Build Tests TumbleWeed(x86) (https://build.opensuse.org/project/show/Virtualization:Appliances:Images:Testing_x86:tumbleweed)
- Build Tests TumbleWeed(arm) (https://build.opensuse.org/project/show/Virtualization:Appliances:Images:Testing_arm:tumbleweed)
- Build Tests TumbleWeed(s390) (https://build.opensuse.org/project/show/Virtualization:Appliances:Images:Testing_s390:tumbleweed)
- Build Tests TumbleWeed(ppc64le) (https://build.opensuse.org/project/show/Virtualization:Appliances:Images:Testing_ppc:tumbleweed)
- Build Tests RawHide(x86) (https://build.opensuse.org/project/show/Virtualization:Appliances:Images:Testing_x86:rawhide)
- Build Tests RawHide(s390) (https://build.opensuse.org/project/show/Virtualization:Appliances:Images:Testing_s390:rawhide)
- Build Tests Leap(x86) (https://build.opensuse.org/project/show/Virtualization:Appliances:Images:Testing_x86:leap)
- Build Tests Fedora(x86) (https://build.opensuse.org/project/show/Virtualization:Appliances:Images:Testing_x86:fedora)

- Build Tests Fedora(arm) (https://build.opensuse.org/project/show/Virtualization:Appliances:Images:Testing_arm:fedora)
- Build Tests CentOS(x86) (https://build.opensuse.org/project/show/Virtualization:Appliances:Images:Testing_x86:centos)
- Build Tests Ubuntu(x86) (https://build.opensuse.org/project/show/Virtualization:Appliances:Images:Testing_x86:ubuntu)
- Build Tests Ubuntu(arm) (https://build.opensuse.org/project/show/Virtualization:Appliances:Images:Testing_arm:ubuntu)
- Build Tests Debian(x86) (https://build.opensuse.org/project/show/Virtualization:Appliances:Images:Testing_x86:debian)
- Build Tests ArchLinux(x86) (https://build.opensuse.org/project/show/Virtualization:Appliances:Images:Testing_x86:archlinux)

1 The Appliance Concept

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

KIWI NG can create appliances in various forms: beside classical installation ISOs and images for virtual machines it can also build images that boot via PXE or Vagrant boxes.

In KIWI NG, the appliance is specified via a collection of human readable files in a directory, also called the `image description`. At least one XML file `config.xml` or `.kiwi` is required. In addition there may as well be other files like scripts or configuration data.

2 Use Cases

The following list shows a selection of use cases for which an appliance is needed:

Private and Public Clouds

Cloud environments are managed through an API provided by the cloud service provider. The classic way to install a machine is not possible in such an environment because there is no physical access to the machine. An appliance is needed to be registered with the cloud.

Custom Linux Distribution

Linux distributors provide their distribution based on a collection of packages and release them on an install media like a DVD or an USB stick. Typically a lot more software components exists for the distribution which are not part of the default installation media or the installation media comes with software and installation routines that are not matching your target audience. With an appliance made by KIWI NG you can create an installation media that matches custom criteria as needed by the customer and does not require extra post processing steps after the default installation method provided by the distributor.

Live Systems

The ability to have a Linux OS that runs from a small storage device like a USB stick or a SD card is the swiss army knife of many system administrators. The creation of such a live system includes use of technologies that are not part of a standard installation process. An appliance builder is needed to create this sort of system.

Embedded Systems

Embedded Systems like the Raspberry Pi come with limited hardware components. Their boot sequences often do not allow for classic installation methods through USB or DVD devices. Instead they boot through SD card slots or via the network. SoC (System on Chip) devices also tend to implement non standard boot methods which can only be implemented through custom OS appliances.

And More

...

3 Contact

- Mailing list (https://groups.google.com/forum/#!forum/kiwi-images)

  The `kiwi-images` group is an open group and anyone can subscribe (mailto:kiwi-images+subscribe@googlegroups.com), even if you do not have a Google account.

- Matrix (https://matrix.org)

  An open network for secure, decentralized communication. Please find the `#kiwi:matrix.org` room via Matrix (https://matrix.to/#kiwi:matrix.org) on the web or by using the supported clients (https://matrix.org/ecosystem/clients).

1 Overview

Note

Abstract

This document provides a conceptual overview about the steps of creating an image with KIWI NG. It also explains the terminology regarding the concept and process when building system images with KIWI NG 10.2.35.

1.1 Basic Workflow

Note

Abstract

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

This description applies for version 10.2.35.

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

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

- can be deployed inside an existing virtual environment without requiring configuration at start-up.
- automatically configure themselves in a known target environment.
- prompt the user for an interactive system configuration.

The image creation process with KIWI NG is automated and does not require any user interaction. The information required for the image creation process is provided by the primary configuration file named `config.xml`. This file is validated against the schema documented in: *Chapter 8, Image Description*.

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 (#description-components) 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.

1.1.1 Components of an Image Description

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

These are the optional components of an image description:

1. `config.sh` shell script
   Is the configuration shell script that runs at the end of the *Section 7.10.1, "The Prepare Step"* if present. It can be used to fine tune the unpacked image.
   Note that the script is directly invoked by the operating system if its executable bit is set. Otherwise it is called by `bash` instead.

2. `images.sh` shell script

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

3. Overlay tree directory

   The *overlay tree* is a folder (called `root`) or a tarball file (called `root.tar.gz`) that contains files and directories that will be copied to the target image build tree during the *Section 7.10.1, "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>
```

1.2 Conceptual Overview

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

KIWI NG creates images in a two step process:

1. The first step, the *prepare operation*, generates a so-called *unpacked image tree* (directory) using the information provided in the image description.

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

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

1.3 Terminology

Appliance

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

Image

The result of a KIWI NG build process.

Image Description

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

Overlay Files

A directory structure with files and subdirectories stored as part of the Image Description. This directory structure is packaged as a file `root.tar.gz` or stored inside a directory named `root`. Additional overlay directories for selected profiles are supported too and are taken into account if the directory name matches the name of the profile. The content of each of the directory structures is copied on top of the existing file system (overlayed) of the appliance root. This also includes permissions and attributes as a supplement.

KIWI NG

An OS appliance builder.

Virtualization Technology

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

1.4 System Requirements

To use and run KIWI NG, you need:

- A recent Linux distribution, see *Section 5.1, "Build Host Constraints"* for details.
- Enough free disk space to build and store the image. We recommend a minimum of 15GB.
- Python version 3.5 or higher
- Git (package `git`) to clone a repository.
- Any virtualization technology to start the image. We recommend QEMU.

The project officially tests and supports the following distributions:

- Fedora Linux (https://fedoraproject.org) 40+
- CentOS Stream (https://centos.org) 9+
- CentOS Stream Hyperscale (https://centos.org/hyperscale) 9+
- openSUSE Tumbleweed (https://get.opensuse.org/tumbleweed/) after April 2024
- openSUSE Leap 16.0+ (https://get.opensuse.org/leap/16.0/)
- Canonical Ubuntu Linux (https://ubuntu.com/) 24.04+
- Debian GNU/Linux (https://debian.org) 12+
- Arch Linux (https://archlinux.org) after April 2024

2 Installation

Note

This document describes how to install KIWI NG.

Apart from the preferred method to install KIWI NG via a distribution package manager, it is also available on pypi (https://pypi.org/project/kiwi/) and can be installed using Python's package manager pip as follows:

```
$ sudo pip install kiwi
```

2.1 Installation from OBS

The most up to date packages of KIWI NG can be found on the Open Build Service in the Virtualization:Appliances:Builder (https://download.opensuse.org/repositories/Virtualization:/Appliances:/Builder) project.

To install KIWI NG, follow these steps:

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

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

4. Install KIWI NG:

   ```
   $ sudo zypper --gpg-auto-import-keys install python3-kiwi
   ```

Note

Other Distributions

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

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

$ sudo dnf install python3-kiwi
```

2.2 Installation from Distribution Repositories

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

Note

There are many packages that contain the name *kiwi* in their name, some of these are not even python packages. Please double check the packages' description whether it is actually the KIWI NG Appliance builder before installing it. Please also note, depending on how the responsible packager has integrated KIWI NG into the distribution, the install name can be different from the instructions provided in: Installation from OBS (#installation-from-obs)

To install KIWI NG for the desired distribution, run the following command:

Leap/Tumbleweed:

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

Fedora/Rawhide:

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

2.3 Installation for SUSE Linux Enterprise

KIWI NG is available and supported for SUSE Linux Enterprise (SLE). The recommended and supported way is to install KIWI NG by using zypper.

However, if you rely on some plugins for KIWI NG, either the plugin itself or any dependencies might not be available for your service pack.

If you want to proceed anyway, keep these things in mind:

- Plugins that are not provided by SLE are not supported.
- You probably need to install dependencies via **`pip`**. The **`pip`** command installs these dependencies from PyPI (the Python Package Index). However, this approach will not update the RPM database.
- Depending on your security concerns, installing Python packages outside the secured SLE installation may not be desirable.
- Python packages installed from PyPI won't contain any SUSE customizations.
- Depending on your extension and its dependencies, you might even need a more recent Python version.

2.4 Example Appliance Descriptions

There are two places for example appliance descriptions:

The KIWI NG build tests:

The KIWI NG project itself hosts a collection of appliance descriptions which are used for integration testing of the KIWI NG builder itself. These descriptions are required to build prior any KIWI NG release and are also used as the base for this documentation. Please check them out when working with this reference guide:

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

$ tree -L 3 kiwi/build-tests
```

The KIWI NG community descriptions project:

There is a GitHub project hosting example appliance descriptions to be used with the next generation KIWI NG. Contributions from the community makes up the contents of this repository and users who need an example for a specific use case and distribution can clone the project as follows:

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

3 Quick Start

Note

Abstract

This document describes how to start working with KIWI NG, an OS appliance builder. This description applies for version 10.2.35.

3.1 Before you start

1. Install KIWI NG first, either via your distributions' package manager (see *Chapter 2, Installation*) or via:

```
$ sudo pip install kiwi
```

2. Clone the KIWI NG repository containing example appliances (see *Section 2.4, "Example Appliance Descriptions"*):

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

Note

In case the following procedure causes any trouble please take a look at the *Chapter 5, Troubleshooting* chapter and/or reach out at: *Section 3, "Contact"*

3.2 Choose a First Image

Find example appliance descriptions in the KIWI NG repository checkout as follows:

```
$ tree -L 3 kiwi/build-tests
```

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

3.3 Build your First Image

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

```
$ sudo kiwi-ng system build \
    --description kiwi/build-tests/x86/leap/test-image-disk \
    --set-repo https://download.opensuse.org/distribution/leap/15.6/repo/oss \
    --target-dir /tmp/myimage
```

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

If you don't wish to create a openSUSE Leap 15.6 image, substitute the folder following the `--description` option with another folder that contains the image description which you selected.

3.4 Run your Image

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

```
$ sudo qemu -boot c \
    -drive file=kiwi-test-image-disk.x86_64-1.15.6.raw,format=raw,if=virtio \
    -m 4096 -serial stdio
```

3.5 Tweak and Customize your Image

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

4 Working from the Command Line

Note

This document provides a list of the existing KIWI Next Generation (KIWI NG) commands for version 10.2.35.

4.1 kiwi-ng

4.1.1 SYNOPSIS

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

kiwi-ng -h | --help
kiwi-ng [--profile=<name>...]
        [--setenv=<variable=value>...]
        [--temp-dir=<directory>]
        [--type=<build_type>]
        [--logfile=<filename>]
        [--logsocket=<socketfile>]
        [--loglevel=<number>]
        [--debug]
        [--debug-run-scripts-in-screen]
        [--color-output]
        [--config=<configfile>]
        [--kiwi-file=<kiwifile>]
    image <command> [<args>...]
kiwi-ng [--logfile=<filename>]
        [--logsocket=<socketfile>]
        [--loglevel=<number>]
        [--debug]
        [--debug-run-scripts-in-screen]
        [--color-output]
        [--config=<configfile>]
    result <command> [<args>...]
kiwi-ng [--profile=<name>...]
        [--setenv=<variable=value>...]
        [--shared-cache-dir=<directory>]
        [--temp-dir=<directory>]
        [--target-arch=<name>]
        [--type=<build_type>]
```

```
        [--logfile=<filename>]
        [--logsocket=<socketfile>]
        [--loglevel=<number>]
        [--debug]
        [--debug-run-scripts-in-screen]
        [--color-output]
        [--config=<configfile>]
        [--kiwi-file=<kiwifile>]
    system <command> [<args>...]
kiwi-ng -v | --version
kiwi-ng help
```

4.1.2 DESCRIPTION

KIWI NG is an imaging solution that is based on an image XML description. A description can consist of a single `config.xml` or `.kiwi` file. It may also include additional files, such as scripts or configuration data.

A collection of example image descriptions can be found in the following GitHub repository: https://github.com/OSInside/kiwi-descriptions. Most of the descriptions provide a so-called appliance image. Appliance is a small, text-based image including a predefined remote source setup to allow installation of missing software components.

Although KIWI NG operates in two steps, the system build command combines both steps into one to make it easier to start with KIWI NG.

The first step is to prepare a directory that includes the contents of a new filesystem based on one or more software package sources. The second step uses the prepared contents of the new image root tree to create an output image.

KIWI NG supports the creation of the following image types:

- ISO Live Systems
- virtual disk for e.g cloud frameworks
- OEM expandable disk for system deployment from ISO or the network
- file system images for deployment in a PXE boot environment

Depending on the image type, different disk formats and architectures are supported.

4.1.3 GLOBAL OPTIONS

`--color-output`

Use escape sequences to print different types of information in colored output. for this option to work, the underlying terminal must support those escape characters. Error messages appear in red, warning messages in yellow, and debugging information is printed in light grey.

`--config`

Use specified runtime configuration file. If not specified, the runtime configuration is expected to be in the `~/.config/kiwi/config.yml` or `/etc/kiwi.yml` files.

`--debug`

Print debug information on the command line. Same as: `--loglevel 10`.

`--debug-run-scripts-in-screen`

Run scripts called by KIWI NG in a screen session.

`--logfile`

Specify log file. The logfile contains detailed information about the process. The special call: `--logfile stdout` sends all information to standard out instead of writing to a file.

`--logsocket`

Send log data to the specified Unix Domain socket in the same format as with `--logfile`.

`--loglevel`

Specify logging level as a number. Further info about the available log levels can be found at: https://docs.python.org/3/library/logging.html#logging-levels Setting a log level displays all messages above the specified level.

```
----------------------------
| Level    | Numeric value |
----------------------------
| CRITICAL | 50            |
| ERROR    | 40            |
| WARNING  | 30            |
| INFO     | 20            |
| DEBUG    | 10            |
| NOTSET   | 0             |
----------------------------
```

`--profile`

Select profile to use. The specified profile must be part of the XML description. The option can be specified multiple times to allow a combination of profiles.

`--setenv`

export environment variable and its value into the caller environment. This option can be specified multiple times

`--shared-cache-dir`

Specify an alternative shared cache directory. The directory is shared via bind mount between the build host and image root system, and it contains information about package repositories and their cache and meta data. The default location is `/var/cache/kiwi`.

`--temp-dir`

Specify an alternative base temporary directory. The provided path is used as base directory to store temporary files and directories. Default is `/var/tmp`.

`--target-arch`

Specify an image architecture. By default, the host architecture is used as the image architecture. If the specified architecture name does not match the host architecture (thus requesting a cross architecture image build), you must configure the support for the image architecture and binary format on the building host. This must be done during the preparation stage, and it is beyond the scope of KIWI NG.

`--type`

Select an image build type. The specified build type must be configured as part of the XML description.

`--kiwi-file`

Basename of kiwi file that contains the main image configuration elements. If not specified, kiwi uses a file named `config.xml` or a file matching `*.kiwi`

`--version`

Show program version

4.1.4 EXAMPLE

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

$ sudo kiwi-ng system build \
```

```
    --description kiwi/build-tests/x86/leap/test-image-disk \
    --set-repo https://download.opensuse.org/distribution/leap/15.6/repo/oss \
    --target-dir /tmp/myimage
```

4.2 kiwi-ng result list

4.2.1 SYNOPSIS

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

kiwi-ng result list -h | --help
kiwi-ng result list --target-dir=<directory>
kiwi-ng result list help
```

4.2.2 DESCRIPTION

List build results from a previous build or create command. During multiple image builds with the same target directory, the build result information is overwritten every time you build an image. This means that the build result list is valid for the last build only.

4.2.3 OPTIONS

`--target-dir`
: Directory containing the KIWI NG build results.

4.3 kiwi-ng result bundle

4.3.1 SYNOPSIS

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

kiwi-ng result bundle -h | --help
kiwi-ng result bundle --target-dir=<directory> --id=<bundle_id> --bundle-dir=<directory>
```

```
    [--bundle-format=<format>]
    [--zsync_source=<download_location>]
    [--package-as-rpm]
    [--no-compress]
kiwi-ng result bundle help
```

4.3.2 DESCRIPTION

Create a result bundle from the image build in the specified target directory. Each resulting image contains the specified bundle identifier as part of its filename. Uncompressed image files are also compressed as an XZ archive. An SHA checksum is generated for each resulting image.

4.3.3 OPTIONS

`--bundle-dir`

Directory containing the bundle results, compressed versions of image results, and SHA checksum files.

`--bundle-format`

Specify the bundle format to create the bundle. If provided, this setting will overwrite an eventually provided `bundle_format` attribute from the main image description. The format string can contain placeholders for the following elements:

- %N : Image name
- %P : Concatenated profile name (_)
- %A : Architecture name
- %I : Bundle ID
- %T : Image build type name
- %M : Image Major version number
- %m : Image Minor version number
- %p : Image Patch version number
- %v : Image Version string

`--id`

Bundle ID. It is a free-form text appended to the image version information as part of the result image filename.

`--target-dir`

Directory containing the KIWI NG build results.

`--zsync_source`

Download location of the bundle file or files. Only relevant if `zsync` is used to sync the bundle.

- The zsync control file is created for the bundle files marked for compression.
- All files in a bundle must be stored in the same download location.

`--package-as-rpm`

Create an RPM package containing the result files.

`--no-compress`

Do not compress the result image file(s). Note: Image files that were already produced as compressed variants stays compressed.

4.4 kiwi-ng system prepare

4.4.1 SYNOPSIS

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

kiwi-ng system prepare -h | --help
kiwi-ng system prepare --description=<directory> --root=<directory>
    [--allow-existing-root]
    [--clear-cache]
    [--ignore-repos]
    [--ignore-repos-used-for-build]
    [--set-repo=<source,type,alias,priority,imageinclude,package_gpgcheck,
{signing_keys},components,distribution,repo_gpgcheck,repo_sourcetype>]
    [--set-repo-credentials=<user:pass_or_filename>]
    [--add-repo=<source,type,alias,priority,imageinclude,package_gpgcheck,
{signing_keys},components,distribution,repo_gpgcheck,repo_sourcetype>...]
```

```
    [--add-repo-credentials=<user:pass_or_filename>...]
    [--add-package=<name>...]
    [--add-bootstrap-package=<name>...]
    [--ca-cert=<cert-file>...]
    [--ca-target-distribution=<suse|rhel|debian|archlinux>]
    [--delete-package=<name>...]
    [--set-container-derived-from=<uri>]
    [--set-container-tag=<name>]
    [--add-container-label=<label>...]
    [--set-type-attr=<attribute=value>...]
    [--set-release-version=<version>]
    [--signing-key=<key-file>...]
kiwi-ng system prepare help
```

4.4.2 DESCRIPTION

Create a new image root directory. The prepare step sets up a new image root directory from the specified XML description. The specified directory acts as a root directory of the new image root system. You can enter the system as root via chroot using the following command:

```
$ chroot <directory> bash
```

4.4.3 OPTIONS

`--add-bootstrap-package`

Specify a package to install as part of the early KIWI NG bootstrap phase. The option can be specified multiple times.

`--add-container-label`

Add a container label to the container configuration metadata. It overwrites the label with the provided key-value pair if it was already defined in the XML description.

`--add-package`

Specify a package to add (install). The option can be specified multiple times.

`--add-repo`

Add a new repository to the existing repository setup in the XML description. This option can be specified multiple times. For details about the supported option values, see the **–set-repo** information below

`--add-repo-credentials`

For **uri://user:pass@location** type repositories, set the user and password connected with an add-repo specification. If the provided value is a filename in the filesystem, the first line of the file is used as credentials.

`--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 validates the signature of the package from the original repository source for any build. Some package managers unconditionally trust the contents of the cache, which works for cache data dedicated to one build. In case of KIWI NG, the cache is shared between multiple image builds on the host for performance reasons.

`--ca-cert`

Add a cert-file to the directory storing additional local CA certificates. The import will occur immediately after the bootstrap process, where the required CA update tooling is expected to be installed. This option is useful for situations where certificates are not packaged, or the certificates are required during the build process, e.g. due to proxy servers in the build environment that need certificates in chroot. The option can be specified multiple times.

`--ca-target-distribution`

Specify target distribution for the import of certificates via the –ca-cert options(s) and/or the provided <certificates> from the image description. The selected distribution is used in KIWI to map the distribution specific CA storage path and update tool for the import process.

`--delete-package`

Specify a package to delete. The option can be specified multiple times.

`--description`

Path to the KIWI NG XML description. The directory must contain at least a config.xml of *.kiwi XML description.

`--ignore-repos`

Ignore all repository configurations in the XML description. This option is normally used in combination with the `--add-repo` option. Otherwise, an image build operation results in an error.

`--ignore-repos-used-for-build`

Works the same way as `--ignore-repos`,` but the repository configurations that have the imageonly attribute set to **true** are not ignored.

`--root`

Path to the new root system.

`--set-repo`

Overwrite the first repository entry in the XML description with the provided information:

- **source**
  Source URL pointing to a package repository that 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: `rpm-md`, `apt-deb`.
- **alias**
  An alias name for the repository. If not specified, KIWI NG generates an alias name based hex representation of uuid4. While the hex is used to uniquely identify the repository, it is not descriptive. We recommend to use a descriptive and unique alias name.
- **priority**
  A number indicating the repository priority. How the value is evaluated depends on the selected package manager. 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 the repository is part of the system image repository setup or not.
- **package_gpgcheck**
  Set to either **true** or **false** to specify if the repository must validate the package signatures.
  - **{signing_keys}**

  List of signing_keys enclosed in curly brackets and delimited by the semicolon. The reference to a signing key must be provided in the URI format.

- **components**
  Component list for Debian-based repos as a space-delimited string.

- **distribution**
  Main distribution name for Debian-based repos.

- **repo_gpgcheck**
  Set to either **true** or **false** to specify if the repository must validate the repository signature.

- **repo_sourcetype**
  Specify the source type of the repository path. Supported values are baseurl, metalink or mirrorlist. With baseurl the source path is interpreted as simple URI. If metalink is set the source path is resolved as metalink URI and if mirrorlist is set the source path is resolved as a mirrorlist file. If not specified, baseurl is the default

`--set-repo-credentials`
: For **uri://user:pass@location** type repositories, set the user and password connected to the set-repo specification. If the provided value is a filename in the filesystem, the first line of that file is used as credentials.

`--set-container-derived-from`
: Overwrite the source location of the base container for the selected image type. The setting is only effective if the configured image type is setup with an initial derived_from reference

`--set-container-tag`
: Overwrite the container tag in the container configuration. The setting applies only if the container configuration provides an initial tag value.

`--set-type-attr`
: Overwrite/set the attribute with the provided value in the selected build type section. Example: `--set-type-attr volid=some`

`--set-release-version`
: Overwrite/set the release-version element in the selected build type preferences section

`--signing-key`
: 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 validates repository and package signatures during build time. This option can be specified multiple times.

4.5 kiwi-ng system update

4.5.1 SYNOPSIS

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

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

4.5.2 DESCRIPTION

Update a previously prepare image root tree. The update command refreshes the contents of the root directory according to the repository setup of the image XML description. The update command can also be used to add or remove packages from the image root tree.

4.5.3 OPTIONS

`--add-package`
: Specify a package to add (install). The option can be specified multiple times.

`--delete-package`
: Specify a package to delete. The option can be specified multiple times.

`--root`
: Path to the root directory of the image.

4.6 kiwi-ng system build

4.6.1 SYNOPSIS

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

kiwi-ng system build -h | --help
```

```
kiwi-ng system build --description=<directory> --target-dir=<directory>
    [--allow-existing-root]
    [--clear-cache]
    [--ignore-repos]
    [--ignore-repos-used-for-build]
    [--set-repo=<source,type,alias,priority,imageinclude,package_gpgcheck,
{signing_keys},components,distribution,repo_gpgcheck,repo_sourcetype>]
    [--set-repo-credentials=<user:pass_or_filename>]
    [--add-repo=<source,type,alias,priority,imageinclude,package_gpgcheck,
{signing_keys},components,distribution,repo_gpgcheck,repo_sourcetype>...]
    [--add-repo-credentials=<user:pass_or_filename>...]
    [--add-package=<name>...]
    [--add-bootstrap-package=<name>...]
    [--ca-cert=<cert-file>...]
    [--ca-target-distribution=<suse|rhel|debian|archlinux>]
    [--delete-package=<name>...]
    [--set-container-derived-from=<uri>]
    [--set-container-tag=<name>]
    [--add-container-label=<label>...]
    [--set-type-attr=<attribute=value>...]
    [--set-release-version=<version>]
    [--signing-key=<key-file>...]
kiwi-ng system build help
```

4.6.2 DESCRIPTION

Build an image in one step. The build command combines preparation and building steps, which makes it possible to create an image with a single command. The build command creates the root directory of the image under `<target-dir>/build/image-root` and writes a log file `<target-dir>/build/image-root.log`. The result image files are created in the specified target directory.

4.6.3 OPTIONS

`--add-bootstrap-package`

Specify package to install as part of the early KIWI NG bootstrap phase. The option can be specified multiple times.

`--add-container-label`

Add a container label in the container configuration metadata. It overwrites the label with the provided key-value pair if it is already defined in the XML description.

`--add-package`

Specify package to add (install). The option can be specified multiple times.

`--add-repo`

Add a new repository to the existing repository setup in the XML description. This option can be specified multiple times. For details about the possible option values see the **–set-repo** information below.

`--add-repo-credentials`

For **uri://user:pass@location** repositories, set the user and password associated with an add-repo specification. If the provided value describes a filename in the filesystem, the first line of that file is used as credentials.

`--allow-existing-root`

Use an existing root directory from a previous build attempt. Use with caution, because this can cause an inconsistent root tree if the existing contents does not fit to the previous image type setup.

`--clear-cache`

Delete repository cache for each of the used repositories before installing any package. This is useful when an image build validates the signature of the package from the original repository source for any build. Some package managers unconditionally trust the contents of the cache, which works for cache data dedicated to one build. In case of KIWI NG, the cache is shared between multiple image builds on that host for performance reasons.

`--ca-cert`

Add a cert-file to the directory storing additional local CA certificates. The import will occur immediately after the bootstrap process, where the required CA update tooling is expected to be installed. This option is useful for situations where certificates are not packaged, or the certificates are required during the build process, e.g. due to proxy servers in the build environment that need certificates in chroot. The option can be specified multiple times.

`--ca-target-distribution`

Specify target distribution for the import of certificates via the –ca-cert options(s) and/or the provided <certificates> from the image description. The selected distribution is used in KIWI to map the distribution specific CA storage path and update tool for the import process.

`--delete-package`

Specify package to delete. The option can be specified multiple times.

`--description`

Path to an 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. This option is used in combination with the `--add-repo` option. Otherwise, there are no repositories available for an image build, which leads to an error.

`--ignore-repos-used-for-build`

Works the same way as `--ignore-repos`, except that repository configuration with the imageonly attribute set to **true** is not ignored.

`--set-repo`

Overwrite the first repository entry in the XML description with the provided information:

- **source**
  Source URL pointing to a package repository that 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: `rpm-md`, `apt-deb`.

- **alias**
  An alias name for the repository. If not specified, KIWI NG generates an alias name as result of hex representation from uuid4. While the hex is used to uniquely identify the repository, it is not descriptive. We recommend using descriptive aliases.

- **priority**
  A number indicating the repository priority. How the value is evaluated depends on the selected package manager. Refer to the package manager documentation for details about the supported priority ranges and their meaning.

- **imageinclude**
  Set to either **true** or **false** to indicate if the repository is be part of the system image repository setup or not.

- **package_gpgcheck**
  Set to either **true** or **false** to indicate if the repository should validate the package signatures.

- **{signing_keys}**
  List of signing_keys enclosed in curly brackets and delimited by the semicolon. The reference to a signing key must be provided in the URI format.

- **components**
  Component list for Debian-based repos as space-delimited string.

- **distribution**
  Main distribution name for Debian-based repos.

- **repo_gpgcheck**
  Set to either **true** or **false** to indicate if the repository should validate the repository signature.

- **repo_sourcetype**
  Specify the source type of the repository path. Supported values are baseurl, metalink or mirrorlist. With baseurl the source path is interpreted as simple URI. If metalink is set the source path is resolved as metalink URI and if mirrorlist is set the source path is resolved as a mirrorlist file. If not specified, baseurl is the default

`--set-repo-credentials`
: For **uri://user:pass@location** type repositories, set the user and password connected to the set-repo specification. If the provided value describes a filename in the filesystem, the first line of that file is used as credentials.

`--set-container-derived-from`
: Overwrite the source location of the base container for the selected image type. The setting applies only if the configured image type is setup with an initial derived_from reference.

`--set-container-tag`
: Overwrite the container tag in the container configuration. The setting is only effective if the container configuration provides the initial tag value.

`--set-type-attr`
: Overwrite/set the attribute with the provided value in the selected build type section. Example: `--set-type-attr volid=some`

`--set-release-version`

Overwrite/set the release-version element in the selected build type preferences section

`--signing-key`

Set the key file to be trusted and imported into the package manager database before performing any operation. This is useful when an image build validates repository and package signatures during build time. This option can be specified multiple times.

`--target-dir`

Path to store the build results.

4.6.4 URI_TYPES

- **http://** | **https://** | **ftp://**
  Remote repository delivered via the 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.
- **dir://**
  Local directory.

4.7 kiwi-ng system create

4.7.1 SYNOPSIS

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

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

```
kiwi-ng system create help
```

4.7.2 DESCRIPTION

Create an image from the previously prepared image root directory. The `kiwi create` command is normally issued after the `kiwi prepare` command, and it and builds the requested image type in the specified target directory.

4.7.3 OPTIONS

`--root`
: Path to the image root directory. This directory is normally created by the `kiwi prepare` command. Keep in mind that if the specified directory is not created using the `kiwi prepare` command, KIWI NG stores image build metadata in the image/ directory. This directory must exist for the `kiwi create` command to work correctly.

`--target-dir`
: Path to store the build results.

`--signing-key`
: Trusted key file to be imported into the package manager database before performing any operation. This is useful if an image build validates repository and package signatures during build time. In the create step, this option only affects the boot image. This option can be specified multiple times.

4.8 kiwi-ng image resize

4.8.1 SYNOPSIS

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

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

4.8.2 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. The OEM boot code in KIWI NG offers a resizing procedure that can be used to make use of the additional free space. For OEM image builds, it is advisable to run the resizing operation.

4.8.3 OPTIONS

`--root`

The path to the root directory. If not specified, kiwi searches the root directory in build/image-root under the specified target directory.

`--size`

New size of the image. The value is either a size in bytes, or it can be specified with m (MB) or g (GB). Example: 20g

`--target-dir`

Directory containing the kiwi build results.

4.9 kiwi-ng image info

4.9.1 SYNOPSIS

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

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

4.9.2 DESCRIPTION

Provides information about the specified image description. If no specific info option is provided, the command lists basic information about the image. This information is also available in the image XML description file. Specifying an extension option like `resolve-package-list` makes a dependency resolver to run through the list of packages, providing more detailed information about the image description.

4.9.3 OPTIONS

`--add-repo`

Add repository with given source, type, alias and priority.

`--description`

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. This option is usually used together with the –add-repo option. Otherwise there are no repositories available for the processing the requested image information, which could lead to an error.

`--list-profiles`

list profiles available for the selected/default type.

NOTE: If the image description is designed in a way that there is no default build type configured and/or the main build type is also profiled, it's required to provide this information to kiwi to list further profiles for this type. For example: kiwi-ng –profile top_level_entry_profile image info …

`--print-kiwi-env`

print kiwi profile environment variables. The listed variables are available in the shell environment of the kiwi hook scripts.

NOTE: The kiwi profile environment grows during the build process. When used in early stages e.g. in a post_bootstrap.sh script it can happen that not all variables have a value. The setup of the kiwi profile environment in the image info output can therefore also only list the static configuration values which are known at the beginning of a build process.

`--resolve-package-list`

Solve package dependencies and return a list of all packages including their attributes, for example size, shasum, and more.

`--print-xml`

Print image description in the XML format. The specified image description is converted to XML and sent to the XSLT stylesheet processor. The result is then validated using the RelaxNG schema and the schematron rules. The command is normally used to convert an old image description to the latest schema.

`--print-yaml`

Behaves similar to `--print-xml`, but after validation, the result is converted to the YAML format. The command can be used for different operations:

- Conversion of the specified image description from or into different formats. This requires the `anymarkup` Python module to be installed. The module is not a hard requirement and loaded on demand. If the module is missing, requests to convert to other format than XML fail.
- Update of an old image description to the latest schema

5 Troubleshooting

Note

Abstract

This document describes situations which leads to issues during build or boot time of the image build with KIWI NG. The suggested solutions are considered best practice but are just one out of other possible solution candidates.

5.1 Build Host Constraints

For building images a host system is required that runs the build process. Tools to create the image are used from that host and this creates an indirect dependency to the target image. For example; Building an Ubuntu image requires the apt and dpkg tools and metadata to be available and functional on the host to build an Ubuntu image. There are many more of those host vs. image dependencies and not all of them can be resolved in a clear and clean way.

The most compatible environment is provided if the build host is of the same distribution than the target image. In other cases our recommendation is that the build host is of the same distribution than the target and near to the major version (+-1) compared to the target. Such an environment can be found in:

- The Open Build Service OBS (https://build.opensuse.org).
- The KIWI NG boxed plugin: *Section 6.1, "Building in a Self-Contained Environment"*

In general, our goal is to support any major distribution with KIWI NG. However for building images we rely on core tools which are not under our control. Also several design aspects of distributions like **secure boot** and working with **upstream projects** are different and not influenced by us. There are many side effects that can be annoying especially if the build host is not of the same distribution vendor than the image target.

5.1.1 Package Manager Behavior

One of the key requirements for KIWI NG is that the target distribution consists out of a collection of software packages provided through software repositories. As of today this applies to every Linux distribution in some shape or form. For the installation of this software packages there are tools available called `package manager` and KIWI NG implements an API for a variety of package managers to support the different Linux distributions.

In KIWI NG the package manager tools are called in a way that allows for a non interactive installation of the packages specified in the image description. However, KIWI NG does intentionally not configure other features of the package manager to keep their distribution default behavior as much as possible. There are many options that can be set for a package manager to behave differently. This part of the troubleshooting exists to inform about the most common surprises or unexpected behavior of certain package managers and solutions how to address them.

Note

All of the following information applies to package manager calls performed inside of the image root tree. Meaning `after` the image bootstrap phase. During the image bootstrap phase all package manager calls happens on the build host and their behavior can only be influenced by changing the setup of the build host. An isolated build environment is required to address issues in this area. See *Section 6.1, "Building in a Self-Contained Environment"* for details.

5.1.1.1 zypper: modalias matching

A module alias is an identifier that exists if the respective hardware was found by the kernel on this system. The package manager can use this information to match package(s) that holds a reference to the alias name and installs it automatically. For example: Automatically install a driver package if the respective hardware is present. In zypper this behavior is by default enabled. When building an image this feature can be unwanted and lead to interesting side effects. As it's required to build the image on some host, there is also some hardware available during build. This must not necessarily be the same or compatible hardware the image is expected to run on later. For example: The buildhost has a nvidia graphics card. An image build process

would install the nvidia driver package due to the modalias match. The image is expected to run on a system without a nvidia card. The build process would install unneeded software. To switch off modalias matching in zypper follow these steps:

1. Add a `post_bootstrap.sh` hook script to your image description with the following code:

   ```
   echo 'ZYPP_MODALIAS_SYSFS="" > .kiwi.package_manager.env
   ```

2. Optionally add a `config.sh` hook script to your image description with the following code:

   ```
   rm .kiwi.package_manager.env
   ```

   This will delete the custom environment file such that it does not appear in the final image

3. Rebuild your image. The `kiwi.package_manager.env` environment file is used by kiwi and added to the execution environment of the package manager. The `ZYPP_MODALIAS_SYSFS` variable can be used to switch off the modalias matching. For more details on zypper setting please refer to: https://doc.opensuse.org/projects/libzypp/HEAD/zypp-envars.html

5.2 Architectures

With regards to the information in *Section 5.1, "Build Host Constraints"* one requirement between the build host and the image when it comes to architecture support is, that the image architecture should match the build host architecture. Cross arch building would require any core tool that is used to build an image to be cross arch capable.

To patch e.g an x86_64 system such that it can build an aarch64 image would require some work on binutils and hacks as well as performance tweaks which is all not worth the effort and still can lead to broken results. Thus we recommend to provide native systems for the target architecture and build there. One possible alternative is to use the kiwi boxed plugin as mentioned above together with a box created for the desired architecture. However keep in mind the performance problematic when running a VM of a different architecture.

The majority of the image builds are based on the x86 architecture. As mentioned KIWI NG also supports other architectures, shown in the table below:

Architecture	Supported
x86_64	yes
ix86	yes **note:distro**
s390	yes **note:distro**
s390x	yes **note:distro**
armv6	yes **note:distro**
armv7	yes **note:distro**
aarch64	yes **note:distro**
ppc64	yes **note:distro**

5.3 Host Security Settings Conflicts with KIWI

Note

Abstract

This page provides further information how to solve image build problems caused by `selinux` security policies.

Linux systems are protected against write/read or other operations depending on the application which wants to access or modify data. The rules for this protection are provided in security policies. There are several applications enforcing these security settings, e.g `apparmor` or `selinux`. In this troubleshooting chapter the focus is set on `selinux`

Protecting files, process groups, kernel filesystems, device nodes and more from unauthorized access and restrict it to a certain set of applications is a nice concept. However, if taken serious no other application except the ones configured in the security policy will function properly.

When building an appliance, the appliance builder has to have access to a wide range of services. It must be able to create a new package database elsewhere in the system. It must be able to create, read and write device nodes, create filesystems, partitions, bootloader configurations etc etc. The list is very long and no security policy could cover this in a way that it would not be open to everything which in the end leads to a pointless exercise and no security at all.

This means for users who would like to keep the security settings of the system enforced and unchanged, the only way to allow KIWI NG to do its job is to run it through `boxbuild` as explained in *Section 6.1, "Building in a Self-Contained Environment"*

For users who can afford to open the system security policy, the following procedure will make KIWI NG to work:

```
sudo setenforce 0
```

This action disables selinux temporary. To disable selinux permanently perform the following steps:

1. Open the SELinux configuration file: `/etc/selinux/config`
2. Locate the following line: `SELINUX=enforcing`
3. Change the value to disabled:

   ```
   SELINUX=disabled
   ```

4. On the next reboot, SELinux is permanently disabled.

Note

similar instructions applies to other application security subsystems like `apparmor`. Due to the complexity of these systems this article just mentions the most common issue people run into when building images on systems protected through `selinux`.

5.4 Incompatible Filesystem Settings on Host vs. Image

Note

Abstract

This page provides further information how to solve image boot problems if the filesystem tool chain on the image build host is incompatible with the image target distribution

When KIWI NG builds an image which requests the creation of a filesystem, the required filesystem creation tool, for example `mkfs.xfs`, is called from the host on which KIWI NG gets called. It is expected that the generated filesystem is compatible with the image target distribution. This expectation is not always correct and depends on the compatibility of the filesystem default settings between build host and image target. We know about the following settings that causes an incompatible filesystem which will not be able to be used on boot:

Ext[2,3,4]

Check `/etc/mke2fs.conf` on the build host and make sure the configured `inode_size` is the same as the setting used for the target image. To solve an issue of this type use the following filesystem creation option in your KIWI NG image configuration:

```
<type fscreateoptions="-I inode-size"/>
```

XFS

Check the XFS metadata setup on the build host and make sure the settings are compatible with the target image. XFS has the default settings compiled in, thus it might be needed to build the image first and use the `xfs_info` tool in a `disk.sh` script to fetch the settings at build time of the image. We know from community reports that the setting `sparse=1` will cause issues on older versions of grub's xfs module, which does not know how to handle this setting properly. To solve an issue of this type use the following filesystem creation option in your KIWI NG image configuration:

```
<type fscreateoptions="-i sparse=0"/>
```

btrfs

btrfs and default page sizes (4k vs 64k). By default btrfs autodetects the sectorsize according to the used kernel page size. If the sectorsize differs from the page size, the created filesystem can not be mounted by the image target kernel. If there is a different kernel

page size between the kernel on the system the image is build on and the later kernel used for the image, it's required to specify the filesystem sector size to match with the kernel page size of the kernel used for the image. This can be done like in the following example:

```
<type fscreateoptions="--sectorsize 4k"/>
```

Note

There can be more inconsistencies in the area of filesystems which we haven't listed here. In general it's advisable to build the image in a compatible environment. At best the build host distribution is of the same major Linux version than the image target. For this purpose KIWI NG provides the so called `boxed-plugin`. Further details can be found in *Section 6.1, "Building in a Self-Contained Environment"*

5.5 Boxbuild Tweaks

Note

Abstract

This document describes a few ways to modify box build VMs for testing/debugging.

5.5.1 Increase Box Build Image Size

In particularly large builds, you may find that the upstream build boxes aren't quite large enough, and fail to build during the final few steps. While it is a bit of a kludge, it is possible to increase the size of the build box.

To do so, follow these steps:

Note

For this example, we will assume the box increase question is an Ubuntu box, located in ~/.kiwi_boxes/ubuntu

- While the VM is offline, locate the VM you want to modify and resize with `qemu-img`. Here we will increase the size by 20G. The VM will have to be told to utilize this space in the following steps.

```
$ qemu-img resize Ubuntu-Box.x86_64-1.22.04-System-BuildBox.qcow2 +20G
```

- When relaunching your `kiwi-ng` box build, make sure you use `--no-snapshot` and `--box-debug` options within your build command/script. Example:

```
$ kiwi --debug --profile="Disk" --type oem system boxbuild --no-snapshot \
--box-memory=32G --box-smp-cpus=16 --box-debug --box ubuntu -- \
--description ./ubuntu-jammy --target-dir /build/kiwi/outputs/
```

- When the build fails and drops you into the VM console, you will need to extend the partition of the VM rootfs, then resize with `resize2fs`. In this example, `parted` was used and the partition in question was /dev/vda3.

```
$ parted
# Can run parted print to check for relevant partitions if needed.
(parted) $ print
(parted) $ resizepart 3 100%
# Exit from parted
(parted) $ quit
# Run resize2fs to grow the filesystem to fill the space
$ resize2fs /dev/vda3
```

- From this point, depending on where your build failed, it may be possible to continue your build from inside the box, using the existing 9p mount points defined by your build command. Using the command above as an example, `/result` within the box maps up to `/build/kiwi/outputs` on the host, and it's possible to run

```
$ kiwi-ng --profile="Disk" --type oem  system create \
--root=/result/build/image-root/ --target-dir=/result
```

- If the rebuild from within was successful, you can copy the files from `/result` to `/bundle`, from within the VM, where `/bundle` maps to your `target-dir` on the host.

6 KIWI Plugins

Note

This document provides a list of the existing KIWI Next Generation (KIWI NG) plugins which provides extended functionality for version 10.2.35.

6.1 Building in a Self-Contained Environment

Note

Abstract

Users building images with KIWI NG face problems if they want to build an image matching one of the following criteria:

- build should happen as non root user.
- build should happen on a host system distribution for which no KIWI NG packages exists.
- build happens on an incompatible host system distribution compared to the target image distribution. For example building an apt/dpkg based system on an rpm based system.
- run more than one build process at the same time on the same host.
- run a build process for a different target architecture compared to the host architecture (Cross Arch Image Build)

This document describes how to perform the build process in a self contained environment using fast booting virtual machines to address the issues listed above.

The changes on the machine to become a build host will be reduced to the requirements of the KIWI NG `boxed plugin`

6.1.1 Requirements

Add the KIWI NG repo from the Open Build Service. For details see *Section 2.1, "Installation from OBS"*. The following KIWI NG plugin needs to be installed on the build system:

```
$ sudo zypper in python3-kiwi_boxed_plugin
```

6.1.2 Building with the boxbuild command

The installation of the KIWI NG boxed plugin has registered a new kiwi command named `boxbuild`. The command implementation uses KVM as virtualization technology and runs the KIWI NG `build` command inside of a KVM controlled virtual machine. For running the build process in a virtual machine it's required to provide VM images that are suitable to perform this job. We call the VM images `boxes` and they contain kiwi itself as well as all other components needed to build appliances. Those boxes are hosted in the Open Build Service and are publicly available at the `Subprojects` tab in the: Virtualization:Appliances:SelfContained (https://build.opensuse.org/project/show/Virtualization:Appliances:SelfContained) project.

As a user you don't need to work with the boxes because this is all done by the plugin and provided as a service by the KIWI NG team. The `boxbuild` command knows where to fetch the box and also cares for an update of the box when it has changed.

Building an image with the `boxbuild` command is similar to building with the `build` command. The plugin validates the given command call with the capabilities of the `build` command. Thus one part of the `boxbuild` command is exactly the same as with the `build` command. The separation between `boxbuild` and `build` options is done using the `--` separator. The following example shows how to build an example from the `kiwi-descriptions` repo:

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

$ kiwi-ng --profile Virtual system boxbuild --box leap -- \
      --description kiwi-descriptions/suse/x86_64/suse-leap-15.6 \
      --target-dir /tmp/myimage
```

Note

The provided `--description` and `--target-dir` options are setup as shared folders between the host and the box. No other data will be shared with the host.

6.1.3 Sharing Backends

As mentioned above, the `boxbuild` call shares the two host directories provided in `--description` and `--target-dir` with the box. To do this the following sharing backends are supported:

`--9p-sharing`

With QEMU's `9pfs` you can create virtual filesystem devices (virtio-9p-device) and expose them to the box. For more information see 9pfs (https://wiki.qemu.org/Documentation/9psetup). Using this sharing backend does not require any setup procedure from the user and is also the default for `boxbuild`

`--sshfs-sharing`

SSHFS is a FUSE-based filesystem client for mounting remote directories over a Secure Shell connection (SSH). In `boxbuild` this is used to mount directories from the host into the box. Because this runs through an SSH connection the host must allow connections from the box. If you plan to use `sshfs` add the following SSH public key to the `~/.ssh/authorized_keys` file of the user which is expected to call `boxbuild`

```
echo "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCtiqDaYgEMkr7za7qc4iPXftgu/
j3sodPOtpoG8PinwRX6/3xZteOJzCH2qCZjEgA5zsP9lxy/119cWXvdxFUvyEINjH77unzRnaHj/
yTXPhHuhHgAiEubuHer2gZoOs+UH4cGJLKCrabjTjZdeK9KvL
+hoAgJaWxDUvGsXYDQTBHXlKjniOL1MGbltDBHnYhu4k+PjjJ+UEBN
+8+F74Y5fYgIovXXY88WQrybuEr1eAYjhvk/ln6TKw1P6uvVMuIbAGUgnZFntDCI91Qw8ps1j
+lX3vNc8ZBoOwM6nHZqq4FAqbXuH+NvQFS/xDM6wwZQhAe+14dTQBA5F1mgCVf+fSbteb0/
CraSGmgKIM8aPnK8rfF+BY6Jar3AJFKVRPshRzrQj6CWYu3BfmOLupCpqOK2XFyoU2lEpaZDejgPSJq/
IBGZdjKplWJFF8ZRQ01a8eX8K2fjrQt/4k9c7Pjlg1aDH8Sf+5+vcehlSNs1d50wnFoaIPrgDdy04omiaJ8=
 kiwi@boxbuild" >> ~/.ssh/authorized_keys
```

The public key mentioned here is associated with an SSH key pair we provide in the prebuilt box images.

Warning

If the `sshfs` backend is used without the host trusting the box, the `boxbuild` call will become interactive at the time of the sshfs mount. In this case the user might be asked for a passphrase or depending on the host `sshd` setup the request will be declined and the boxbuild fails.

`--virtiofs-sharing`

QEMU virtio-fs shared file system daemon. Share a host directory tree with a box through a virtio-fs device. For more information see virtiofs (https://manpages.ubuntu.com/manpages/jammy/en/man1/virtiofsd.1.html). Using this sharing backend does not require any setup procedure from the user

Warning

virtiofs support was added but considered experimental and not yet stable across the distributions. Feedback welcome.

6.1.4 Building in Container

By default and also as preferred method, boxbuild runs in a KVM virtual machine. However, support for building in container instances via `podman` is also implemented. The boxes built as virtual machine images and also as OCI containers hosted on the openSUSE registry. In container mode boxbuild pulls the requested box from the remote registry to the local registry and starts a container instance. The following example shows how to build the mentioned KIWI NG integration test image in a container:

```
$ kiwi-ng --profile Virtual system boxbuild --container --box leap -- \
      --description kiwi-descriptions/suse/x86_64/suse-leap-15.6 \
      --target-dir /tmp/myimage
```

Note

The provided `--description` and `--target-dir` options are setup as shared volumes between the host and the box container. In addition containers also shares the kiwi cache from `/var/cache/kiwi` with the host for better performance.

Warning

For building in a container several runtime constraints exists and the isolation model is not as strict as it is when building in a VM. Please read the following information to get clarity on the existing constraints.

`loop devices`

As kiwi requires loop devices and calls other operations which requires root privileges, `podman` is started through `sudo`. As podman runs daemonless, the calling user must have the privileges to perform the needed kiwi actions. Non privileged builds are therefore not possible in container mode with kiwi.

`linux capabilities`

Several linux capabilities as they can be found in https://man7.org/linux/man-pages/man7/capabilities.7.html are set when `boxbuild` starts the container:

- AUDIT_WRITE
- AUDIT_CONTROL
- CAP_MKNOD
- CAP_SYS_ADMIN

`host device nodes`

The host device nodes from `/dev` are shared with the container instance. This is required to work with loop devices inside of the container. Device isolation is therefore not possible in container mode with kiwi. The loop device handling for container based builds, restricts the number of simultaneous kiwi build processes on this host to the number of available loop device nodes exposed from the host kernel.

6.2 Building based on Containers

Note

Abstract

When building images exposes one of the following requirements the stackbuild plugin provides an opportunity to address it:

- Preserve the image rootfs for a later rebuild without requiring the original software repositories.
- Build an image based on an existing container.

- Build an image based on a container stack.
- Transform a container into a KIWI NG image type

6.2.1 Installation

Add the KIWI NG repo from the Open Build Service. For details see *Section 2.1, "Installation from OBS"*. The following KIWI NG plugin needs to be installed on the build system:

```
$ sudo zypper in python3-kiwi_stackbuild_plugin
```

6.2.2 Concept

The design of the stackbuild plugin is two fold:

First the plugin comes with a command called `stash` which allows to store a kiwi built root tree as an OCI container. OCI stands for *Open Container Interface* and is a defacto standard format in the container world. Once the container got created it can be managed using the preferred container toolchain. The plugin code itself uses `podman` to work with containers.

As a next step and with the root tree as a container the plugin offers the opportunity to build images based on one ore more containers. That's also the reason why the plugin is called *stackbuild* as it allows you to stack different root containers together. Consequently the other command provided is named `stackbuild`.

The `stash` and `stackbuild` commands can be used independently from each other. If there is already a registry with containers that should be used to build images from, `stackbuild` can directly consume them.

This concept leads to a number of use cases and a few of them were picked and put into the abstract of this article. For the purpose of documenting the functionality of the plugin only a part of the possibilities are taken into account as follows:

6.2.3 Create a stash

The `stash` command creates an OCI compliant container from a given KIWI Next Generation (KIWI NG) image root tree and registers it in the local container registry. From there a user can push it to any registry of choice.

The following example creates a stash of a Tumbleweed build and illustrates how to register it in a foreign container registry:

```
# Build some image...
$ git clone https://github.com/OSInside/kiwi.git
$ sudo kiwi-ng system build \
    --description kiwi/build-tests/x86/tumbleweed/test-image-MicroOS/ \
    --set-repo http://download.opensuse.org/tumbleweed/repo/oss \
    --target-dir /tmp/myTWToday

# Stash the image root into a container
$ sudo kiwi-ng system stash \
    --root /tmp/myTWToday/build/image-root \
    --container-name twmos-snapshot

# Register the stash in a registry
$ podman login
$ podman push twmos-20211008 \
    docker://docker.io/.../twmos-snapshot:2021-10-08
```

If the `stash` command is called multiple times with the same container-name this leads to a new layer in the container for each call. To inspect the number of layers added to the container the following command can be used:

```
$ podman inspect twmos-snapshot
```

To list all stashes created by the `stash` command the following command can be used

```
$ kiwi-ng system stash --list
```

6.2.4 Rebuild from a stash

The `stackbuild` command takes the given container(s) from the local or remote registry and uses it/them to either rebuild an image from that data or build a new image on top of that data. If multiple containers are given the `stackbuild` command stacks them together in the order as they were provided.

Note

When using multiple containers the result stack root tree is created from a sequence of rsync commands into the same target directory. The stackbuild plugin does this with any container content given and does not check, validate or guarantee that the selection of

containers are actually stackable or leads to an usable root tree. This means it's in the responsibility of the caller to make sure the provided containers can actually be stacked together in the given order.

To simply rebuild the image from the stash created in Create a stash (#stash) call `stackbuild` as follows:

```
# Delete the image
$ sudo rm -rf /tmp/myTWToday

# Rebuild image from stash
$ sudo kiwi-ng system stackbuild \
    --stash twmos-snapshot:2021-10-08 \
    --target-dir /tmp/myTWToday
```

This rebuilds the image from the stash and the KIWI NG configuration inside of the stash. As all rootfs data is already in the stash, the command will not need external resources to rebuild the image.

6.2.5 Turn a container into a VM image

Another use case for the `stackbuild` plugin is the transformation of container images into another image type that is supported by KIWI NG. The following example demonstrates how an existing container image from the openSUSE registry can be turned into a virtual machine image.

When moving a container into a virtual machine image the following aspects has to be taken into account:

1. A container image usually has no kernel installed.
2. A container image usually has no bootloader installed.
3. A container image usually has no user configured.

For a VM image the mentioned aspects are mandatory. Therefore the following KIWI NG image description contains this additional information which the container cannot provide: Create the KIWI NG description as follows:

```
$ mkdir container_to_VM_layer
$ vi container_to_VM_layer/config.kiwi
```

And place the following content:

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

<image schemaversion="8.0" name="Leap-VM">
    <description type="system">
        <author>The Author</author>
        <contact>user@example.org</contact>
        <specification>
            Leap Container as VM
        </specification>
    </description>
    <preferences>
        <type image="oem" filesystem="xfs" firmware="uefi">
            <oemconfig>
                <oem-resize>false</oem-resize>
            </oemconfig>
        </type>
        <version>1.99.1</version>
        <packagemanager>zypper</packagemanager>
        <locale>en_US</locale>
        <keytable>us</keytable>
        <timezone>UTC</timezone>
    </preferences>
    <repository type="rpm-md" alias="Leap">
        <source path="https://download.opensuse.org/distribution/leap/15.6/repo/oss"/>
    </repository>
    <packages type="image">
        <package name="grub2"/>
        <package name="grub2-x86_64-efi" arch="x86_64"/>
        <package name="grub2-i386-pc"/>
        <package name="shim"/>
        <package name="kernel-default"/>
    </packages>
    <users>
        <user password="$1$wYJUgpM5$RXMMeASDc035eX.NbYWFl0" home="/root" name="root"
 groups="root"/>
    </users>
</image>
```

To build the virtual machine image from the current hosted Leap 15.3 container at SUSE, call the following `stackbuild` command:

```
$ sudo kiwi-ng system stackbuild \
    --stash leap:15.6 \
    --from-registry registry.opensuse.org/opensuse \
    --target-dir /tmp/myLeap \
```

```
    --description container_to_VM_layer
```

The resulting virtual machine image can be booted as follows:

```
$ qemu-kvm Leap-VM.x86_64-1.99.1.raw
```

7 Concept and Workflow

Note

Abstract

The following sections describe the concept and general workflow of building appliances with KIWI NG 10.2.35.

7.1 Host Requirements To Build Images

Building OS images requires several tools and sub-systems to be present on the host KIWI NG host. For example, to build a virtual disk image, tools for partition table setup or tools to create filesystems must to be available on the host that builds the image.

The number of required components depends on the selected image type and the features used with the image. It's unreasonable to expect KIWI NG users to know which exact components are needed to build the image. A mechanism called `kiwi-systemdeps` is designed to handle the host requirements.

`kiwi-systemdeps` consists out of a collection of sub-packages provided with the `python-kiwi` main package. Each individual package requires a number of tools and subsystem packages that belongs to the package category. There are the following systemdeps packages:

`kiwi-systemdeps-core`:

- Supports building the simple root archive `tbz` image type.
- Installs the package managers which are supported by the target distribution as well as the `tar` archiving tool.

`kiwi-systemdeps-containers`:

- Supports building `OCI` image types used with `docker`, `podman`.
- Installs the distribution specific tool chain to build OCI compliant container images.

`kiwi-systemdeps-containers-wsl`:

- Supports building `appx` and `wsl` image types. The `wsl` type is the successor for the former `appx` type
- Installs the distribution specific tool chain to build WSL compliant container images on Windows systems.

`kiwi-systemdeps-iso-media`:

- Supports building `iso` image types and `oem` install media.
- Installs all tools required to build ISO filesystems.
- Depends on the `-core`, `-filesystems` and `-bootloaders` kiwi-systemdeps packages.

`kiwi-systemdeps-bootloaders`:

- Supports building bootable `oem` and `iso` image types.
- Installs all bootloader tools depending on the host architecture to allow setup and install of the bootloader. The pulled in components are required for any image that is able to boot through some BIOS or firmware.
- Depends on the `-core` kiwi-systemdeps packages.

`kiwi-systemdeps-filesystems`:

- Supports building `fs-type`, `oem`, `pxe`, `kis` and live `iso` image types.
- Installs all tools to create filesystems supported by KIWI NG. The pulled in components are needed for any image type that needs to create a filesystem. This excludes the archive-based image types like `docker`, `appx` or `tbz`. The package also installs tools one level below the actual filesystem creation toolkit. These are components to manage loop devices as well as partition table setup and subsystem support like LVM and LUKS.
- Depends on the `-core` kiwi-systemdeps packages.

`kiwi-systemdeps-disk-images`:

- Supports building the `oem` image type.
- Installs all tools to create virtual disks. Virtual disks in KIWI NG are created using the QEMU toolchain.
- Depends on the `-filesystems` and `-bootloaders` kiwi-systemdeps packages.

`kiwi-systemdeps-image-validation`:

- Installs the `jing` tool to validate the image description. This is useful for detailed error reports from KIWI NG if an image description validation error occurs. In addition, the `anymarkup` Python module is installed if the the option to install recommended packages is set. With `anymarkup` available, KIWI NG can also handle image descriptions in another format than the XML markup (for example,YAML).

Depending on the image type the `kiwi-systemdeps` packages can help to quickly setup the host system for building images. In case the host must support everything, there is also the main `kiwi-systemdeps` package that has all other existing systemdeps packages as its dependency.

Note

Pulling in all `kiwi-systemdeps` packages can result in a large number packages installed on the host., because the required packages themselves have other dependencies (for example, java for jing).

7.2 Setting up Repositories

The repository selection is a crucial part of an appliance. KIWI NG allows the end user to customize the selection of repositories and packages via the `repository` element.

7.2.1 Adding repositories

KIWI NG installs packages into an appliance from the repositories defined in the image description. This means that at least one repository **must** be defined. Otherwise, KIWI NG cannot fetch any packages.

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

```
<image schemaversion="8.0" name="{exc_image_base_name}">
    <!-- snip -->
    <repository type="rpm-md" alias="kiwi" priority="1">
        <source path="obs://Virtualization:Appliances:Builder/openSUSE_Leap_15.6"/>
    </repository>
    <repository type="rpm-md" alias="OS" imageinclude="true">
        <source path="{exc_repo}"/>
    </repository>
</image>
```

The above example specifies two repositories:

1. The repository belonging to the KIWI NG project: *obs://Virtualization:Appliances:Builder/openSUSE_Leap_15.6* at the Open Build Service (OBS).
2. The RPM repository belonging to the OS project: *{exc_repo}*, at the Open Build Service (OBS). The translated http URL is also included in the final appliance.

The `repository` element accepts one `source` child element that contains the URL of the repository in an correct format along with the following optional attributes:

- `imageinclude`: Specifies whether the repository should be added to the resulting image. Default is false.
- `imageonly`: A repository with `imageonly="true"` is not available during image build, but is present in the resulting appliance. Default is false.
- `priority`: An integer value for all packages in this repository. If the same package is available in more than one repository, then the one with the highest priority is used.
- `alias`: Name to use for the repository. It appears as the repository's name in the image visible via `zypper repos` or `dnf repolist`. If `alias` is not specified, KIWI NG generates an alias name using hex representation from uuid4.
- `repository_gpgcheck`: Specify whether or not this specific repository values the result of the repository signature validation. The default value is `false`.
- `package_gpgcheck`: Specify whether or not this specific repository values the result of the package signature validation for each package taken from this repository. The default value is `false`.

- `components`: Distribution components used for `deb` repositories. Default is `main`.
- `distribution`: Distribution name information, used for deb repositories.
- `profiles`: List of profiles to which this repository applies.
- `customize`: Script to run custom modifications to the repo file or files. Repo files allow for several customization options, but not all of them are supported to be set by kiwi through the current repository schema. As the used options do not follow any standard, and they are not compatible between package managers and distributions, the only way to handle this is through a script hook which is invoked with the repo file as parameter for each file created by KIWI NG.
  An example for a script call to add the `module_hotfixes` option for a `dnf` compatible repository configuration could look as follows:

```
repo_file=$1
echo 'module_hotfixes = 1' >> ${repo_file}
```

Note

If the script is provided as a relative path, it is expected to be found in the image description directory:

7.2.1.1 Supported repository paths

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

- `http://URL` or `https://URL` or `ftp://URL`: a URL to the repository available via HTTP(s) or FTP.
- `obs://$PROJECT/$REPOSITORY`: checks whether the repository `$REPOSITORY` of the project `$PROJECT` available on the Open Build Service (OBS). By default, KIWI NG looks for projects on build.opensuse.org (https://build.opensuse.org), but this can be overridden using the runtime configuration file (see *Section 7.7, "The Runtime Configuration File"*). Note that it is not possible to add repositories using the `obs://` path from **different** OBS instances (use direct URLs to the `.repo` file instead in this case).

- `obsrepositories:/`: special path only available for builds using the Open Build Service. The repositories configured for the OBS project where the KIWI NG image resides are made available inside the appliance. This allows you to configure the repositories of your image from OBS itself, without modifying the image description.

- `dir:///path/to/directory` or `file:///path/to/file`: an absolute path to a local directory or file available on the host building the appliance.

7.3 Adding and Removing Packages

In addition to the *Section 7.2, "Setting up Repositories"* setup, the package setup is required. KIWI NG allows the end user to completely customize the selection of packages via the `packages` element.

```
<image schemaversion="8.0" name="{exc_image_base_name}">
    <packages type="bootstrap">
        <package name="udev"/>
        <package name="filesystem"/>
        <package name="openSUSE-release"/>
        <!-- additional packages installed before the chroot is created -->
    </packages>
    <packages type="image">
        <package name="patterns-openSUSE-base"/>
        <!-- additional packages to be installed into the chroot -->
    </packages>
</image>
```

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

- `type`: either `bootstrap`, `image`, `delete`, `uninstall` or one of the following build types: `docker`, `iso`, `oem`, `kis`, `oci`.
  Packages for `type="bootstrap"` are pre-installed to populate the images' root file system before chrooting into it.
  Packages in `type="image"` are installed immediately after the initial chroot into the new root file system.
  Packages in `type="delete"` and `type="uninstall"` are removed from the image, for details see Uninstall System Packages (#uninstall-system-packages).

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

- `profiles`: a list of profiles to which this package selection applies (see *Section 7.4, "Image Profiles"*).
- `patternType`: selection type for patterns, supported values are: `onlyRequired`, `plus-Recommended`, see: The product and namedCollection element (#product-and-namedcollection-element).

The following sections describes the different child elements of a `packages` group.

7.3.1 The package element

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

```
<image schemaversion="8.0" name="{exc_image_base_name}">
    <!-- snip -->
    <packages type="bootstrap">
        <package name="udev"/>
    </packages>
</image>
```

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

```
<image schemaversion="8.0" name="{exc_image_base_name}">
    <!-- snip -->
    <packages type="bootstrap">
        <package name="/usr/bin/my-binary"/>
    </packages>
</image>
```

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

Packages can also be included only on specific host architectures via the `arch` attribute. KIWI NG compares the `arch` attributes value with the host architecture that builds the image according to the output of `uname -m`.

```
<image schemaversion="8.0" name="{exc_image_base_name}">
    <!-- snip -->
    <packages type="image">
        <package name="grub2"/>
        <package name="grub2-x86_64-efi" arch="x86_64"/>
        <package name="shim" arch="x86_64"/>
    </packages>
</image>
```

This results in `grub2-x86_64-efi` and `shim` being only installed if the build host is a 64bit x86 machine, but `grub2` will be installed independent of the architecture.

7.3.2 The `archive` element

In certain situations, it is necessary to include additional packages into the image that are not available in the package manager's native format. KIWI NG supports the inclusion of regular tar archives via the `archive` element, whose `name` attribute specifies the filename of the archive (KIWI NG looks for the archive in the image description folder).

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

KIWI NG extracts the archive into the root directory of the image using GNU tar (https://www.gnu.org/software/tar/). This means that only archives supported by it can be included. When multiple `archive` elements are specified then they are applied in a top to bottom order. If a file is already present in the image, then the file from the archive overwrites it (same as with the image overlay).

7.3.3 Uninstall System Packages

KIWI NG supports two different methods for removing packages from the appliance:

1. Packages present as a child element of `<packages type="uninstall">` are gracefully uninstalled by the package manager together with dependent packages and orphaned dependencies.

2. Packages present as a child element of `<packages type="delete">` are removed by RPM/DPKG without any dependency check, potentially breaking dependencies and compromising the underlying package database.

Both types of removals take place after `config.sh` is run in the *Section 7.10.1, "The Prepare Step"* (see also *Section 7.6, "User-Defined Scripts"*).

Warning

An `uninstall` packages request deletes:

- the listed packages
- the packages dependent on the listed ones
- any orphaned dependency of the listed packages

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

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

```
<image schemaversion="8.0" name="{exc_image_base_name}">
    <!-- snip -->
    <packages type="image">
        <package name="ca-certificates"/>
        <package name="coreutils"/>
        <package name="ninja"/>
        <package name="clang"/>
        <package name="meson"/>
        <archive name="foo_app_sources.tar.gz"/>
    </packages>
    <!-- These packages will be uninstalled after running config.sh -->
    <packages type="uninstall">
        <package name="ninja"/>
        <package name="meson"/>
        <package name="clang"/>
```

```
    </packages>
</image>
```

The tools `meson`, `clang` and `ninja` are then available during the *Section 7.10.1, "The Prepare Step"*, and they can be used in `config.sh` (for further details, see *Section 7.6, "User-Defined Scripts"*), for example to build `foo_app`:

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

The `<packages type="uninstall">` element ensures that the final appliance no longer contains the tools required to build `foo_app`, thus making the image smaller.

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

7.3.4 The `product` and `namedCollection` element

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

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

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

7.3.5 The `ignore` element

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

```
<image schemaversion="8.0" name="{exc_image_base_name}">
    <packages type="image" patternType="plusRecommended">
        <namedCollection name="network-server"/>
        <package name="grub2"/>
        <package name="kernel"/>
        <ignore name="ejabberd"/>
        <ignore name="puppet-server"/>
    </packages>
</image>
```

Packages can be marked as ignored during the installation by adding a `ignore` child element with the mandatory `name` attribute set to the name of the package. Optionally, you can specify the architecture via `arch`, similarly to The package element (#package-element).

Warning

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

7.4 Image Profiles

A *profile* is a namespace for additional settings that can be applied by KIWI NG in addition to the default settings (or other profiles), making it possible to build multiple appliances with the same build type but with different configurations.

The use of profiles is advisable to distinguish image builds of the same type but with different settings. In the following example, two virtual machine images of the `oem` type are configured: one for QEMU (using the `qcow2` format) and one for VMWare (using the `vmdk` format).

```
<image schemaversion="8.0" name="{exc_image_base_name}">
    <profiles>
        <profile name="QEMU" description="virtual machine for QEMU"/>
        <profile name="VMWare" description="virtual machine for VMWare"/>
    </profiles>
    <preferences>
```

```
        <version>15.0</version>
        <packagemanager>zypper</packagemanager>
    </preferences>
    <preferences profiles="QEMU">
        <type image="oem" format="qcow2" filesystem="ext4"/>
    </preferences>
    <preferences profiles="VMWare">
        <type image="oem" format="vmdk" filesystem="ext4"/>
    </preferences>
</image>
```

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

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

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

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

In the above example, the profile `QEMU` inherit the settings from `VM`.

For further details on the usage of *profiles*, see *Section 11.19, "Building Images with Profiles"*

7.5 Adding Users

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

```
<image schemaversion="8.0" name="{exc_image_base_name}">
    <users>
        <user
            password="this_is_soo_insecure"
            home="/home/me" name="me"
            groups="users" pwdformat="plain"
```

```
        />
        <user
            password="$1$wYJUgpM5$RXMMeASDc035eX.NbYWFl0"
            home="/root" name="root" groups="root"
        />
    </users>
</image>
```

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

- `name`: the UNIX username
- `password`: The password for the user account. It can be provided either in clear-text form (`pwdformat="plain"`) or in encrypted form (`pwdformat="encrypted"`). Using plain passwords is not advisable, as anyone with access to the image description can see the password. It is recommended to generate a hash of your password using `openssl` as follows:

  ```
  $ openssl passwd -1 -salt 'xyz' YOUR_PASSWORD
  ```

Additionally, the following optional attributes can be specified:

- `home`: the path to the user's home directory
- `groups`: A comma-separated list of UNIX groups. The first element of the list is used as the user's primary group. The remaining elements are appended to the user's supplementary groups. When no groups are assigned, the system's default primary group is be used.
- `id`: The numeric user ID of the account.
- `pwdformat`: The format in which `password` is provided, either `plain` or `encrypted` (the latter is the default).

7.6 User-Defined Scripts

Note

Abstract

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

KIWI NG supports the following optional scripts that it runs in a root environment (chroot) containing an appliance:

post_bootstrap.sh

Runs at the end of the `bootstrap` phase as part of the *Section 7.10.1, "The Prepare Step"*. The script can be used to configure the package manager with additional settings that apply in the following chroot-based installation step which completes the installation. The script can also be used for other tasks.

config.sh

Runs at the end of the *Section 7.10.1, "The Prepare Step"* and after users have been set and the *overlay tree directory* has been applied. It is usually used to apply a permanent and final change of data in the root tree, such as modifying a package-specific config file.

config-overlay.sh

Available only if `delta_root="true"` is set. In this case, the script runs at the end of the *Section 7.10.1, "The Prepare Step"* prior the umount of the overlay root tree. It runs after `config.sh` (if specified), and it is the last entry point to change the delta root tree.

config-host-overlay.sh

Available only if `delta_root="true"` is set. In this case, the script runs at the end of the *Section 7.10.1, "The Prepare Step"* prior the umount of the overlay root tree. The script is called **NOT CHROOTED** from the host with the image root directory as its working directory. It runs after `config.sh` and `config-overlay.sh` (if any or both are specified), and it is the last entry point to change the delta root tree.

images.sh

Executed at the beginning of the *Section 7.10.2, "The Create Step"*. It runs in the same image root tree created by the prepare step, but it is invoked whenever an image needs to be created from that root tree. It is normally used to apply image type specific changes to the root tree, such as a modification to a config file that must be done when building a live iso, but not when building a virtual disk image.

pre_disk_sync.sh

Executed for the disk image type `oem` only, and it runs right before the synchronization of the root tree into the disk image loop file. The `pre_disk_sync.sh` can be used to change content of the root tree as a last action before the sync to the disk image is performed. This is useful, for example, for deleting components from the system which were needed earlier or cannot be modified afterwards when syncing into a read-only filesystem.

disk.sh

Executed for the disk image type `oem` only, and it runs after the synchronization of the root tree to the disk image loop file. The chroot environment for this script call is the virtual disk itself and not the root tree. The script `disk.sh` is normally used to apply changes at parts of the system that are not an element of the file-based root tree, such as the partition table, the contents of the final initrd, the bootloader, filesystem attributes, etc.

KIWI NG executes scripts via the operating system if their executable bit is set (in that case, a shebang is mandatory); otherwise they are invoked via the Bash shell. If a script exits with a non-zero exit code, KIWI NG reports the failure and aborts the image creation.

7.6.1 Developing/Debugging Scripts

Creating a custom script may require some experimenting and testing. To help developers with this task, KIWI NG calls scripts associated with a `screen` session. The connection to `screen` is only done if KIWI NG is called with the `--debug` option.

In this mode, a script can be started using the following template:

```
# The magic bits are still not set

echo "break"
/bin/bash
```

Calling the script executes a `screen` session executes, which gives you access to the break in shell. You can then implement the desired script code in this environment.. Once the shell is closed the KIWI NG process continues.

In addition to providing a fully featured terminal throughout the execution of the script code, you also have have control of the session during the process of the image creation. Listing the active sessions for script execution can be done as follows:

```
$ sudo screen -list

There is a screen on:
     19699.pts-4.asterix     (Attached)
1 Socket in /run/screens/S-root.
```

Note

As shown above the screen session for executing the script code provides extended control, which can be considered a security risk. Because of that, KIWI NG only runs scripts through `screen` when explicitly enabled via the `--debug` switch. In production, all scripts must run natively and must not require a terminal to operate correctly.

7.6.1.1 Script Template for config.sh / images.sh

KIWI NG provides a collection of methods and variables that offer custom actions. For details, see Functions and Variables Provided by KIWI NG (#image-customization-methods). The following template shows how to import this information into a script:

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

...
```

Warning

Modifications of the unpacked root tree

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

7.6.1.2 Functions and Variables Provided by KIWI NG

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

7.6.1.2.1 Functions

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

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

baseSetRunlevel {value}

Set the default run level.

baseStripAndKeep {list of info-files to keep}

Helper function for the `baseStrip*` functions that reads a list of files to check from stdin for removing params: files which should be kept.

baseStripLocales {list of locales}

Removes all locales, except for the ones given as the parameter.

baseStripTranslations {list of translations}

Removes all translations, except those given as the parameter.

baseStripUnusedLibs

Removes libraries that are not directly linked against applications in the bin directories.

baseUpdateSysConfig {filename} {variable} {value}

Updates the contents of a sysconfig variable.

baseSystemdServiceInstalled {service}

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

baseSysVServiceInstalled {service}

Prints the name `${service}` if a SysV init service with the same name is found; otherwise it prints nothing.

baseSystemdCall {service_name} {args}

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

baseInsertService {servicename}

Activates the specified service via **`systemctl`**.

baseRemoveService {servicename}

Deactivates the specified service via **`systemctl`**.

baseService {servicename} {on|off}

Activates or deactivates a service via **`systemctl`**. The function requires the service name and the value `on` or `off` as parameters.

The following example enables the sshd service on boot:

```
baseService sshd on
```

suseInsertService {servicename}

Calls baseInsertService. It exists only for compatibility reasons.

suseRemoveService {servicename}

Calls baseRemoveService. It exists only for compatibility reasons.

suseService {servicename} {on|off}

Calls baseService. It exists only for compatibility reasons.

suseSetupProduct

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

baseVagrantSetup

Configures the image to work as a vagrant box by performing the following changes:

- add the `vagrant` user to `/etc/sudoers` or `/etc/sudoers.d/vagrant`
- insert the insecure vagrant ssh key, apply recommended ssh settings and start the ssh daemon
- create the default shared folder `/vagrant`

Debug {message}

Helper function to print the supplied message if the variable DEBUG is set to 1 (disabled by default).

Echo {echo commandline}

Helper function to print a message to the controlling terminal.

Rm {list of files}

Helper function to delete files and log the deletion.

7.6.1.2.2 Profile Environment Variables

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

$kiwi_compressed

A value of the `compressed` attribute set in the `type` element in `config.xml`.

$kiwi_delete

A list of all packages which are children of the `packages` element with `type="delete"` in `config.xml`.

$kiwi_drivers

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

$kiwi_iname

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

$kiwi_iversion

The image version as a string.

$kiwi_keytable

The contents of the keytable setup as specified in `config.xml`.

$kiwi_language

The contents of the locale setup as specified in `config.xml`.

$kiwi_profiles

A comma-separated list of profiles used to build this image.

$kiwi_timezone

The contents of the timezone setup as specified in `config.xml`.

$kiwi_type

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

Note

.profile.extra

If there is the file `/.profile.extra` available in the initrd, KIWI NG imports the file importing `/.profile`.

7.6.1.3 Configuration Tips

1. **Locale configuration:**

   To set locale, KIWI NG relies on **`systemd-firstboot`** that writes the locale configuration file `/etc/locale.conf`. The values for the locale settings are taken from the description XML file in the `<locale>` element under `<preferences>`.

   Keep im mind that if the build distribution does not use `/etc/locale.conf`, the **`systemd-firstboot`** does not have any effect on the locale settings. For example, in the SLE12 distribution, **`systemd-firstboot`** is only effective when locales in `/etc/sysconfig/language` are not set, or when the file does not exist at all. For compatibility reasons, the file `/etc/sysconfig/language` in SLE12 has precedence over `/etc/locale.conf`, and management tools can still use `sysconfig` files for locale settings.

   In any case, it is possible to configure the locale setting inside the `config.sh` script in KIWI NG using in distribution-specific way, or by adding any additional configuration file as part of the overlay root-tree.

2. **Stateless systemd UUIDs:**

   Machine ID files (`/etc/machine-id`, `/var/lib/dbus/machine-id`) may be created and set during the image package installation depending on the distribution. Those UUIDs must be unique and must be set only once in each deployment.

   If `/etc/machine-id` does not exist or contains the string `uninitialized` (systemd v249 and later), this triggers firstboot action in systemd, and the services are run with `ConditionFirstBoot=yes`. Unless the file already contains a valid machine ID, systemd generates a machine ID and write it into the file, creating it if necessary. See the machine-id man page (https://www.freedesktop.org/software/systemd/man/machine-id.html) for more details.

   Depending on whether firstboot action should be triggered or not, `/etc/machine-id` can be created, removed, or set to `uninitialized` by `config.sh`.

   To prevent images from including a generated machine ID, KIWI NG clears `/etc/machine-id` if it exists and does not contain the string `uninitialized`. This only applies to images based on a dracut initrd.

Note

`rw` is necessary if `/etc/machine-id` does not exist.

For systemd to be able to write `/etc/machine-id` on boot, either the file must exist (so that a bind mount can be created) or `/etc` must be writable.

By default, the root filesystem is mounted read-only by dracut/systemd, so a missing `/etc/machine-id` will lead to an error on boot. To force the initial mount to be read-write, add the `rw` option to the kernel commandline.

Note

Avoid inconsistent `/var/lib/dbus/machine-id`

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

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

7.7 The Runtime Configuration File

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

The runtime configuration file must adhere to the YAML (https://yaml.org/) syntax, and the file can be pointed to via the global `--config` option at call time of KIWI NG. If no config file is provided, KIWI NG searches for the runtime configuration file in the following locations:

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

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

7.8 Customizing the Boot Process

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

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

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

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

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

`kiwi-overlay`
: Allows to boot disk images with the attribute `overlayroot` set to `true`. A disk like that has its root partition compressed and readonly. The disk boots up using overlayfs for the root filesystem with a separate partition on the same disk for persistent data.

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

`kiwi-lib`
: Provides common functions used by dracut modules.

Note

Using Custom Boot Image Support

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

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

Along with this setting, you must provide a reference to a boot image description in the `boot` attribute as follows:

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

While KIWI NG supports this approach, it is recommended using dracut instead. Keep also in mind that although KIWI NG supports creation of custom boot images, KIWI NG does not include any official boot image descriptions. You can find an OEM boot description example at https://build.opensuse.org/package/show/Virtualization:Appliances:Images:Testing_x86:tumbleweed/custom-oem-boot-description and an PXE boot description example at https://build.opensuse.org/package/show/Virtualization:Appliances:Images:Testing_x86:tumbleweed/custom-pxe-boot-description

The custom boot image descriptions makes it possible to completely customize the behavior of the initrd. This concept is mostly used in PXE environments that are usually heavily customized and require a specific boot and deployment workflow.

7.8.1 Boot Image Hook-Scripts

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

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

To hook in a custom boot script to this workflow, it is necessary to provide a dracut module that dracut picks when KIWI NG calls it. The module files can be provided either as a package or as part of the overlay directory in the image description.

The following example shows how to include a custom hook script before the system rootfs is 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 the configuration file:

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

add_dracutmodules+=" my-module "
```

3. Create the hook script:

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

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

```
#!/bin/bash

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

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

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

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

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

Note

Declaring Extra Tools for Hook Scripts

The `install()` function called by dracut can define extra tools required by the specified hook script. The `inst_multiple` command and its parameters instruct dracut to include these extra tools and items into the initrd.

The specified tools and items can be files. Normally, they are executables and libraries required by the hook script.

- Each file must be included in the KIWI NG description either in a package, archive, or in the root tree of the image description directory.
- The parameters of the `inst_multiple` command are space separated.
- Each parameter can be a single executable name if it exists in `/bin`, `/sbin`, `/usr/bin`, or `/usr/sbin` directories.
- Otherwise, a full path to the file is required. This normally applies for libraries and other special files.

When KIWI NG calls dracut, the `90my-module` 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 many other possibilities to customize the initrd than shown in the example above. For more information, visit the dracut project page (https://dracut.wiki.kernel.org/index.php/Main_Page).

7.8.2 Boot Image Parameters

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

`rd.kiwi.term`

Exports the TERM variable into the initrd environment. If the default value for the terminal emulation is not correct, `rd.kiwi.term` can be used to overwrite the default. The environment is also passed to the systemd unit that calls dialog based programs in KIWI NG dracut code, which means that the TERM setting applies there too.

`rd.kiwi.debug`

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

`rd.kiwi.dialog.timeout=seconds|off`

Sets a timeout value for dialogs invoked by kiwi dracut modules. By default the timeout is set to 60 seconds. If set to the special value `off`, the dialog will never timeout.

`rd.kiwi.install.devicepersistency=by-*`

Instructs an OEM installation image to present the list of available disk devices using the device names as provided by the specified `by-*` representation. For example `rd.kiwi.install.devicepersistency=by-path` would show the by-path device names in the list of available disks.

`rd.kiwi.install.pxe`

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

`rd.kiwi.install.image=URI`

Specifies the remote location of the system image in a PXE based OEM installation.

`rd.kiwi.install.pass.bootparam`

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

`rd.kiwi.install.retain_last`

Instructs an OEM installation to retain the contents of the last partition on the target disk. This setting is only useful if the last partition does not belong to the main OS e.g. an extra data partition added via the `spare_part` attribute in the type setup of the image description. The implementation also checks if the start address of the last partition on the target disk matches with the start adress of the image to be deployed. Only if they match the data on the last partition can be retained.

`rd.kiwi.oem.luks.reencrypt`

For OEM LUKS2 encrypted disk images. If set, reencrypts the disk prior an eventual resize and therefore creates a new key pool and master key. The reencryption is advisable if the image binary is not protected. With access to the image binary it's possible to extract the luks header which then allows to decrypt the data unless it was reencrypted. The

reencryption process only runs if the checksum of the luks header still matches the one from the original disk image. Be aware that the reencryption will ask for the passphrase if the image has been built with an initial luks passphrase.

`rd.kiwi.oem.luks.reencrypt_randompass`

For OEM LUKS2 encrypted disk images in combination with `rd.kiwi.oem.luks.reencrypt`. Reset insecure built time passphrase, set via the `luks=` attribute, with a random onetime passphrase that will be stored in memory at `/run/.kiwi_reencrypt.keyfile`.

Warning

The passphrase will only persist as long as the system does not reboot. Using this option usually requires that the boot process implements code to set a retrievable keyfile information for subsequent boot processes of this system, e.g TPM setup or similar.

`rd.kiwi.oem.disk.consistency`

For OEM disk images providing an installation image. If set, the installation image will check against all disks that are not the selected target disk if there is any disk in the system that has the same PTUUID compared to the image that is about to be installed. If such a disk is found this indicates that the same image was already installed to another storage disk on the same system which will cause device id inconsistencies for the entire system. In such a case an error message is displayed providing information about the conflicting device and the installation will be cancelled.

`rd.kiwi.oem.maxdisk=size[KMGT]`

Specifies the maximum disk size an unattended OEM installation uses for image deployment. Unattended OEM deployments default to deploying on `/dev/sda` (or more precisely, the first device that is not filtered out by `oem-device-filter`). With RAID controllers, you may have big JBOD disks along with a 480G RAID1 configured for OS deployment. With `rd.kiwi.oem.maxdisk=500G`, the deployment is performed on the RAID disk.

`rd.kiwi.oem.force_resize`

Forces the disk resize process on an OEM disk image. If set, no sanity check for unpartitioned/free space is performed and also an eventually configured `<oem-resize-once>` configuration from the image description will not be taken into account. The disk resize will be started which includes re-partition as well as all steps to resize the block layers up to the filesystem holding the data. As `rd.kiwi.oem.force_resize` bypasses all sanity

checks to detect if such a resize process is needed or not, it can happen that all program calls of the resize process ends without any effect if the disk is already properly resized. It's also important to understand that the partition UUIDs will change on every resize which might be an unwanted side effect of a forced resize.

`rd.kiwi.oem.installdevice`

Configures the disk device to use in an OEM installation. This overwrites or resets any other OEM device-specific settings, such as `oem-device-filter`, `oem-unattended-id` or `rd.kiwi.oem.maxdisk`, and continues the installation on the given device. The device must exist and must be a block special.

Note

Non interactive mode activated by rd.kiwi.oem.installdevice

When setting `rd.kiwi.oem.installdevice` explicitly through the kernel command line, KIWI NG uses the device without prompting for confirmation.

`rd.live.overlay.size`

Specifies the size for the `tmpfs` filesystem of a live ISO image that is used for the `overlayfs` mount process. If the write area of the overlayfs mount uses this tmpfs, any new data written during the runtime of the system is written in this space. The default value is `50%`, meaning half of the available RAM space can be used for writing new data.

`rd.live.overlay.persistent`

Instructs a live ISO image to prepare a persistent write partition.

`rd.live.encrypt`

Instructs a live ISO image to encrypt the persistent write partition. The boot process becomes interactive in this case and the user has to specify a passphrase at boot time.

`rd.live.overlay.cowfs`

Specifies which filesystem of a live ISO image to use for storing data on the persistent write partition.

`rd.live.cowfile.mbsize`

Specifies the size of the COW file in MB. When using tools like `live-grub-stick`, the live ISO image is copied as a file on the target device, and a GRUB loopback setup is created there to boot the live system from the file. In this case, the persistent write setup that

normally creates an extra write partition on the target will fail in most situations, because the target has no free and unpartitioned space available. To prevent this from happening, a COW file (live_system.cow) of a partition is created alongside the live ISO image file. The default size of the COW file is 500MB.

`rd.live.cowfile.path`

Effectively used in isoscan loop mounted live systems. For details on this type of live system refer to *Section 11.2, "Deploy ISO Image as File on a FAT32 Formated USB Stick"*. Specifies the path of the COW file below the `/run/initramfs/isoscan` loop mount point. If not specified the cowfile is placed at `/run/initramfs/isoscan/live_system.cow`.

`rd.live.dir`

Specifies a directory that contains the live OS root directory. Default is `LiveOS`.

`rd.live.squashimg`

Specifies the name of the squashfs image file which contains the OS root. Default is `squashfs.img`.

`rd.kiwi.allow_plymouth`

By default kiwi stops plymouth if present and active in the initrd. Setting rd.kiwi.allow_plymouth will keep plymouth active in the initrd including all effects that might have to the available consoles.

7.8.2.1 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. To prevent this, activate dracut's builtin debug mode in combination with the KIWI NG debug mode as follows:

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

This must be set at the kernel command line. With these parameters activated, the system enters a limited shell environment when a fatal error occurs during boot. The shell provides a basic set of tools, and it can be used for inspection using the following command:

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

7.9 Overview

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

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

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

Chapter 8, Image Description:

The `config.xml` file contains the image description, which is a collection of general settings of the final image, like the image layout installed packages, present users, etc.

Note

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

Section 7.6, "User-Defined Scripts":

If present, custom configuration shell scripts run at different stages of the build process. They can be used to fine tune the image in ways that are not possible via the settings provided in `config.xml`.

Overlay tree directory:

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

CD root user data:

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

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

7.10 Image Building Process

KIWI NG creates images in a two step process: The first step, the *prepare* operation, generates a so-called *unpacked image tree* (directory) using the information provided in the `config.xml` configuration file (see *Chapter 8, Image Description*)

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

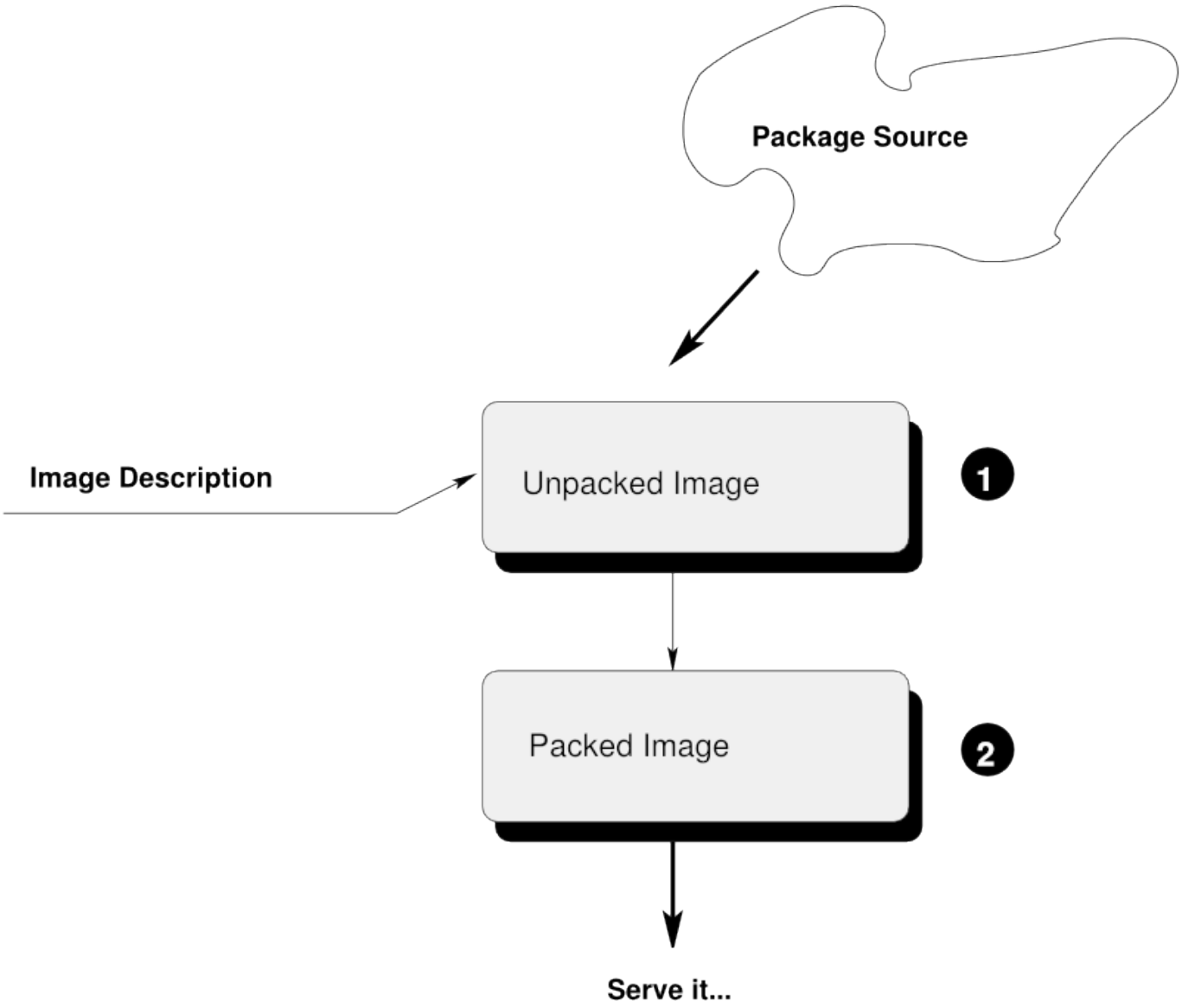

FIGURE 7.1: IMAGE CREATION ARCHITECTURE

7.10.1 The Prepare Step

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

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

The prepare step consists of the following substeps:

1. **Create Target Root Directory**

   By default KIWI NG aborts with an error if the target root tree already exists to avoid accidental deletion of an existing unpacked image. The option `--allow-existing-root` can be used to work based on an existing root tree

2. **Bootstrap Target Root Directory**

   First, KIWI NG configures the package manager to use the repositories specified in the configuration file, via the command line, or both. After the repository setup, the packages specified in the `bootstrap` section of the image description are installed in a temporary directory external to the target root tree. This establishes the initial environment to support the completion of the process in a chroot setting. At the end of the `bootstrap` phase the script `post_bootstrap.sh` is executed, if present.

   ### Note

   The essential bootstrap packages are usually `filesystem` and `glibc-locale` to specify as part of the bootstrap. The dependency chain of these two packages is usually sufficient to populate the bootstrap environment with all required software to support the installation of packages into the new root tree.

3. **Install Packages**

   After the `bootstrap` phase all other `<packages>` sections are used to complete the installation as chroot operation. KIWI NG uses the package manager as installed in the `bootstrap` phase and installs all other packages as configured.

   ### Note

   The installation of software packages through the selected package manager may install unwanted packages. Removing these packages can be accomplished by marking them for deletion in the image description, see *Section 7.3.3, "Uninstall System Packages"*.

4. **Apply the Overlay Tree**

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

5. **Apply Archives**
   All archives specified in the `archive` element of the `config.xml` file are applied in the specified order (top to bottom) after the overlay tree copy operation is complete (see *Section 7.3.2, "The `archive` element"*). Files and directories are extracted relative to the top level of the new root tree. As with the overlay tree, it is possible to overwrite files already existing in the target root tree.

6. **Execute the user-defined script `config.sh`**
   At the end of the preparation stage the script `config.sh` is executed (if present). It is run in the top level directory of the target root tree. The script's primary function is to complete the system configuration. For more details about custom scripts see *Section 7.6, "User-Defined Scripts"*

7. **Modify the Root Tree**
   The unpacked image tree is now finished to be converted into the final image in the *create step*. It is possible to make manual modifications to the unpacked tree before it is converted into the final image.
   Since the unpacked image tree is just a directory, it can be modified using the standard tools. Optionally, it is also possible to "change root (**`chroot`**)" into it, for instance to invoke the package manager. Beside the standard file system layout, the unpacked image tree contains an additional directory named `/image` that is not present in a regular system. It contains information KIWI NG requires during the create step, including a copy of the `config.xml` file.
   By default, KIWI NG will not stop after the *prepare step* and will directly proceed with the *create step*. Therfore to perform manual modifications, proceed as follows:

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

Warning

Modifications of the unpacked root tree

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

7.10.2 The Create Step

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

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

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

1. **Execute the User-defined Script `images.sh`**
   At the beginning of the image creation process the script named `images.sh` is executed (if present). For more details about custom scripts see *Section 7.6, "User-Defined Scripts"*

2. **Create the Requested Image Type**
   KIWI NG converts the unpacked root into an output format appropriate for the requested build type.

8 Image Description

Note

This document explains the toplevel structure of the KIWI NG image description document for version 10.2.35

8.1 Image Description Elements

Note

This document provides a reference for the elements and attributes of the KIWI NG XML document in version 10.2.35

8.1.1 <image>

The toplevel of any KIWI NG image description

```
<image schemaversion="8.0" name="{exc_image_base_name}">
    <!-- descendants -->
</image>
```

The image definition starts with an image tag and requires the schema format at version 8.0. The attribute name specifies the name of the image which is also used for the filenames created by KIWI. Because we don't want spaces in filenames the name attribute must not have any spaces in its name.

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

displayname

Allows setup of the boot menu title for the selected boot loader. So you can have *suse-SLED-foo* as the image name but a different name as the boot display name. Spaces are not allowed in the display name because it causes problems for some boot loaders and kiwi did not take the effort to separate the ones which can display them correctly from the ones which can't

id

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

8.1.2 <include>

Optional include of XML file content from file

```
<image schemaversion="8.0" name="{exc_image_base_name}">
    <include from="file://description.xml"/>
</image>
```

with file `description.xml` as follows:

```
<image>
    <description type="system">
        <author>name</author>
        <contact>contact</contact>
        <specification>text</specification>
    </description>
</image>
```

This will replace the `include` statement with the contents of `description.xml`. The validation of the result happens after the inclusion of all `include` references. The value for the `from` attribute is interpreted as an URI, as of now only local URI types are supported as well as the `this://` resource locator which translates into the path to the KIWI image description.

Note

The include information must be embedded into an `<image>` root node. Only the inner elements of the root node will be included. The processing of XML data via XSLT always requires a root node which is the reason why this is required to be specified for include files as well.

Note

Nesting of include statements in other include files is not supported. This will lead to unresolved include statements in the final document and will cause the runtime checker to complain about it.

Note

The include is implemented via a XSLT stylesheet and therefore expects an XML document. Other markup formats are not supported as include reference.

8.1.3 <description>

Provide an image identity.

```
<description type="system">
  <author>name</author>
  <contact>contact</contact>
  <specification>text</specification>
</description>
```

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

The following optional sub sections can be inserted below the description tag:

license

Specifies the license name which applies to this image description.

8.1.4 <certificates>

Add a cert-file to the directory storing additional local CA certificates. The import will occur immediately after the bootstrap process, where the required CA update tooling is expected to be installed. This setting is useful for situations where certificates are not packaged, or the certificates are required during the build process, e.g. due to proxy servers in the build environment that need certificates in chroot. The required `target_distribution` attribute must be set to allow kiwi a correct matching for the CA store path and the update tool with regards to the image target distribution. The following settings apply:

Distribution	CA Store	Update Tool
SUSE	/etc/pki/trust/anchors	update-ca-certificates

Distribution	CA Store	Update Tool
RedHat	/etc/pki/ca-trust/source/anchors	update-ca-certificates
Debian Based	/usr/local/share/ca-certificates	update-ca-certificates
Arch Linux	/etc/ca-certificates/trust-source/anchors	update-ca-trust

```
<certificates target_distribution="suse|rhel|debian|archlinux">
  <certificate name="/some/ca/filename1"/>
  <certificate name="/some/ca/filename2"/>
</certificates>
```

8.1.5 <preferences>

Setup image type and layout.

```
<preferences arch="arch">
```

```
  <version>1.2.3</version>
  <packagemanager name="zypper"/>
  <type image="tbz"/>
</preferences>
```

The mandatory preferences section contains information about the supported image type(s), the used package manager, the version of this image, and further optional elements. The preferences section can be configured to apply only for a certain architecture. In this case specify the `arch` attribute with a value as it is reported by **`uname -m`**

8.1.5.1 <preferences><version>

The mandatory image version must be a three-part version number of the format: **Major**.**Minor**.**Release**. In case of changes to the image description the following rules should apply:

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

8.1.5.2 <preferences><packagemanager>

The mandatory packagemanager element specifies which package manager should be used to handle software packages. The packagemanager setup is connected to the distribution used to build the image. The following table shows which package manager is connected to which distributor:

Distribution	**Package Manager**
SUSE	zypper

Distribution	Package Manager
Red Hat	dnf4 / dnf5
Debian Based	apt
Arch Linux	pacman
Alpine Linux	apk

In general the specification of one preferences section is sufficient. However, it's possible to specify multiple preferences sections and distinguish between the sections via the profiles attribute.

In combination with the above the preferences element supports the following optional elements:

8.1.5.3 <preferences><rpm-locale-filtering>

locale-filtering can be set to "true" or "false". If set to "true" it sets the install_lang macro for RPM based installations to the RPM configured locale list. This results in language specific files to become filtered out by `rpm` if they don't match the configured list.

```
<preferences>
  <rpm-locale-filtering>true</rpm-locale-filtering>
</preferences>
```

Note

It depends on the individual package design if the install_lang macro contents apply to the package or not.

8.1.5.4 <preferences><rpm-check-signatures>

Specifies whether package signatures should be checked or not

```
<preferences>
  <rpm-check-signatures>true</rpm-check-signatures>
</preferences>
```

8.1.5.5 <preferences><rpm-excludedocs>

Specifies whether files marked as documentation should be skipped during installation

```
<preferences>
  <rpm-excludedocs>true</rpm-excludedocs>
</preferences>
```

8.1.5.6 <preferences><keytable>

Specifies the name of the console keymap to use. The value corresponds to a map file in `/usr/share/kbd/keymaps/xkb`.

```
<preferences>
  <keytable>us</keytable>
</preferences>
```

8.1.5.7 <preferences><timezone>

Specifies the time zone. Available time zones are located in the `/usr/share/zoneinfo` directory. Specify the attribute value relative to `/usr/share/zoneinfo`. For example, specify Europe/Berlin for `/usr/share/zoneinfo/Europe/Berlin`.

```
<preferences>
  <timezone>Europe/Berlin</timezone>
```

```
</preferences>
```

8.1.5.8 <preferences><locale>

Specifies the name of the UTF-8 locale to use, which defines the contents of the RC_LANG system environment variable used in the image and to run the custom scripts specified as part of the KIWI NG image description. Please note only UTF-8 locales are supported here which also means that the encoding must *not* be part of the locale information. This means you need to specify the locale using the 4-digit name like the following example: en_US or en_US,de_DE

```
<preferences>
  <locale>en_US</locale>
</preferences>
```

8.1.5.9 <preferences><bootsplash-theme>

Specifies the name of the plymouth bootsplash theme to use

```
<preferences>
  <bootsplash-theme>bgrt</bootsplash-theme>
</preferences>
```

8.1.5.10 <preferences><bootloader-theme>

Specifies the name of the bootloader theme to use if that used bootloader has theme support.

```
<preferences>
  <bootloader-theme>openSUSE</bootloader-theme>
</preferences>
```

Along with the version and the packagemanager at least one image type element must be specified to indicate which image type should be build.

8.1.5.11 <preferences><release-version>

Specifies the distribution global release version as consumed by package managers. Currently the release version is not set or set to `0` for package managers which requires a value to operate. With the optional `release-version` section, users have an opportunity to specify a custom value which is passed along the package manager to define the distribution release.

Note

The release version information is currently used in dnf/dnf5 and microdnf package managers only. It might happen that it gets applied to the other package manager backends as well. This will happen on demand though.

8.1.5.12 <preferences><type>

At least one type element must be configured. It is possible to specify multiple type elements in a preferences block. To set a given type description as the default image use the boolean attribute primary and set its value to true:

```
<preferences>
  <type image="typename" primary="true"/>
</preferences>
```

The image type to be created is determined by the value of the image attribute. The following list describes the supported types and possible values of the image attribute:

image="tbz"

A simple tar archive image. The tbz type packs the contents of the image root tree into a xz compressed tarball.

image="btrfs|ext2|ext3|ext4|squashfs|xfs"

A filesystem image. The image root tree data is packed into a filesystem image of the given type. An image of that type can be loop mounted and accessed according to the capabiities of the selected filesystem.

image="iso"

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.

image="oem"

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. For use in cloud frameworks like Amazon EC2, Google Compute Engine or Microsoft Azure this disk type also supports the common virtual disk formats.

image="docker"

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

image="oci"

An archive image that builds a container matching the OCI (Open Container Interface) standard. The container should be able to run with any oci compliant container engine.

image="appx"

An archive image suitable for the Windows Subsystem For Linux container engine. The image can be loaded From a Windows System that has support for WSL activated.

image="wsl"

An archive image suitable for the Windows Subsystem For Linux container engine >= v2.4.4. The image represents a gzip compressed tar file as described in https://learn.microsoft.com/en-us/windows/wsl/build-custom-distro and can be loaded From a Windows System that has support for WSL in that version activated.

image="kis"

An optional root filesystem image associated with a kernel and initrd. The use case for this component image type is highly customizable. Many different deployment strategies are possible.

For completion of a type description, there could be several other optional attributes and child elements. The `type` element supports a plethora of optional attributes, some of these are only relevant for certain build types and will be covered in extra chapters that describes the individual image types more detailed. Certain attributes are however useful for nearly all build types and will be covered next:

bootpartition="true|false":

Boolean parameter notifying KIWI NG whether an extra boot partition should be used or not (the default depends on the current layout). This will override KIWI NG's default layout.

bootpartsize="nonNegativeInteger":

For images with a separate boot partition this attribute specifies the size in MB. If not set the boot partition size is set to 200 MB

eficsm="true|false":

For images with an EFI layout, specify if the legacy CSM (BIOS) mode should be supported or not. By default CSM mode is enabled.

efipartsize="nonNegativeInteger":

For images with an EFI fat partition this attribute specifies the size in MB. If not set the EFI partition size is set to 20 MB

efifatimagesize="nonNegativeInteger":

For ISO images (live and install) the EFI boot requires an embedded FAT image. This attribute specifies the size in MB. If not set the FAT image size is set to 20 MB

efiparttable="msdos|gpt":

For images with an EFI firmware specifies the partition table type to use. If not set defaults to the GPT partition table type for disk images, MBR (msdos) for ISO images.

dosparttable_extended_layout="true|false":

For oem disk images, specifies to make use of logical partitions inside of an extended one. If set to true and if the msdos table type is active, this will cause the fourth partition to be an extended partition and all following partitions will be placed as logical partitions inside of that extended partition. This setting is useful if more than 4 primary partitions needs to be created in an msdos table

btrfs_quota_groups="true|false":

Boolean parameter to activate filesystem quotas if the filesystem is `btrfs`. By default quotas are inactive.

btrfs_set_default_volume="true|false":

For oem disk images using the btrfs filesystem, requests to set a default volume for the rootfs which is used when the filesystem gets mounted. In case a `true` value is provided or the attribute is not specified at all, kiwi will make a volume the default volume. This can be either `/` or the configured root subvolume or the configured root snapshot. Consequently the entry created for the rootfs in the `/etc/fstab` file will not contain any specific volume definition. In case a `false` value is provided, kiwi will not set any default volume which also means that the entry for the rootfs in the `/etc/fstab` file requires a volume definition which is placed by kiwi as a `subvol=` parameter in the respective fstab field entry. In addition the parameter `rootflags=subvol=` is added to the kernel commandline such that early initrd code has a chance to know about the rootfs volume.

btrfs_root_is_subvolume="true|false":

Tell kiwi to create a root volume to host (`/`) inside. The name of this subvolume is by default set to: `@`. The name of the subvolume can be changed via a volume entry of the form:

```
<systemdisk>
```

```
  <volume name="@root=TOPLEVEL_NAME"/>
</systemdisk>
```

By default the creation of a toplevel volume is set to: `true`

btrfs_root_is_snapper_snapshot="true|false":

Boolean parameter that tells KIWI NG to install the system into a btrfs snapshot. The snapshot layout is compatible with the snapper management toolkit and follows a concept by SUSE. By default snapshots are turned off.

btrfs_root_is_readonly_snapshot="true|false":

Boolean parameter notifying KIWI NG that the btrfs root filesystem snapshot has to made read-only. if this option is set to true, the root filesystem snapshot it will be turned into read-only mode, once all data has been placed to it. The option is only effective if `btrfs_root_is_snapper_snapshot` is also set to true. By default the root filesystem snapshot is writable.

bootstrap_package="package_name":

For use with the `apt` packagemanager only. Specifies the name of a bootstrap package which provides a bootstrap tarball in `/var/lib/bootstrap/PACKAGE_NAME.ARCH.tar.xz`. The tarball will be unpacked and used as the bootstrap rootfs to begin with. This allows for an alternative bootstrap method. For further details see *Section 11.22, "Circumvent Debian Bootstrap"*.

compressed="true|false":

Specifies whether the image output file should be compressed or not. This option is only used for filesystem only images or for the `pxe` or `cpio` types.

editbootconfig="file_path":

Specifies the path to a script which is called right before the bootloader is installed. The script runs relative to the directory which contains the image structure.

editbootinstall="file_path":

Specifies the path to a script which is called right after the bootloader is installed. The script runs relative to the directory which contains the image structure.

filesystem="btrfs|ext2|ext3|ext4|squashfs|xfs":

The root filesystem

firmware="efi|uefi|bios|ec2|ofw|opal":

Specifies the boot firmware of the appliance. This attribute is used to differentiate the image according to the firmware which boots up the system. It mostly impacts the disk layout and the partition table type. By default `bios` is used on x86, `ofw` on PowerPC and `efi` on ARM.

- `efi` Standard EFI layout
- `uefi` Standard EFI layout for secure boot
- `bios` Standard BIOS layout for x86
- `ec2` Standard BIOS layout for x86 using Xen grub modules for old style Xen boot in Amazon EC2
- `ofw` Standard PPC layout
- `opal` Standard openPOWER PPC64 layout. kexec based boot process

force_mbr="true|false":

Boolean parameter to force the usage of a MBR partition table even if the system would default to GPT. This is occasionally required on ARM systems that use a EFI partition layout but which must not be stored in a GPT. Note that forcing a MBR partition table incurs limitations with respect to the number of available partitions and their sizes.

fsmountoptions="option_string":

Specifies the filesystem mount options which are passed via the `-o` flag to **`mount`** and are included in `/etc/fstab`.

fscreateoptions="option_string":

Specifies the filesystem options used to create the filesystem. In KIWI NG the filesystem utility to create a filesystem is called without any custom options. The default options are filesystem specific and are provided along with the package that provides the filesystem utility. For the Linux `ext[234]` filesystem, the default options can be found in the `/etc/mke2fs.conf` file. Other filesystems provides this differently and documents information about options and their defaults in the respective manual page, e.g **`man mke2fs`**. With the `fscreateoptions` attribute it's possible to directly influence how the filesystem will be created. The options provided as a string are passed to the command that creates the filesystem without any further validation by KIWI NG. For example, to turn off the journal on creation of an ext4 filesystem the following option would be required:

```
<preferences>
```

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

kernelcmdline="string":

Additional kernel parameters passed to the kernel by the bootloader.

root_clone="number"

For oem disk images, this attribute allows to create `number` clone(s) of the root partition, with `number` >= 1. A clone partition is content wise an exact byte for byte copy of the origin root partition. However, to avoid conflicts at boot time the UUID of any cloned partition will be made unique. In the sequence of partitions, the clone(s) will always be created first followed by the partition considered the origin. The origin partition is the one that will be referenced and used by the system. Also see *Section 11.12, "Partition Clones"*

boot_clone="number"

Same as `root_clone` but applied to the boot partition if present

luks="passphrase|file:///path/to/keyfile|random":

Supplying a value will trigger the encryption of the partition serving the root filesystem using the LUKS extension. The supplied value represents either the passphrase string or the location of a key file if specified as `file://...` resource or the reserved name `random`. When using a passphrase the system will interactively ask for that passphrase on first boot unless it is set empty. In case of an empty passphrase the system cannot be considered secure. When using a key file the information from the file is read and used as a passphrase. The given key file is **not automatically** placed into the system or added to the `etc/crypttab` which means the passphrase in the key file is by default requested from an interactive dialog at boot time. When using the reserved word `random`, kiwi will create a key file with a random passphrase and place this information into `etc/crypttab`. This allows the system to boot without user interaction but also requires the initrd to be protected in some way because it will contain the keyfile. The use of `random` is therefore only secure if the image adds additional security that encrypts the initrd like it is e.g. done in the IBM secure execution process. If the encryption of the system is combined with the attribute `bootpartition="false"` it's important to understand that this will place `/boot` into the encrypted area of the system and leaves reading boot data from it as a responsibility to the bootloader. Not every bootloader can cope with that and those that can e.g. grub will then open an interactive dialog at the bootloader level asking for the credentials to decrypt the root filesystem.

luks_version="luks|luks1|luks2":

Specify which `LUKS` version should be used. If not set and by default `luks` is used. The interpretation of the default depends on the distribution and could result in either 'luks1' or 'luks2'. The specification of the `LUKS` version allows using a different set of `luksformat` options. To investigate the differences between the two please consult the `cryptsetup` manual page.

target_blocksize="number":

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

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

target_removable="true|false":

Indicate if the target disk for oem images is deployed to a removable device e.g a USB stick or not. This only affects the EFI setup if requested and in the end avoids the creation of a custom boot menu entry in the firmware of the target machine. By default the target disk is expected to be non-removable

selinux_policy.attribute="targeted|mls|minimum":

The `selinux_policy` attribute sets the SELinux policy to use. `targeted` policy is the default policy. Only change this option if you want to use the `mls` or `minimum` policy.

spare_part="number":

Request a spare partition right before the root partition of the requested size. The attribute takes a size value and allows a unit in MB or GB, e.g 200M. If no unit is given the value is considered to be mbytes. A spare partition can only be configured for the disk image type oem

spare_part_mountpoint="dir_path":

Specify mount point for spare partition in the system. Can only be configured for the disk image type oem

spare_part_fs="btrfs|ext2|ext3|ext4|xfs":

Specify filesystem for spare partition in the system. Can only be configured for the disk image type oem

spare_part_fs_attributes="attribute_list":

Specify filesystem attributes for the spare partition. Attributes can be specified as comma separated list. Currently the attributes `no-copy-on-write` and `synchronous-updates` are available. Can only be configured for the disk image type oem

spare_part_is_last="true|false":

Specify if the spare partition should be the last one in the partition table. Can only be configured for the `oem` type with oem-resize switched off. By default the root partition is the last one and the spare partition lives before it. With this attribute that setup can be toggled. However, if the root partition is no longer the last one the oem repart/resize code can no longer work because the spare part would block it. Because of that moving the spare part at the end of the disk is only applied if oem-resize is switched off. There is a runtime check in the KIWI NG code to check this condition

devicepersistency="by-uuid|by-label":

Specifies which method to use for persistent device names. This will affect all files written by kiwi that includes device references for example `etc/fstab` or the `root=` parameter in the kernel commandline. By default by-uuid is used

squashfscompression="uncompressed|gzip|lzo|lz4|xz|zstd":

Specifies the compression type for mksquashfs

erofscompression="text"

Specifies the compression type and level for erofs. The attribute is a free form text because erofs allows paramters for the different compression types. Please consult the erofs man page for details how to specify a value for the `-z` option on `mkfs.erofs` and pass a proper value as erofscompression

standalone_integrity="true|false":

For the `oem` type only, specifies to create a standalone `dm_integrity` layer on top of the root filesystem

integrity_legacy_hmac="true|false":

For the `oem` type only and in combination with the `standalone_integrity` attribute, Allow to use old flawed HMAC calculation (does not protect superblock).

Warning

Do not use this attribute unless compatibility with a specific old kernel is required!

integrity_keyfile="filepath":

For the `oem` type only and in combination with the `standalone_integrity` attribute, protects access to the integrity map using the given keyfile.

integrity_metadata_key_description="string":

For the `oem` type only and in combination with the `embed_integrity_metadata` attribute, specifies a custom description of an integrity key as it is expected to be present in the kernel keyring. The information is placed in the integrity metadata block. If not specified kiwi creates a key argument string instead which is based on the given `integrity_keyfile` filename. The format of this key argument is:

```
:BASENAME_OF_integrity_keyfile_WITHOUT_FILE_EXTENSION
```

embed_integrity_metadata="true|false":

For the `oem` type only, and in combination with the `standalone_integrity` attribute, specifies to write a binary block at the end of the partition serving the root filesystem, containing information to create the `dm_integrity` device map in the following format:

```
|header|0xFF|dm_integrity_meta|0xFF|0x0|
```

header:

Is a string of the following information separated by spaces

- **version**: currently set to `1`
- **fstype**: name of `filesystem` attribute
- **access**: either `ro` or `rw` depending on the filesystem capabilities
- `integrity`: fixed identifier value

dm_integrity_meta:

Is a string of the following information separated by spaces

- **provided_data_sectors**: number of data sectors
- **sector_size**: sector size in byte, defaults to 512

- **parameter_count**: number of parameters needed to construct the integrity device map. After the `parameter_count` a list of space separated parameters follows and the `parameter_count` specifies the quantity of these parameters
- **parameters**: The first element of the parameter list contains information about the used hash algorithm which is not part of the superblock and provided according to the parameters passed along when KIWI NG calls `integritysetup`. As of now this defaults to:
  - internal_hash:sha256

  All subsequent parameters are taken from the `flags` field of the dm-integrity superblock. see the dm-integrity documentation on the web for possible flag values.

verity_blocks="number|all":
: For the `oem` type only, specifies to create a dm verity hash from the number of given blocks (or all) placed at the end of the root filesystem For later verification of the device, the credentials information produced by `veritysetup` from the cryptsetup tools are needed. This data as of now is only printed as debugging information to the build log file. A concept to persistently store the verification metadata as part of the partition(s) will be a next step.

embed_verity_metadata="true|false":
: For the `oem` type only, and in combination with the `verity_blocks` attribute, specifies to write a binary block at the end of the partition serving the root filesystem, containing information for `dm_verity` verification in the following format:

```
|header|0xFF|dm_verity_credentials|0xFF|0x0|
```

header:
: Is a string of the following information separated by spaces

  - **version**: currently set to `1`
  - **fstype**: name of `filesystem` attribute
  - **access**: either `ro` or `rw` depending on the filesystem capabilities
  - `verity`: fixed identifier value

dm_verity_credentials:
: Is a string of the following information separated by spaces

- **hash_type**: hash type name as returned by `veritysetup`
- **data_blksize**: data blocksize as returned by `veritysetup`
- **hash_blksize**: hash blocksize as returned by `veritysetup`
- **data_blocks**: number of data blocks as returned by `veritysetup`
- **hash_start_block**: hash start block as required by the kernel to construct the device map
- **algorithm**: hash algorithm as returned by `veritysetup`
- **root_hash**: root hash as returned by `veritysetup`
- **salt**: salt hash as returned by `veritysetup`

Note

dm-verity Runtime Options

Additional veritysetup options can be provided at boot time using the kernel parameter `rd.kiwi.verity_options=option1,option2`. This allows runtime customization.

KIWI automatically provides `hash-offset` and `hash-block-size` parameters based on the image configuration. These should not be specified manually via `rd.kiwi.verity_options=`.

Common Options:

- `panic-on-corruption`: System panics on corruption detection
- `restart-on-corruption`: System restarts on corruption (often default)
- `ignore-corruption`: Ignore corruption (debugging only)
- `ignore-zero-blocks`: Skip verification of zero blocks
- `check-at-most-once`: Verify each block only once

For complete options, see veritysetup(8) (https://man7.org/linux/man-pages/man8/veritysetup.8.html).

overlayroot="true|false":

For the `oem` type only, specifies to use an `overlayfs` based root filesystem consisting out of a squashfs compressed read-only root filesystem combined with an optional write-partition or tmpfs. The optional kernel boot parameter `rd.root.overlay.temporary` can be used to point the write area into a `tmpfs` instead of a persistent write-partition. In this mode all written data is temporary until reboot of the system. The kernel boot parameter `rd.root.overlay.size` can be used to configure the size for the `tmpfs` that is used for the `overlayfs` mount process if `rd.root.overlay.temporary` is requested. That size configures the amount of space available for writing new data during the runtime of the system. The default value is set to `50%` which means one half of the available RAM space can be used for writing new data. By default the persistent write-partition is used. The size of that partition can be influenced via the optional `<size>` element in the `<type>` section or via the optional `<oem-resize>` element in the `<oemconfig>` section of the XML description. Setting a fixed `<size>` value will set the size of the image disk to that value and results in an image file of that size. The available space for the write partition is that size reduced by the size the squashfs read-only system needs. If the `<oem-resize>` element is set to `true` an eventually given `<size>` element will not have any effect because the write partition will be resized on first boot to the available disk space. To disable the use of any overlay the kernel boot parameter `rd.root.overlay.readonly` can be used. It takes precedence over all other overlay kernel parameters because it leads to the deactivation of any overlayfs based action and just boots up with the squashfs root filesystem. In fact this mode is the same as not installing the `kiwi-overlay` dracut module.

overlayroot_readonly_filesystem="squashfs|erofs":

For the `oem` type only, specifies the filesystem type to use as read-only filesystem in an `overlayroot` setup. By default `squashfs` is used

overlayroot_write_partition="true|false":

For the `oem` type only, allows to specify if the extra read-write partition in an `overlayroot` setup should be created or not. By default the partition is created and the kiwi-overlay dracut module also expect it to be present. However, the overlayroot feature can also be used without dracut (`initrd_system="none"`) and under certain circumstances it is handy to configure if the partition table should contain the read-write partition or not.

overlayroot_readonly_partsize="mbsize":

Specifies the size in MB of the partition which stores the squashfs compressed read-only root filesystem in an overlayroot setup. If not specified kiwi calculates the needed size by a preliminary creation of the squashfs compressed file. However this is only accurate

if no changes to the root filesystem data happens after this calculation, which cannot be guaranteed as there is at least one optional script hook which is allowed and applied after the calculation. In addition the pre-calculation requires some time in the build process. If the value can be provided beforehand this also speeds up the build process significantly

bootfilesystem="btrfs|ext2|ext3|ext4|xfs|fat32|fat16":

If an extra boot partition is required this attribute specify which filesystem should be used for it. The type of the selected bootloader might overwrite this setting if there is no alternative possible though.

flags="overlay|dmsquash":

For the iso image type specifies the live iso technology and dracut module to use. If set to overlay the kiwi-live dracut module will be used to support a live iso system based on squashfs + overlayfs. If set to dmsquash the dracut standard dmsquash-live module will be used to support a live iso system based on the capabilities of the upstream dracut module.

format="gce|ova|qcow2|vagrant|vmdk|vdi|vhd|vhdx|vhd-fixed":

For disk image type oem, specifies the format of the virtual disk such that it can run on the desired target virtualization platform.

formatoptions="string":

Specifies additional format options passed on to qemu-img formatoptions is a comma separated list of format specific options in a name = value format like qemu-img expects it. kiwi will take the information and pass it as parameter to the -o option in the qemu-img call

fsmountoptions="string":

Specifies the filesystem mount options which also ends up in fstab The string given here is passed as value to the -o option of mount

fscreateoptions="string":

Specifies options to use at creation time of the filesystem

force_mbr="true|false":

Force use of MBR (msdos table) partition table even if the use of the GPT would be the natural choice. On e.g some arm systems an EFI partition layout is required but must not be stored in a GPT. For those rare cases this attribute allows to force the use of the msdos table including all its restrictions in max partition size and amount of partitions

gpt_hybrid_mbr="true|false":

For disk types, create a hybrid GPT/MBR partition table with an 'accurate' MBR table that will have no 'bootable' flagged partition For ISO types, create a hybrid GPT/MBR partition table where the MBR partition table contains a whole-disk 'protective' partition and a second bootable-flagged partition (intended to make the image bootable in both UEFI and BIOS modes on as much hardware as possible) In both cases, only has any effect if the EFI partition table type is GPT

hybridpersistent="true|false":

For the live ISO type, triggers the creation of a partition for a COW file to keep data persistent over a reboot

hybridpersistent_filesystem="ext4|xfs":

For the live ISO type, set the filesystem to use for persistent writing if a hybrid image is used as disk on e.g a USB Stick. By default the ext4 filesystem is used.

initrd_system="kiwi|dracut|none":

Specify which initrd builder to use, default is set to `dracut`. If set to `none` the image is build without an initrd. Depending on the image type this can lead to a non bootable system as its now a kernel responsibility if the given root device can be mounted or not.

metadata_path="dir_path":

Specifies a path to additional metadata required for the selected image type or its tools used to create that image type.

Note

Currently this is only effective for the appx container image type.

installboot="failsafe-install|harddisk|install":

Specifies the bootloader default boot entry for the initial boot of a KIWI NG install image.

Note

This value is only evaluated for grub

install_continue_on_timeout="true|false":

Specifies the boot timeout handling for the KIWI NG install image. If set to "true" the configured timeout or its default value applies. If set to "false" no timeout applies in the boot menu of the install image.

installprovidefailsafe="true|false":

Specifies if the bootloader menu should provide an failsafe entry with special kernel parameters or not

installiso="true|false"

Specifies if an install iso image should be created. This attribute is only available for the `oem` type. The generated ISO image is an hybrid ISO which can be used as disk on e.g a USB stick or as ISO.

installpxe="true|false":

Specifies if a tarball that contains all data for a pxe network installation should be created. This attribute is only available for the `oem` type.

mediacheck="true|false":

For ISO images, specifies if the bootloader menu should provide an mediacheck entry to verify ISO integrity or not. Disabled by default and only available for the x86 arch family.

mdraid="mirroring|striping":

Setup software raid in degraded mode with one disk Thus only mirroring and striping is possible

primary="true|false":

Specifies this type to be the primary type. If no type option is given on the commandline, KIWI NG will build this type

ramonly="true|false":

For all images that are configured to use the overlay filesystem this setting forces any COW(Copy-On-Write) action to happen in RAM.

rootfs_label="string":

Label name to set for the root filesystem. By default `ROOT` is used

volid="string":

For the ISO type only, specifies the volume ID (volume name or label) to be written into the master block. There is space for 32 characters.

application_id="string":

For the ISO/(oem install ISO) type only, specifies the Application ID to be written into the master block. There is space for 128 characters.

vhdfixedtag="GUID_string":

For the VHD disk format, specifies the GUID

derived_from="string":

For container images, specifies the image URI of the container image. The image created by KIWI NG will use the specified container as the base root to work on.

delta_root="true|false":

For container images and in combination with the `derived_from` attribute. If `delta_root` is set to `true`, {kiwi-ng} creates a container image which only contains the differences compared to the given `derived_from` container. Such a container is on its own no longer functional and requires a tool which is able to provision a container instance from the `derived_from` container combined with the `delta_root` application container. Such a tool exists with the oci-pilot (https://github.com/Elektrobit/oci-pilot) project and allows to manage applications as containers that feels like native applications on the host system.

ensure_empty_tmpdirs="true|false":

For OCI container images, specifies whether to ensure /run and /tmp directories are empty in the container image created by Kiwi. Default is true.

publisher="string":

For ISO images, specifies the publisher name of the ISO.

eficsmpart_id="number":

For OEM images, specifies the partition number for the EFI CSM (legacy BIOS) partition. For details about the evaluation and impact of custom partition ids please see: *Section 11.10, "Custom Disk Partitions"*

efipart_id="number":

For OEM images, specifies the partition number for the EFI partition. For details about the evaluation and impact of custom partition ids please see: *Section 11.10, "Custom Disk Partitions"*

rootpart_id="number":

For OEM images, specifies the partition number for the ROOT partition. For details about the evaluation and impact of custom partition ids please see: *Section 11.10, "Custom Disk Partitions"*

bootpart_id="number":

For OEM images, specifies the partition number for the BOOT partition. For details about the evaluation and impact of custom partition ids please see: *Section 11.10, "Custom Disk Partitions"*

The following sections shows the supported child elements of the `type` element including references to their usage in a detailed type setup:

8.1.5.13 <preferences><type><luksformat>

The `luksformat` element is used to specify additional luks options passed on to the `cryptsetup luksFormat` call. The element requires the attribute `luks` to be set in the `<type>` section referring to `luksformat`. Several custom settings related to the LUKS and LUKS2 format features can be setup. For example the setup of the `dm_integrity` feature:

```
<luksformat>
  <option name="--cipher" value="aes-gcm-random"/>
  <option name="--integrity" value="aead"/>
</luksformat>
```

8.1.5.14 <preferences><type><bootloader>

The `bootloader` element is used to select the bootloader. At the moment, `grub2`, `systemd_boot` and the combination of zipl plus userspace grub2 `grub2_s390x_emu` are supported. The special `custom` entry allows to skip the bootloader configuration and installation and leaves this up to the user, which can be done by using the `editbootinstall` and `editbootconfig` custom scripts.

Note

bootloaders provides a very different set of features and only work within their individual implementation priorities. KIWI NG provides an API for bootloaders but not all API methods can be implemented for all bootloaders due to the fact that some features only exists in one but not in another bootloader. If a bootloader setting is used that is not understood by the selected bootloader the image build process will fail with an exception message.

name="grub2|systemd_boot|grub2_s390x_emu|zipl":

Specifies the bootloader to use for this image.

Note

systemd_boot ESP size

The implementation to support systemd-boot reads all data from the ESP (EFI Standard Partition). This also includes the kernel and initrd which requires the size of the ESP to be configured appropriately. By default KIWI NG configures the ESP with 20MB. For systemd_boot this is usually too small and can be changed with the `efipartsize` attribute. Reading boot relevant files from another filesystem requires to provide alternative EFI filesystem drivers e.g efifs and also needs adaptions on the setup of `bootctl`.

Note

systemd_boot and shim

At the moment the EFI image provided along with systemd-boot is not compatible with the shim signed loader provided in an extra effort by the distributions.

In addition to the mandatory name attribute, the following optional attributes are supported:

bls="true|false":
: Specifies whether to use Bootloader Spec-style configuration if `grub` in the image supports it. It is a no-op if the `blscfg` module is not available. This option is only available for `grub` and defaults to true if not set.

output_console="none|console|gfxterm|serial|vga_text|mda_text|morse|spkmodem":
: Specifies the bootloader output console. The attribute is available for the `grub` bootloader type. Multiple console types can be provided separated by a space

input_console="none|console|serial|at_keyboard|usb_keyboard":
: Specifies the bootloader input console. The attribute is available for the `grub` bootloader type. Multiple console types can be provided separated by a space. If no input_console is provided, the value for the output_console is used if specified.

grub_template="filename":

Specifies a custom grub bootloader template file which will be used instead of the one provided with Kiwi. A static bootloader template to create the grub config file is only used in Kiwi if the native method via the grub mkconfig toolchain does not work properly. As of today, this is only the case for live and install ISO images. Thus, this setting only affects the oem and iso image types.

The template file should contain a Template string (https://docs.python.org/3.4/library/string.html#template-strings) and can use the following variables:

Variable	Description
search_params	parameters needed for grub's `search` command to locate the root volume
default_boot	number of the default menu item to boot

Variable	Description
kernel_file	the filename of the kernel file
initrd_file	the filename of the initial ramdisk file
boot_options	kernel command line options for booting normally
failsafe_boot_options	kernel command line options for boot-

Variable	Description
	ing in failsafe mode
gfxmode	the resolution to use for the bootloader; passed to grub's `gfxmode` command
theme	the name of a graphical theme to use
boottimeout	the boot menu timeout, set by the

Variable	Description
	`timeout` attribute
boottimeout_style	the boot time-out style, set by the `timeout_style` attribute
serial_line_setup	directives used to initialize the serial port, set by the `serial_line` attribute
title	a title for the image:

Variable	Description
	this will be the `<image>` tag's `display-name` attribute or its `name` attribute if `display-name` is not set; see: <image> (#sec-image)
boot-path	the bootloader lookup path

Variable	Description
bootdirectory_name	the name of the grub directory
efi_image_name	architecture-specific EFI boot binary name
terminal_setup	the bootloader console mode, set by the `console` attribute

serial_line="string":

Specifies the bootloader serial line setup. The setup is effective if the bootloader console is set to use the serial line. The attribute is available for the grub bootloader only.

timeout="number":

Specifies the boot timeout in seconds prior to launching the default boot option. By default, the timeout is set to 10 seconds. It makes sense to set this value to `0` for images intended to be started non-interactively (e.g. virtual machines).

timeout_style="countdown|hidden":

Specifies the boot timeout style to control the way in which the timeout interacts with displaying the menu. If set, the display of the bootloader menu is delayed after the timeout expired. In countdown mode, an indication of the remaining time is displayed. The attribute is available for the grub loader only.

targettype="CDL|LDL|FBA|SCSI|GPT":

Specifies the device type of the disk zipl should boot. On zFCP and/or KVM devices, use `SCSI` or `GPT`; on DASD devices, use `CDL` or `LDL`; on emulated DASD devices, use `FBA`. The attribute is available for the zipl loader only.

8.1.5.15 <preferences><type><bootloader><securelinux>

Used to specify data required to setup secure linux execution. Secure linux execution reads the kernel, initrd and boot parameters from an encrypted data blob and couples the image to the machine it gets executed on. Typically the private key is protected in hardware on the machine itself. KIWI NG supports secure execution for the IBM secure linux target on the s390 platform along with the bootloaders `zipl` and `grub2-s390x-emu`

```
<securelinux>
    <hkd_cert name="some1-host.crt"/>
    <hkd_cert name="some2-host.crt"/>
    <hkd_ca_cert name="some-ca.crt"/>
    <hkd_sign_cert name="some1-signing.crt"/>
    <hkd_sign_cert name="some2-signing.crt"/>
    <hkd_revocation_list name="some1-revocation.crl"/>
    <hkd_revocation_list name="some2-revocation.crl"/>
</securelinux>
```

Except for the `hkd_ca_cert` all other certificates can be specified multiple times.

hkd_cert:

The file specified in hkd_cert defines the `Host Key Document` and tightly couples the image to the host matching the document

hkd_ca_cert:

Required in combination with `hkd_cert`, providing the `Common Authority` certificate (signed by the root CA) that is used to establish a chain of trust for the verification of the `Host Key Document`.

hkd_sign_cert:

Required in combination with `hkd_cert`, providing the `Signing` certificate that is used to establish a chain of trust for the verification of the `Host Key Document`.

hkd_revocation_list:

Optional in combination with `hkd_cert`, providing the the revocation list to check on use of expired certificates in the chain of trust.

8.1.5.16 <preferences><type><bootloader><bootloadersettings>

Used to specify custom arguments for the tools called to setup secure boot e.g `shiminstall`, installation of the bootloader e.g `grub-install` or configuration of the bootloader e.g `grub-mkconfig`.

```
<bootloadersettings>
    <shimoption name="--suse-enable-tpm"/>
    <shimoption name="--bootloader-id" value="some-id"/>
    <installoption name="--suse-enable-tpm"/>
    <configoption name="--debug"/>
</bootloadersettings>
```

Note

{kiwi-ng} does not judge on the given parameters and if the provided data is effectively used depends on the individual bootloader implementation.

8.1.5.17 <preferences><type><containerconfig>

Used to describe the container configuration metadata in docker or wsl image types. For details see: *Section 10.4, "Build a Container Image"* and: *Section 10.5, "Build a WSL Container Image"*

8.1.5.18 <preferences><type><vagrantconfig>

Used to describe vagrant configuration metadata in a disk image that is being used as a vagrant box. For details see: *Section 11.7, "Image Description for Vagrant"*

8.1.5.19 <preferences><type><systemdisk>

Used to describe the volumes of the disk area which contains the root filesystem. Volumes are either a feature of the used filesystem or LVM is used for this purpose. For details see: *Section 11.11, "Custom Disk Volumes"*

Note

When both `<partitions>` and `<systemdisk>` are used, `<partitions>` are evaluated first and mount points defined in `<partitions>` cannot be redefined as `<systemdisk>` volumes. The two types define a complete disk setup, so there cannot be any overlapping volumes or mount points. As a result, whatever is written in `<partitions>` cannot be expressed in the same way in `<volumes>`.

8.1.5.20 <preferences><type><partitions>

Used to describe the geometry of the disk on the level of the partition table. For details see: *Section 11.10, "Custom Disk Partitions"*

8.1.5.21 <preferences><type><oemconfig>

Used to customize the deployment process in an oem disk image. For details see: *Section 10.3.5, "OEM Customization"*

8.1.5.22 <preferences><type><size>

Used to customize the size of the resulting disk image in an oem image. For details see: *Section 10.2.2, "Modifying the Size of the Image"*

8.1.5.23 <preferences><type><machine>

Used to customize the virtual machine configuration which describes the components of an emulated hardware. For details see: *Section 10.2.3, "Customizing the Virtual Machine"*

8.1.5.24 <preferences><type><installmedia>

Used to customize the installation media images created for oem images deployment. For details see: *Section 10.3.6, "Installation Media Customization"*

8.1.5.25 <preferences><type><initrd>

Used to specify custom arguments for the initrd tooling e.g. dracut

```
<initrd action="setup">
    <dracut uefi="true"/>
</initrd>
<initrd action="add|omit">
    <dracut module="some_a"/>
    <dracut module="some_b"/>
    <dracut driver="driver_name"/>
</initrd>
```

dracut.uefi.true:

As part of the `setup` action, enables creating a UKI EFI binary

dracut.module.NAME:

As part of the `add` or `omit` action, adds or omits the given module name. The element can be specified multiple times

dracut.driver.NAME:

As part of the `add` or `omit` action, adds or omits the given kernel driver. The element can be specified multiple times and accepts kernel driver names without the .ko extension

8.1.6 <containers>

Setup containers to fetch from a registry assigned to one of the supported container backends

```
<containers source="registry.opensuse.org" backend="podman">
    <container name="some"/>
</containers>
```

```
<containers source="registry.opensuse.org" backend="container-snap">
    <container name="opensuse/bootable" tag="slowroll"/>
</containers>
```

The optional containers element specifies the location of one ore more containers on a registry `source` server. KIWI NG will take this information and fetch the containers as OCI archives to the image. On first boot those container archives will be loaded into the local container backend store for the selected backend and the archive files get deleted.

Supported `backend` values are `docker`, `podman`, and `container-snap`. The `backend` attribute is mandatory and specifies for which container backend the image should be available in the system. The `containers` element has the following optional attributes:

arch="arch"

The containers section can be configured to apply only for a certain architecture. In this case specify the `arch` attribute with a value as it is reported by **`uname -m`**.

profiles="name[,name]"

A list of profiles to which this containers selection applies (see *Section 7.4, "Image Profiles"*).

8.1.7 <containers><container>

Details about the container

```
<containers source="registry.opensuse.org" backend="podman">
    <container name="some"/>
</containers>
```

The `name` attributes is mandatory and specifies the name of the container as it exists in the registry. The `container` element has the following optional attributes:

path="some/path"

The path to the container in the registry. If not specified the value defaults to `/`

fetch_only="true|false"

If set to `true` kiwi will only fetch the container but does not setup the systemd unit for loading the container into the local registry. In this mode the container archive file stays in the system and can be handled in a custom way. By default `fetch_only` is set to `false`.

tag="tagname"

Specifies the container tag to fetch. If not set the tag name defaults to `latest`

arch="arch"

The container section can be configured to apply only for a certain architecture. In this case specify the `arch` attribute with a value as it is reported by **`uname -m`**.

8.1.8 <repository>

Setup software sources for the image.

```
<repository arch="arch">
  <source path="uri"/>
</repository>
```

The mandatory repository element specifies the location and type of a repository to be used by the package manager as a package installation source. KIWI NG supports apk, apt, dnf4, dnf5, pacman and zypper as package managers, specified with the packagemanager element. The repository element has the following optional attributes:

alias="name"

Specifies an alternative name for the configured repository. If the attribute is not specified KIWI NG will generate a random alias name for the repository. The specified name must match the pattern: `[a-zA-Z0-9_-.]+`

components="name"

Used for Debian (apt) based repositories only. Specifies the component name that should be used from the repository. By default the `main` component is used

distribution="name"

Used for Debian (apt) based repositories only. Specifies the distribution name used in the repository data structure.

imageonly="true|false"

Specifies whether or not this repository should be configured in the resulting image without using it at build time. By default the value is set to false

repository_gpgcheck="true|false"

Specifies whether or not this specific repository values the result of the repository signature validation. If not set, the default is `false`. If set the relevant key information needs to be provided on the KIWI NG commandline using the `--signing-key` option or via the `<signing>` element as part of the `<repository>` setting in the image description.

package_gpgcheck="true|false"

Specify whether or not this specific repository values the result of the package signature validation for each package taken from this repository. If not set, the default value is false. If set, the same key information requirements as for the `repository_gpgcheck` attribute applies.

customize="/path/to/custom_script"

Custom script hook which is invoked with the repo file as parameter for each file created by KIWI NG.

Note

If the script is provided as relative path it will be searched in the image description directory

imageinclude="true|false"

Specifies whether the given repository should be configured as a repository in the image or not. The default behavior is that repositories used to build an image are not configured as a repository inside the image. This feature allows you to change the behavior by setting the value to true.

Note

Scope of repository uri's

The repository is configured in the image according to the source path as specified with the path attribute of the source element. Therefore, if the path is not a fully qualified URL, you may need to adjust the repository file in the image to accommodate the expected location. It is recommended that you use the alias attribute in combination with the imageinclude attribute to avoid having unpredictable random names assigned to the repository you wish to include in the image.

password="string"

Specifies a password for the given repository. The password attribute must be used in combination with the username attribute. Dependent on the repository location this information may not be used.

username="name"

Specifies a user name for the given repository. The username attribute must be used in combination with the password attribute. Dependent on the repository location this information may not be used.

prefer-license="true|false"

The repository providing this attribute will be used primarily to install the license tarball if found on that repository. If no repository with a preferred license attribute exists, the search happens over all repositories. It's not guaranteed in that case that the search order follows the repository order like they are written into the XML description.

priority="number"

Specifies the repository priority for this given repository. Priority values are treated differently by different package managers. Repository priorities allow the package management system to disambiguate packages that may be contained in more than one of the configured repositories. The zypper package manager for example prefers packages from a repository with a *lower* priority over packages from a repository with higher priority values. The value 99 means "no priority is set". For other package managers please refer to the individual documentation about repository priorities.

sourcetype="baseurl|metalink|mirrorlist"

Specifies the source type of the repository path. Depending on if the source path is a simple url or a pointer to a metadata file or mirror list, the configured package manager needs to be setup appropriately. By default the source is expected to be a simple repository baseurl.

use_for_bootstrap="true|false"

Used for Debian (apt) based repositories only. It specifies whether this repository should be the one used for bootstrapping or not. It is set to 'false' by default. Only a single repository is allowed to be used for bootstrapping, if no repository is set for the bootstrap the last one in the description XML is used.

8.1.8.1 <repository><source>

The location of a repository is specified by the path attribute of the mandatory source child element:

```
<repository alias="kiwi">
  <source path="obs://Virtualization:Appliances:Builder/openSUSE_Leap_15.6"/>
</repository>
```

The location specification may include the `%arch` macro which will expand to the architecture of the image building host. The value for the path attribute may begin with any of the following location indicators:

- `file:///local/path/to/file` An absolute path to a file accessible through the local file system.
- `ftp://<ftp://>` A ftp protocol based network location.
- `http://<http://>` A http protocol based network location.
- `https://<https://>` A https protocol based network location.

Note

https repositories

When specifying a https location for a repository it is generally necessary to include the openssl certificates and a cracklib word dictionary as package entries in the bootstrap section of the image configuration. The names of the packages to include are individual to the used distribution. On SUSE systems as one example this would be `openssl-certs` and `cracklib-dict-full`

- `obs://Open:Build:Service:Project:Name` A reference to a project in the Open Build Service (OBS). KIWI NG translates the given project path into a remote url at which the given project hosts the packages.
- `obsrepositories:/` A placeholder for the Open Build Service (OBS) to indicate that all repositories are taken from the project configuration in OBS.

A repository `<source>` element can optionally contain one ore more signing keys for the packages from this repository like shown in the following example:

```
<repository alias="kiwi">
  <source path="obs://Virtualization:Appliances:Builder/openSUSE_Leap_15.6">
    <signing key="file:///path/to/sign_key_a"/>
    <signing key="file:///path/to/sign_key_b"/>
  </source>
</repository>
```

All signing keys from all repositories will be collected and incorporated into the keyring as used by the selected package manager.

8.1.9 <packages>

Setup software components to be installed in the image.

```
<packages type="type"/>
```

The mandatory packages element specifies the setup of a packages group for the given type. The value of the type attribute specifies at which state in the build process the packages group gets handled, supported values are as follows:

type="bootstrap"
: Bootstrap packages, list of packages to be installed first into a new (empty) root tree. The packages list the required components to support a chroot environment from which further software components can be installed

type="image"
: Image packages, list of packages to be installed as part of a chroot operation inside of the new root tree.

type="uninstall|delete"
: Packages to be uninstalled or deleted. For further details see *Section 7.3.3, "Uninstall System Packages"*

type="*image_type_name*"
: Packages to be installed for the given image type name. For example if set to type="iso", the packages in this group will only be installed if the iso image type is build.

The packages element must contain at least one child element of the following list to provide specific configuration information for the specified packages group:

8.1.9.1 <packages><package>

```
<packages type="image"/>
  <package name="name" arch="arch"/>
</packages>
```

The package element installs the given package name. The optional `arch` attribute can be used to limit the installation of the package to the host architecture from which KIWI NG is called. The `arch` attribute is also available in all of the following elements.

8.1.9.2 <packages><namedCollection>

```
<packages type="image" patternType="onlyRequired">
  <namedCollection name="base"/>
</packages>
```

The namedCollection element is used to install a number of packages grouped together under a name. This is a feature of the individual distribution and used in the implementation of the KIWI NG package manager backend. At the moment collections are only supported for SUSE and Red Hat based distributions. The optional `patternType` attribute is used to control the behavior of the dependency resolution of the package collection. `onlyRequired` installs only the collection and its required packages. `plusRecommended` installs the collection, any of its required packages and any recommended packages.

Note

Collections on SUSE

On SUSE based distributions collections are called `patterns` and are just simple packages. To get the names of the patterns such that they can be used in a namedCollection type the following command: `$ zypper patterns`. If for some reason the collection name cannot be used it is also possible to add the name of the package that provides the collection as part of a `package` element. To get the names of the pattern packages type the following command: `$ zypper search patterns`. By convention all packages that starts with the name “patterns-” are representing a pattern package.

Note

Collections on Red Hat

On Red Hat based distributions collections are called `groups` and are extra metadata. To get the names of these groups type the following command: `$ dnf group list -v`. Please note that since KIWI NG v9.23.39, group IDs are allowed only, e.g.: <namedCollection name="minimal-environment"/>

8.1.9.3 <packages><collectionModule>

```
<packages type="bootstrap">
    <collectionModule name="module" stream="stream" enable="true|false"/>
</packages>
```

In CentOS Stream >= 8 and Red Hat Enterprise Linux >= 8, there are Application Streams that are offered in the form of modules (using Fedora Modularity technology). To build images that use this content KIWI NG offers to enable/disable modules when using the `dnf` or `microdnf` package manager backend. Modules are setup prior the bootstrap phase and its setup persists as part of the image.

There are the following constraints when adding `collectionModule` elements:

- `collectionModule` elements can only be specified as part of the `<packages type="bootstrap">` section. This is because the setup of modules must be done once and as early as possible in the process of installing the image root tree.
- Disabling a module can only be done as a whole and therefore the `stream` attribute is not allowed for disabling modules. For enabling modules the stream` attribute is optional
- The `enable` attribute is mandatory because it should be an explicit setting if a module is effectively used or not.

8.1.9.4 <packages><file>

```
<packages type="image"/>
  <file name="name"/>
</packages>
```

The file element takes the `name` attribute and looks up the given name as file on the system. If specified relative KIWI NG looks up the name in the image description directory. The file is installed using the `rsync` program. The file element has the following optional attributes:

owner="user:group"

The `owner` attribute can be specified to make the file belonging to the specified owner and group. The ownership of the original file is meaningless in this case. The provided value is passed along to the `chown` program.

permissions="perms"

The `permissions` attribute can be specified to store the file with the provided permission. The permission bits of the original file are meaningless in this case. The provided value is passed along to the `chmod` program.

target="some/path"

The `target` attribute can be used to specify a target path to install the file to the specified directory and name. Eventually missing parent directories will be created.

8.1.9.5 <packages><archive>

```
<packages type="image"/>
  <archive name="name" target_dir="some/path"/>
</packages>
```

The archive element takes the `name` attribute and looks up the given name as file on the system. If specified relative KIWI NG looks up the name in the image description directory. The archive is installed using the `tar` program. Thus the file name is expected to be a tar archive. The compression of the archive is detected automatically by the tar program. The optional target_dir attribute can be used to specify a target directory to unpack the archive inside the image root tree.

8.1.9.6 <packages><ignore>

```
<packages type="image"/>
  <ignore name="name"/>
</packages>
```

The ignore element instructs the used package manager to ignore the given package name at installation time. Please note whether or not the package can be ignored is up to the package manager. Packages that are hard required by other packages in the install procedure cannot be ignored and the package manager will simply ignore the request.

8.1.9.7 <packages><product>

```
<packages type="image">
  <product name="name"/>
</packages>
```

The product element instructs the used package manager to install the given product. What installation of a product means is up to the package manager and also distribution specific. This feature currently only works on SUSE based distributions

8.1.10 <users>

Setup image users.

```
<users arch="arch">
  <user
```

```
    name="user"
    groups="group_list"
    home="dir"
    id="number"
    password="text"
    pwdformat="encrypted|plain"
    realname="name"
    shell="path"
  />
</users>
```

The optional users element contains the user setup KIWI NG should create in the system. The optional `arch` attribute can be used to limit the users setup to the host architecture from which KIWI NG is called. At least one user child element must be specified as part of the users element. Multiple user elements may be specified.

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

name="name":

the UNIX username

password="string"

The password for this user account. It can be provided either in cleartext form or encrypted. An encrypted password can be created using `openssl` as follows:

```
$ openssl passwd -1 -salt xyz PASSWORD
```

It is also possible to specify the password as a non encrypted string by using the pwdformat attribute and setting it's value to `plain`. KIWI NG will then encrypt the password prior to the user being added to the system.

Warning

plain text passwords

We do not recommend plain passwords as they will be readable in the image configuration in plain text

All specified users and groups will be created if they do not already exist. The defined users will be part of the group(s) specified with the groups attribute or belong to the default group as configured in the system. If specified the first entry in the groups list is used as the login group.

Additionally, the following optional attributes can be specified:

home="path":
: The path to the user's home directory

groups="group_a,group_b,group_c:id":
: A comma separated list of UNIX groups. The first element of the list is used as the user's primary group. The remaining elements are appended to the user's supplementary groups. When no groups are assigned then the system's default primary group will be used. If a group should be of a specific group id, it can be appended to the name separated by a colon.

Note

Group ID's can only be set for groups that does not yet exist at the time when KIWI NG creates them. A check is made if the desired group is already present and if it exists the user will become a member of that group but any given group ID from the KIWI NG configuration will **not** be taken into account. Usually all standard system groups are affected by this behavior because they are provided by the OS itself. Thus it's by intention not possible to overwrite the group ID of an existing group.

id="number":
: The numeric user id of this account.

pwdformat="plain|encrypted":
: The format in which `password` is provided. The default if not specified is `encrypted`.

8.1.11 <profiles>

Manage image namespace(s).

```
<profiles>
  <profile name="name" description="text"/>
</profiles>
```

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

For example to mark a set of packages as belonging to a profile, simply annotate them with the profiles attribute as shown below:

```
<packages type="image" profiles="profile_name">
  <package name="name"/>
</packages>
```

It is also possible to mark sections as belonging to multiple profiles by separating the names in the profiles attribute with a comma:

```
<packages type="image" profiles="profile_A,profile_B">
  <package name="name"/>
</packages>
```

If a section tag does not have a profiles attribute, it is globally present in the configuration. If global sections and profiled sections contains the same sub-sections, the profiled sections will overwrite the global sections in the order of the provided profiles. For a better overview of the result configuration when profiles are used we recommend to put data that applies in any case to non profiled (global) sections and only extend those global sections with profiled data. For example:

```
<preferences>
  <version>1.2.3</version>
  <packagemanager name="zypper"/>
</preferences>

<preferences profiles="oem_qcow_format">
  <type image="oem" filesystem="ext4" format="qcow2"/>
</preferences>

<preferences profiles="oem_vmdk_format">
  <type image="oem" filesystem="ext4" format="vmdk"/>
</preferences>
```

The above example configures two version of the same oem type. One builds a disk in qcow2 format the other builds a disk in vmdk format. The global preferences section without a profile assigned will be used in any case and defines those preferences settings that are common to any build process. A user can select both profiles at a time but that will result in building the disk format that is specified last because one is overwriting the other.

Use of one or more profile(s) during image generation is triggered by the use of the `--profile` command line argument. multiple profiles can be selected by passing this option multiple times.

8.2 Main Root

```
<image/>
```

The mandatory *Section 8.1.1, "<image>"* element represents the root (top-level element) of an image description. All other elements must be descendants of this element. There can be only one `image` element.

8.3 Image Identity

```
<description/>
```

The mandatory *Section 8.1.3, "<description>"* element contains information about the author, contact, license and the specification about the use case of this image. All data together forms the identity card of the image. There can be only one `description` element

8.4 Image Preferences

```
<preferences/>
```

The mandatory *Section 8.1.5, "<preferences>"* element contains information to classify the image and to describe the layout. All data about the image type, its version, the partition layout and much more is specified here. There can be multiple `preferences` elements

8.5 Image Software Sources

```
<repository/>
```

The mandatory *Section 8.1.8, "<repository>"* element contains information where to find the software packages that are used to build the image. There can be multiple `repository` elements

8.6 Image Content Setup

```
<packages/>
```

The mandatory *Section 8.1.9, "<packages>"* element contains information to list which software should be installed from the configured repositories into the image. Software can be defined as names for packages, collections, archives or products. There can be multiple `packages` elements

8.7 Image Users

```
<users/>
```

The optional *Section 8.1.10, "<users>"* element contains information about system users to be created inside of the image. There can be multiple `users` elements

8.8 Image Namespace

```
<profiles/>
```

The optional *Section 8.1.11, "<profiles>"* element contains information to create one or more namespaces to an image description. The namespace can be used with any of the above elements and therefore tie them into a namespace which can be selected at call time of KIWI NG

8.9 Image Includes

```
<include from="file://filename.xml"/>
```

The optional *Section 8.1.2, "<include>"* element allows to drop in the contents of the specified `filename.xml` file at the place were the `include` statement was specified in the document. The `include` statement is only allowed as descendant of the root (top-level element) of the image description.

9 Image Types

Note

Before building an image with KIWI NG it's important to understand the different image types and their meaning. This document provides an overview about the supported KIWI NG image types, their results and some words about the environment to run the build.

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. For further details refer to *Section 10.1, "Build an ISO Hybrid Live Image"*

Virtual Disk Image

An image representing the system disk, useful for cloud frameworks like Amazon EC2, Google Compute Engine or Microsoft Azure. For further details refer to *Section 10.2, "Build a Virtual Disk Image"*

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. For further details refer to: *Section 10.3, "Build an Expandable Disk Image"*

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. For further details refer to: *Section 10.4, "Build a Container Image"*

WSL Container Image

An archive image suitable for the Windows Subsystem For Linux container engine. The image can be loaded From a Windows System that has support for WSL activated. For further details refer to: *Section 10.5, "Build a WSL Container Image"*

KIS Root File System Image

An optional root filesystem image associated with a kernel and initrd. The use case for this component image type is highly customizable. Many different deployment strategies are possible. For further details refer to: *Section 10.6, "Build KIS Image (Kernel, Initrd, System)"*

AWS Nitro Enclave

An initrd based image using the `eif` binary format. The image is expected to be used in the AWS Nitro Enclave system or for testing in QEMU. For further details refer to: *Section 10.7, "Build an AWS Nitro Enclave"*

9.1 Image Results

KIWI NG execution results in an appliance image after a successful run of *Section 4.6, "kiwi-ng system build"* or *Section 4.7, "kiwi-ng system create"* command. The result is the image binary plus some additional metadata files which are needed for image deployment and/or exists for informative reasons. By default the output files follow this naming convention:

`<image-name>.<arch>-<version>.<extension>`

where `<image-name>` is the name stated in the *Chapter 8, Image Description* as an attribute of the *Section 8.1.1, "<image>"* element. The `<arch>` is the CPU architecture used for the build, `<version>` is the image version defined in *Section 8.1.5, "<preferences>"* element of the image description and the `<extension>` is dependent on the image type and its definition.

Any KIWI NG appliance build results in, at least, the following output files:

1. The image binary, `<image-name>.<arch>-<version>.<image-extension>`:
   This is the file containig the actual image binary, depending on the image type and its definition it can be a virtual disk image file, an ISO image, a tarball, etc.

2. The `<image-name>.<arch>-<version>.packages` file:
   This file includes a sorted list of the packages that are included into the image. In fact this is normalized dump of the package manager database. It follows the following cvs format where each line is represented by:
   `<name>|<epoch>|<version>|<release>|<arch>|<disturl>|<license>`

The values represented here are mainly based on RPM packages metadata. Other package managers may not provide all of these values, in such cases the format is the same and the fields that cannot be provided are set as `None` value. This list can be used to track changes across multiple builds of the same image description over time by diffing the packages installed.

3. The `<image-name>.<arch>-<version>.verified` file:
   This file is the output of a verification done by the package manager against the package data base. More specific it is the output of the **`rpm`** verification process or **`dpkg`** verification depending on the packaging technology selected for the image. In both cases the output follows the RPM verification syntax. This provides an overview of all packages status right before any boot of the image.

Depending on the image type, the following output files exists:

image="tbz"
: For this image type the result is mainly a root tree packed in a tarball:

  - **root archive**: *exc_image_base_name*.x86_64-1.15.6.tar.xz

image="btrfs|ext2|ext3|ext4|squashfs|xfs"
: The image root tree data is packed into a filesystem image of the given type, hence the resutl for an `ext4` image would be:

  - **filesystem image**: *exc_image_base_name*.x86_64-1.15.6.ext4

image="iso"
: The image result is an ISO file:

  - **live image**: *exc_image_base_name*.x86_64-1.15.6.iso

image="oem"
: An image representing an expandable disk image. KIWI NG can also produce an installation ISO for this disk image by setting `installiso="true"` in the *Section 8.1.5, "<preferences>"*) section or a tarball including the artifacts for a network deployment by setting `installpxe="true"`. For further details see *Section 10.3, "Build an Expandable Disk Image"*. The results for `oem` can be:

  - **disk image**: *exc_image_base_name*.x86_64-1.15.6.raw
  - **installation image (optional)**: *exc_image_base_name*.x86_64-1.15.6.install.iso
  - **installation pxe archive (optional)**: *exc_image_base_name*.x86_64-1.15.6.install.tar

  The disk image can also be provided in one of the various virtual disk formats which can be specified in `format` attribute of the *Section 8.1.5, "<preferences>"* section. For further details see *Section 10.2, "Build a Virtual Disk Image"*. The result for e.g `format="qcow2"` would be:

  - **disk image**: *exc_image_base_name*.x86_64-1.15.6.qcow2

  instead of the `raw` default disk format.

image="docker"

An archive image suitable for the docker container engine. The result is a loadable (**`docker load -i <image>`**) tarball:

- **container**: *`exc_image_base_name`*`.x86_64-1.15.6.docker.tar.xz`

image="oci"

An archive image that builds a container matching the OCI (Open Container Interface) standard. The result is a tarball matching OCI standards:

- **container**: *`exc_image_base_name`*`.x86_64-1.15.6.oci.tar.xz`

image="appx"

An archive image suitable for the Windows Subsystem For Linux container engine. The result is an `appx` binary file:

- **container**: *`exc_image_base_name`*`.x86_64-1.15.6.appx`

image="wsl"

An archive image suitable for the Windows Subsystem For Linux container engine >= v2.4.4. The result is a gzip compressed tar archive with the `wsl` extension:

- **container**: *`exc_image_base_name`*`.x86_64-1.15.6.wsl`

image="kis"

An optional root filesystem image associated with a kernel and initrd. All three binaries are packed in a tarball, see *Section 10.6, "Build KIS Image (Kernel, Initrd, System)"* for further details about the kis archive:

- **kis archive**: *`exc_image_base_name`*`.x86_64-1.15.6.tar.xz`

9.2 Image Bundle Format

The result files as mentioned above are used in the KIWI NG result bundler. The `kiwi-ng result bundle` command can be used to copy or package the mandatory image files to create a customer release. In this process it's possible to apply a specific name pattern suitable for the requirements of the release. A typical result bundle call can look like the following:

```
$ kiwi-ng result bundle --target-dir /path/to/image/build_result \
```

```
    --bundle-dir=/path/to/image/release_result \
    --id=release_identifier
```

In this call and depending on the image type the required files as they exist in `/path/to/image/build_result` are copied to `/path/to/image/release_result/`. The only modification on the file names is the `--id` information which is appended with a `-` to at the end of the version substring. If we take *`exc_image_base_name`*`.x86_64-1.15.6.iso` as example. This file would be bundled as *`exc_image_base_name`*`.x86_64-1.15.6-release_identifier.iso`

Depending on the use case and the customer requirements this naming schema and the default way how the kiwi bundler processes the result files is not appropriate. To allow for a more flexible naming schema when bundling results, KIWI NG allows to specify a bundle_format per type like in the following example:

```
<type image="..." bundle_format="name_pattern">
    <!-- type definition -->
</type>
```

The specified `name_pattern` is used as the base name for the image files the bundler uses. As part of the `name_pattern` the following placeholders which gets replaced by their real value can be used:

%N

Turns into the contents of the `name` attribute of the `<image>` section

%P

Turns into the profile name used at build time of the image. If multiple profiles were used to build the image the result name consists out of the individual profile names concatenated by a `_` in the order of their specification in the image description and/or the commandline.

%A

Turns into the architecture name at build time of the image. Arch names are taken from Python's `platform.machine` information.

%I

Turns into the identifier name given via the `--id` option at call time of the bundler

%T

Turns into the contents of the `image` attribute of the `<type>` section

%M

Turns into the major number of the `<version>` section

%m

Turns into the minor number of the `<version>` section

%p

Turns into the patch number of the `<version>` section

%v

Turns into the version text of the `<version>` section

10 Building Images for Supported Types

Note

This document provides an overview how to build and use the KIWI NG supported image types. All images that we provide for testing uses the root password: `linux`

10.1 Build an ISO Hybrid Live Image

Abstract. This page explains how to build a live image. It covers the following topics:

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

A Live ISO image is a system on a removable media, for example a CD/DVD or a USB stick. Booting a Live ISO image does not interfere with other system storage components, making it a useful portable system for demonstration, testing, and debugging.

To add a Live ISO build to your appliance, create a `type` element with `image` set to `iso` in the `config.xml` file as shown below:

```
<image schemaversion="8.0" name="Tumbleweed_appliance">
  <!-- snip -->
  <preferences>
    <type image="iso" primary="true" flags="overlay" hybridpersistent_filesystem="ext4"
 hybridpersistent="true"/>
    <!-- additional preferences -->
  </preferences>
  <!-- snip -->
</image>
```

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

- `flags`: Specifies the dracut module to use.
  If set to `overlay`, the kiwi-live dracut module supplied by KIWI NG is used for booting.
  If set to `dmsquash`, the dracut-supplied dmsquash-live module is used for booting.
  Both modules support a different set of live features. For details see overlay or dmsquash (#live-features)
- `filesystem`: Specifies the root filesystem for the live system.

If set to `squashfs`, the root filesystem is written into a squashfs image. This option is not compatible with device-mapper specific features of the dmsquash-live dracut module. In that case, use overayfs.
If set to a value different from `squashfs`, the root filesystem is written into a filesystem image of the specified type, and the filesystem image written into a squashfs image for compression.
The default value of this option is `ext4`.

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

With the appropriate settings specified in `config.xml`, you can build an image using KIWI NG:

```
$ sudo kiwi-ng system build \
      --description kiwi/build-tests/x86/leap/test-image-live \
      --set-repo https://download.opensuse.org/distribution/leap/15.6/repo/oss \
      --target-dir /tmp/myimage
```

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

```
$ sudo qemu -cdrom \
      kiwi-test-image-live.x86_64-1.15.6.iso \
      -m 4096 -serial stdio
```

The image is now complete and ready to use. See *Section 11.1, "Deploy ISO Image on an USB Stick"* and *Section 11.2, "Deploy ISO Image as File on a FAT32 Formated USB Stick"* for further information concerning deployment.

10.1.1 `overlay` or `dmsquash`

Whether you choose the `overlay` or `dmsquash` dracut module depends on the features you intend to use. The `overlay` module supports only overlayfs based overlays, but with automatic creation of a writable layer for persistence. The `dmsquash` module supports overlayfs as well as device-mapper based overlays.

The following list describes important Live ISO features and their support status in the `overlay` and `dmsquash` modules.

ISO scan

Usable in the same way with both dracut modules. This feature allows to boot the Live ISO as a file from a grub loopback configured bootloader. The `live-grub-stick` tool is one example that uses this feature. For details how to setup ISO scan with the `overlay` module see *Section 11.2, "Deploy ISO Image as File on a FAT32 Formated USB Stick"*

ISO in RAM completely

Usable with the `dmsquash` module through `rd.live.ram`. The `overlay` module does not support this mode, while KIWI NG supports RAM only systems as OEM deployment into RAM from an install ISO media. For details how to setup RAM only deployments in KIWI NG see: *Section 11.9, "Deploy and Run System in a RamDisk"*

Overlay based on overlayfs

Usable with both dracut modules. The readonly root filesystem is overlaid with a readwrite filesystem using the kernel overlayfs filesystem.

Overlay based on device mapper snapshots

Usable with the `dmsquash` module. A squashfs compressed readonly root is overlaid with a readwrite filesystem using a device mapper snapshot.

Media Checksum Verification

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

Live ISO through PXE boot

Boot the Live image via the network. This is possible with both modules, but it uses different technologies. The `overlay` module supports network boot only in combination with the AoE (Ata Over Ethernet) protocol. For details see *Section 11.16, "Booting a Live ISO Image from Network"*. The `dmsquash` module supports network boot by fetching the ISO image into memory from `root=live:` using the `livenet` module.

Persistent Data

Keep new data persistent on a writable storage device. This can be done with both modules but in different ways. The `overlay` module activates persistency with the kernel boot parameter `rd.live.overlay.persistent`. If the persistent setup cannot be created the fallback to the non persistent mode applies automatically. The `overlay` module auto detects if it is used on a disk or ISO scan loop booted from a file. If booted as disk, persistency is setup on a new partition of that disk. If loop booted from file, persistency is setup on a new

cow file. The cow file/partition setup can be influenced with the kernel boot parameters: `rd.live.overlay.cowfs` and `rd.live.cowfile.mbsize`. The `dmsquash` module configures persistency through the `rd.live.overlay` option exclusively and does not support the automatic creation of a write partition in disk mode.

10.2 Build a Virtual Disk Image

Abstract. This page explains how to build a simple disk image. It covers the following topics:

- define a simple disk image in the image description
- build a simple disk image
- run it with QEMU

A simple virtual disk image is a compressed system disk with additional metadata useful for cloud frameworks like Amazon EC2, Google Compute Engine, or Microsoft Azure. It is used as the native disk of a system, and it does not require an additional installation workflow or a complex first boot setup procedure.

To enable KIWI NG to build a simple disk image, add a `type` element with `image="oem"` in `config.xml`, where the `oem-resize` option disabled. An example configuration for a 42 GB large VMDK image with 512 MB RAM, an IDE controller and a bridged network interface is shown below:

```
<image schemaversion="8.0" name="Tumbleweed_appliance">
  <!-- snip -->
  <preferences>
    <type image="oem" filesystem="ext4" format="vmdk">
      <bootloader name="grub2" timeout="0"/>
      <size unit="G">42</size>
      <oemconfig>
          <oem-resize>false</oem-resize>
      </oemconfig>
      <machine memory="512" guestOS="suse" HWversion="4">
        <vmdisk id="0" controller="ide"/>
        <vmnic driver="e1000" interface="0" mode="bridged"/>
      </machine>
    </type>
    <!-- additional preferences -->
  </preferences>
  <!-- snip -->
</image>
```

The following attributes of the `type` element are deserve attention when building simple disk images:

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

The `bootloader`, `size` and `machine` child-elements of `type` can be used to customize the virtual machine image. These elements are described in the following sections: Setting up the Bootloader in the Image (#disk-bootloader), Modifying the Size of the Image (#disk-the-size-element) and Customizing the Virtual Machine (#disk-the-machine-element)

Once your image description is finished , you can build the image using the following KIWI NG command:

```
$ sudo kiwi-ng system build \
     --description kiwi/build-tests/x86/leap/test-image-disk-simple \
     --set-repo https://download.opensuse.org/distribution/leap/15.6/repo/oss \
     --target-dir /tmp/myimage
```

The resulting `.raw` image is stored in `/tmp/myimage`.

You can test the image using QEMU:

```
$ sudo qemu \
    -drive file=kiwi-test-image-disk-simple.x86_64-1.15.6.raw,format=raw,if=virtio \
    -m 4096
```

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

- *Section 11.4, "Image Description for Amazon EC2"*
- *Section 11.5, "Image Description for Microsoft Azure"*
- *Section 11.6, "Image Description for Google Compute Engine"*

For information on how to setup a Vagrant system, see: *Section 11.7, "Image Description for Vagrant"*.

10.2.1 Setting up the Bootloader in the Image

```
<preferences>
```

```
  <type>
     <bootloader name="grub2"/>
  </type>
</preferences>
```

The `bootloader` element defines which bootloader to use in the image, and the element offers several options for customizing its configuration.

For details, see: *Section 8.1.5.14, "<preferences><type><bootloader>"*

10.2.2 Modifying the Size of the Image

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

```
<preferences>
  <type image="oem" format="vmdk">
    <size unit="G" additive="true" unpartitioned="5">20</size>
    <oemconfig>
        <oem-resize>false</oem-resize>
    </oemconfig>
  </type>
</preferences>
```

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

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

10.2.3 Customizing the Virtual Machine

The `machine` child element of `type` can be used to customize the virtual machine configuration, including the number of CPUs and the connected network interfaces.

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

- `ovftype`: The OVF configuration type. The Open Virtualization Format is a standard for describing virtual appliances and distribute them in an archive called Open Virtual Appliance (OVA). The standard describes the major components associated with a disk image. The exact specification depends on the product using the format. Supported values are `zvm`, `powervm`, `xen` and `vmware`.
- `HWversion`: The virtual machine's hardware version (`vmdk` and `ova` formats only), refer to VMware documentation for further information on the supported values.
- `arch`: the VM architecture (`vmdk` format only). Valid values are `ix86` (= `i585` and `i686`) and `x86_64`.
- `xen_loader`: the Xen target loader which is expected to load the guest. Valid values are: `hvmloader`, `pygrub` and `pvgrub`.
- `guestOS`: The virtual guest OS' identification string for the VM (only applicable for `vmdk` and `ova` formats. Note that the name designation is different for the two formats). Note: For vmware ovftools, guestOS is a VMX GuestOS, but not VIM GuestOS. For instance, correct value for Ubuntu 64 bit is "ubuntu-64", but not "ubuntu64Guest". See GUEST_OS_KEY_MAP in guest_os_tables.h at https://github.com/vmware/open-vm-tools for another guestOS values.
- `min_memory`: The virtual machine's minimum memory in MB (`ova` format only).
- `max_memory`: The virtual machine's maximum memory in MB (`ova` format only).
- `min_cpu`: The virtual machine's minimum CPU count (`ova` format only).
- `max_cpu`: The virtual machine's maximum CPU count (`ova` format only).
- `memory`: The virtual machine's memory in MB (all formats).
- `ncpus`: The number of virtual CPUs available to the virtual machine (all formats).

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

10.2.3.1 Modifying the VM Configuration Directly

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

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

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

```
<preferences>
  <type image="oem" filesystem="ext4" format="vmdk">
    <machine memory="512" guestOS="suse" HWversion="4">
      <vmconfig-entry>numvcpus = "4"</vmconfig-entry>
      <vmconfig-entry>cpuid.coresPerSocket = "2"</vmconfig-entry>
    </machine>
  </type>
</preferences>
```

10.2.3.2 Adding Network Interfaces to the VM

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

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

The following example adds a bridged network interface that uses the `e1000` driver:

```
<preferences>
  <type image="oem" filesystem="ext4" format="vmdk">
    <machine memory="4096" guestOS="suse" HWversion="4">
      <vmnic driver="e1000" interface="0" mode="bridged"/>
    </machine>
  </type>
</preferences>
```

The `vmnic` element supports the following attributes:

- `interface`: **Mandatory** interface ID for the VM's network interface.
- `driver`: An optional driver.
- `mac`: The MAC address of the specified interface.
- `mode`: The mode of the interface.

Note that KIWI NG doesn **not** verify the values of the attributes, it only inserts them into the appropriate configuration files.

10.2.3.3 Specifying Disks and Disk Controllers

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

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

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

```
<preferences>
  <type image="oem" filesystem="ext4" format="vmdk">
    <machine memory="512" guestOS="suse" HWversion="4">
      <vmdisk id="0" controller="ide"/>
    </machine>
  </type>
</preferences>
```

Each `vmdisk` element can be further customized using optional attributes:

- `controller`: The disk controller used for the VM guest (`vmdk` format only). Supported values are: `ide`, `buslogic`, `lsilogic`, `lsisas1068`, `legacyESX` and `pvscsi`.
- `device`: The disk device to appear in the guest (`xen` format only).
- `diskmode`: The disk mode (`vmdk` format only). Valid values are `monolithicSparse`, `monolithicFlat`, `twoGbMaxExtentSparse`, `twoGbMaxExtentFlat` and `streamOptimized` (see also https://vdc-download.vmware.com/vmwb-repository/dcr-public/6335f27c-c6e9-4804-95b0-ea9449958403/c7798a8b-4c73-41d9-84e8-db5453de7b17/doc/vddkDataStruct.5.3.html).
- `disktype`: The type of the disk handled internally by the VM (`ova` format only). This attribute is currently unused.
- `id`: The disk ID of the VM disk (`vmdk` format only).

10.2.3.4 Adding CD/DVD Drives

KIWI NG supports adding IDE and SCSCI CD/DVD drives to the virtual machine using the `vmdvd` element for the `vmdk` image format. The following example adds two drives: one with a SCSCI and another with a IDE controller:

```
<preferences>
  <type image="oem" filesystem="ext4">
```

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

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

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

10.3 Build an Expandable Disk Image

Abstract. This page explains how to build an expandable disk image. It covers the following topics:

- build an expandable disk image
- deploy an expandable disk image
- run the deployed system

An expandable disk represents the system disk with the capability to automatically expand the disk and its filesystem to a custom disk geometry. This allows deploying the same disk image on target systems with different hardware setups.

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

1. Make sure you have checked out the example image descriptions (see *Section 2.4, "Example Appliance Descriptions"*).
2. Build an image with KIWI NG:

   ```
   $ sudo kiwi-ng --type oem system build \
       --description kiwi/build-tests/x86/leap/test-image-disk \
       --set-repo https://download.opensuse.org/distribution/leap/15.6/repo/oss \
       --target-dir /tmp/myimage
   ```

   The resulting image is saved in `/tmp/myimage`.

- The disk image with the suffix `.raw` is an expandable virtual disk. It can expand itself to a custom disk geometry.
- The installation image with the suffix `install.iso` is a hybrid installation system which contains the disk image and is capable to install this image on any target disk.

10.3.1 Deployment Methods

The goal of an expandable disk image is to provide the virtual disk data for OEM vendors to support easy deployment of the system to physical storage media.

Basic deployment strategies are as follows:

1. Manual Deployment (#deploy-manually)
   Manually deploy the disk image onto the target disk.
2. CD/DVD Deployment (#deploy-from-iso)
   Boot the installation image and let KIWI NG's installer deploy the disk image from CD/DVD or USB stick onto the target disk.
3. Network Deployment (#deploy-from-network)
   PXE boot the target system and let KIWI NG's installer deploy the disk image from the network onto the target disk.

10.3.2 Manual Deployment

The manual deployment method can be tested using virtualization software like QEMU and an additional virtual a large-size target disk. To do this, follow the steps below.

1. Create a target disk:

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

Note

Retaining the Disk Geometry

If the target disk geometry is less than or equals to the geometry of the disk image itself, the disk expansion that is performed on a physical disk install during the boot workflow is skipped and the original disk geometry stays unchanged.

2. Dump disk image on target disk:

```
$ dd if=kiwi-test-image-disk.x86_64-1.15.6.raw of=target_disk conv=notrunc
```

3. Boot the target disk:

```
$ sudo qemu -hda target_disk -m 4096 -serial stdio
```

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

10.3.3 CD/DVD Deployment

The deployment from CD/DVD via an installation image can also be tested using virtualization software such as QEMU. To do this, follow the steps below.

1. Create a target disk:
   Follow the steps above to create a virtual target disk

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

```
$ sudo qemu -cdrom \
      kiwi-test-image-disk.x86_64-1.15.6.install.iso -hda target_disk \
      -boot d -m 4096 -serial stdio
```

Note

USB Stick Deployment

Like any other ISO image built with KIWI NG, the installation image is also a hybrid image. Thus, it can also be used on USB stick and serve as installation media as explained in *Section 10.1, "Build an ISO Hybrid Live Image"*

10.3.4 Network Deployment

The process of deployment from the network downloads the disk image from a PXE boot server. This requires a PXE network boot server to be setup as described in *Section 11.13, "Setting Up a Network Boot Server"*

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

1. Create an installation PXE TAR archive along with your disk image by replacing the following configuration in kiwi/build-tests/x86/leap/test-image-disk/appliance.kiwi
   Find the line below:

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

   Modify the line as follows:

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

2. Rebuild the image, unpack the resulting `kiwi-test-image-disk.x86_64-1.15.6.install.tar.xz` file to a temporary directory, and copy the initrd and kernel images to the PXE server.

   1. Unpack installation tarball:

      ```
      mkdir /tmp/pxe && cd /tmp/pxe
      tar -xf kiwi-test-image-disk.x86_64-1.15.6.install.tar.xz
      ```

   2. Copy kernel and initrd used for PXE boot:

      ```
      scp pxeboot.kiwi-test-image-disk.x86_64-1.15.6.initrd PXE_SERVER_IP:/srv/
      tftpboot/boot/initrd
      scp pxeboot.kiwi-test-image-disk.x86_64-1.15.6.kernel PXE_SERVER_IP:/srv/
      tftpboot/boot/linux
      ```

3. Copy the disk image, SHA256 file, system kernel, initrd and bootoptions to the PXE boot server.
   Activation of the deployed system is done via `kexec` of the kernel and initrd provided here.

   1. Copy system image and SHA256 checksum:

      ```
      scp kiwi-test-image-disk.x86_64-1.15.6.xz PXE_SERVER_IP:/srv/tftpboot/image/
      scp kiwi-test-image-disk.x86_64-1.15.6.sha256 PXE_SERVER_IP:/srv/tftpboot/
      image/
      ```

2. Copy kernel, initrd and bootoptions used for booting the system via kexec:

```
scp kiwi-test-image-disk.x86_64-1.15.6.initrd PXE_SERVER_IP:/srv/tftpboot/
image/
scp kiwi-test-image-disk.x86_64-1.15.6.kernel PXE_SERVER_IP:/srv/tftpboot/
image/
scp kiwi-test-image-disk.x86_64-1.15.6.config.bootoptions PXE_SERVER_IP:/srv/
tftpboot/image/
```

Note

The config.bootoptions file is used with kexec to boot the previously dumped image. This file specifies the root of the dumped image, and the file can include other boot options. The file provided with the KIWI NG built image connected to the image present in the PXE TAR archive. If other images are deployed, the file must be modified to match the correct root reference.

Note

If the image gets deployed into a ramdisk which is configured by passing `rd.kiwi.ramdisk` as part of the append setup below, it is not required to copy the above mentioned kernel and initrd file to boot the system because for ramdisk deployments the currently active kernel and initrd will be used as kexec cannot be called with a system in memory.

4. Add/Update the kernel command line parameters.

   Edit your PXE configuration (for example `pxelinux.cfg/default`) on the PXE server, and add the following parameters to the append line similar to shown below:

```
append initrd=boot/initrd rd.kiwi.install.pxe
 rd.kiwi.install.image=tftp://192.168.100.16/image/kiwi-test-image-
disk.x86_64-1.15.6.xz
```

   The location of the image is specified as a source URI that can point to any location supported by the `curl` command. KIWI NG uses `curl` to fetch the data from this URI. This means that the image, checksum file, system kernel and initrd can be fetched from any server, and they do not need to be stored on the `PXE_SERVER`.

By default KIWI NG does not use specific `curl` options or flags. But it is possible to specify desired options by adding the `rd.kiwi.install.pxe.curl_options` flag to the kernel command line (`curl` options are passed as comma-separated values), for example:

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

The above instructs KIWI NG to run `curl` as follows:

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

This can be particularly useful when the deployment infrastructure requires specific download configuration. For example, setting more robust retries over an unstable network connection.

Note

KIWI NG replaces commas with spaces and appends the result to the `curl` command. Keep that in mind, because command-line options that include commas break the command.

Note

The initrd and Linux Kernel for PXE boot are always loaded via TFTP from the `PXE_SERVER`.

1. Create a target disk.
   Follow the steps above to create a virtual target disk.
2. Connect the client to the network and boot QEMU with the target disk attached to the virtual machine:

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

Note

QEMU bridged networking

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

10.3.5 OEM Customization

The deployment process of an OEM image can be customized using the `oemconfig` element. This element is a child section of the `type` element, for example:

```
<oemconfig>
  <oem-swapsize>512</oem-swapsize>
</oemconfig>
```

Below is a losr list of optional `oem` element settings.

oemconfig.oem-resize

Determines if the disk has the capability to expand itself to a new disk geometry or not. By default, this feature is activated. The implementation of the resize capability is done in a dracut module packaged as `dracut-kiwi-oem-repart`. If `oem-resize` is set to false, the installation of the corresponding dracut package can be skipped as well.

oemconfig.oem-boot-title

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

oemconfig.oem-bootwait

Determines if the system waits for user interaction before continuing the boot process after the disk image has been dumped to the designated storage device (default value is false). This value is represented by the `kiwi_oembootwait` variable in the initrd.

oemconfig.oem-reboot

When enabled, the system is rebooted after the disk image has been deployed to the designated storage device (default value is false). This value is represented by the `kiwi_oem-reboot` variable in the initrd.

oemconfig.oem-reboot-interactive

When enabled, the system is rebooted after the disk image has been deployed to the designated storage device (default value is false). Before the reboot, a message is displayed, and it and must be acknowledged by the user for the system to reboot. This value is represented by the `kiwi_oemrebootinteractive` variable in the initrd.

oemconfig.oem-silent-boot

Determines if the system boots in silent mode after the disk image has been deployed to the designated storage device (default value is false). This value is represented by the `kiwi_oemsilentboot` variable in the initrd.

oemconfig.oem-shutdown

Determines if the system is powered down after the disk image has been deployed to the designated storage device (default value is false). This value is represented by the `kiwi_oemshutdown` variable in the initrd.

oemconfig.oem-shutdown-interactive

Determines if the system is powered down after the disk image has been deployed to the designated storage device (default value is false). Before the shutdown a message is displayed, and it must be acknowledged by the user for the system to power off. This value is represented by the `kiwi_oemshutdowninteractive` variable in the initrd

oemconfig.oem-swap

Determines if a swap partition is be created. By default, no swap partition is created. This value is represented by the `kiwi_oemswap` variable in the initrd.

oemconfig.oem-swapname

Specifies the name of the swap space. By default, the name is set to `LVSwap`. The default indicates that this setting is only useful in combination with the LVM volume manager. In this case, the swapspace is configured as a volume in the volume group, and every volume requires a name. The name specified in `oemconfig.oem-swapname` here is used as a name of the swap volume.

oemconfig.oem-swapsize

Specifies the size of the swap partition. If a swap partition is created while the size of the swap partition is not specified, KIWI calculates the size of the swap partition, and creates a swap partition at initial boot time. In this case, the swap partition size equals the double amount of RAM of the system. This value is represented by the `kiwi_oemswapMB` variable in the initrd.

oemconfig.oem-systemsize

Specifies the MB size the operating system is allowed to occupy on the target disk. The size limit does not include any swap space or recovery partition considerations. In a set-up *without* the systemdisk element, this value specifies the size of the root partition. In a setup that *includes* the systemdisk element when using LVM, this value specifies the size of the LVM partition that contains all specified volumes. This means that the sum of all specified volume sizes plus the sum of the specified freespace for each volume must be smaller than or equal to the size specified with the `oem-systemsize` element. This value is represented by the variable `kiwi_oemrootMB` in the initrd. The specified value can be dynamically overwritten via the `rd.kiwi.install.systemsize` kernel boot option. If `rd.kiwi.install.systemsize` is set to `all` this will unset the size limit and the maximum possible size applies.

oemconfig.oem-unattended

The installation of the image to the target system occurs automatically without requiring user interaction. If multiple possible target devices are discovered, the image is deployed to the first device. `kiwi_oemunattended` in the initrd.

oemconfig.oem-unattended-id

Selects a target disk device for the installation according to the specified device ID. The device ID corresponds to the name of the device for the configured `devicepersistency`. By default, it is the `by-uuid` device name. If no representation exists, for example for ramdisk devices, the UNIX device node can be used to select one. The given name must be present in the device list detected by KIWI.

oemconfig.oem-skip-verify

Disables the checksum verification process after installing of the image to the target disk. The verification process computes the checksum of the image installed to the target. This value is then compared to the initrd embedded checksum generated at build time of the image. Depending on the size of the image and machine power, computing the checksum may take time.

10.3.6 Installation Media Customization

The installation media created for OEM network or CD/DVD deployments can be customized with the `installmedia` section. It is a child section of the `type` element, for example:

```
<installmedia>
```

```
  <initrd action="omit">
    <dracut module="network-legacy"/>
  </initrd>
</installmedia>
```

The `installmedia` is only available for OEM image types that include the request to create an installation media.

The `initrd` child element of `installmedia` lists dracut modules. The element's `action` attribute determines whether the dracut module is omitted (`action="omit"`) or added (`action="add"`). Use `action="set"` to use only the listed modules and nothing else (that is, none of the dracut modules included by default).

10.4 Build a Container Image

Abstract. This page explains how to build a Container Image. It covers the following topics:

- basic configuration explanation
- how to build a Container Image
- how to run it with a Container Runtime

KIWI NG can build native container images from scratch or using existing images. KIWI NG container images are considered to be native, because a KIWI NG tarball image can be loaded directly into container runtimes like Podman, Docker or Containerd, including common container configurations.

The container configuration metadata is supplied to KIWI NG as part of the *Section 1.1.1, "Components of an Image Description"* using the `<containerconfig>` tag. The following configuration metadata can be specified.

`containerconfig` attributes:

- `name`: Specifies the repository name of the container image.
- `tag`: Sets the tag of the container image.
- `maintainer`: Specifies the author of the container. Equivalent to the `MAINTAINER` directive in a `Dockerfile`.

- `user`: Sets the user name or user id (UID) to be used when running `entrypoint` and `subcommand`. Equivalent of the `USER` directive of a `Dockerfile`.
- `workingdir`: Sets the working directory to be used when running `cmd` and `entrypoint`. Equivalent of the `WORKDIR` directive in a `Dockerfile`.

`containerconfig` child tags:

- `subcommand`: Provides the default execution parameters of the container. Equivalent of the `CMD` directive in a `Dockerfile`.
- `labels`: Adds custom metadata to an image using key-value pairs. Equivalent to one or more `LABEL` directives in a `Dockerfile`.
- `expose`: Defines which ports can be exposed to the outside when running this container image. Equivalent to one or more `EXPOSE` directives in a `Dockerfile`.
- `environment`: Sets environment variables using key-value pairs. Equivalent to one or multiple `ENV` directives in a `Dockerfile`.
- `entrypoint`: Sets the binary to use for executing all commands inside the container. Equivalent of the `ENTRYPOINT` directive in a `Dockerfile`.
- `volumes`: Creates mountpoints with the given name and marks them to hold external volumes from the host or from other containers. Equivalent to one or more `VOLUME` directives in a `Dockerfile`.
- `stopsignal`: The stopsignal element sets the system call signal that will be sent to the container to exit. This signal can be a signal name in the format SIG[NAME], for instance SIGKILL, or an unsigned number that matches a position in the kernel's syscall table, for instance 9. The default is SIGTERM if not defined

Other `Dockerfile` directives such as `RUN`, `COPY` or `ADD`, can be mapped to KIWI NG using the *Section 1.1.1, "Components of an Image Description"* script file to run Bash commands, or the *Section 1.1.1, "Components of an Image Description"* to include additional files.

The following example illustrates how to build a container image based on openSUSE Leap:

1. Make sure you have checked out the example image descriptions (see *Section 2.4, "Example Appliance Descriptions"*).

2. Include the `Virtualization/containers` repository into your list (replace the placeholder `<DIST>` with the name of the desired distribution):

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

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

```
$ zypper in umoci skopeo
```

4. Build an image with KIWI NG:

```
$ sudo kiwi-ng system build \
    --description kiwi/build-tests/x86/leap/test-image-docker \
    --set-repo https://download.opensuse.org/distribution/leap/15.6/repo/oss \
    --target-dir /tmp/myimage
```

5. Test the container image.
   First load the new image into your container runtime:

```
$ podman load -i kiwi-test-image-docker.x86_64-1.15.6.docker.tar.xz
```

   Then run the image:

```
$ podman run --rm -it buildsystem /bin/bash
```

10.5 Build a WSL Container Image

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

KIWI NG can build WSL images using the **`appx`** utility. Make sure you have installed the package that provides the command on your build host.

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

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

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

Note

/meta/data

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

10.5.1 Setup of the WSL-DistroLauncher

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

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

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

created_by

Specifies the name of a publisher organization. An appx container must to be signed off with a digital signature. If the image is build in the Open Build Service (OBS), this is done automatically. Outside of OBS, you must o make sure that the given publisher organization name matches the certificate used for signing.

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

application_id
: Specifies an ID name for the container. The name must start with a letter, and only alphanumeric characters are allowed. KIWI NG doesn not validate the specified name string, because there is no common criteria for various the container architectures.

package_version
: Specifies the version identification for the container. KIWI NG validates it against the Microsoft Package Version Numbering (https://docs.microsoft.com/en-us/windows/uwp/publish/package-version-numbering) rules.

launcher
: Specifies the binary file name of the launcher `.exe` file.

Warning

KIWI NG does not check the configuration in `AppxManifest.xml` ifor validity or completeness.

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

1. Check the example image descriptions, see *Section 2.4, "Example Appliance Descriptions"*.
2. Include the `Virtualization/WSL` repository to the list ((replace `<DIST>` with the desired distribution)):

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

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

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

Note

When building images with the Open Build Servic,e make sure to add the packages from the zypper command above to the project configuration via **`osc meta -e prjconf`** along with the line `support: PACKAGE_NAME` for each package that needs to be installed on the Open Build Service worker that runs the KIWI NG build process.

4. Configure the image type:
   Add the following type and container configuration to `kiwi/build-tests/x86/tumbleweed/test-image-wsl/appliance.kiwi`:

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

Warning

If the configured metadata path does not exist, the build will fail. Furthermore, KIWI NG does not check whether the metadata is complete or is valid according to the requirements of the **`WSL-DistroLauncher`**

5. Build the image with KIWI NG:

```
$ sudo kiwi-ng system build \
    --description kiwi/build-tests/x86/tumbleweed/test-image-wsl \
    --set-repo https://download.opensuse.org/tumbleweed/repo/oss \
    --target-dir /tmp/myimage
```

10.5.2 Testing the WSL image

For testing the image, you need a Windows 10 system. Before you proceed, enable the optional feature named `Microsoft-Windows-Subsystem-Linux`. For further details on how to setup the Windows machine, see: Windows Subsystem for Linux (https://docs.microsoft.com/en-us/windows/wsl/about)

10.6 Build KIS Image (Kernel, Initrd, System)

Abstract. This page explains how to build an image that consists out of three components: the kernel an initrd, and an optional root filesystem image. In KIWI NG terminology, this type of image is called KIS.

A KIS image is a collection of image components that are not associated with a dedicated use case. This means that as far as KIWI NG is concerned, it is not known in which environment these components are expected to be used. The predecessor of this image type was called `pxe` under the assumption that the components will be used in a PXE boot environment. However, this assumption is not always true, and the image components may be used in different ways. Because there are so many possible deployment strategies for a `kernel` plus `initrd` and optional `system root filesystem`, KIWI NG provides this as the universal `KIS` type.

The former `pxe` image type still exist, but it is expected to be used only in combination with the legacy `netboot` infrastructure, as described in *Section 11.14, "Build PXE Root File System Image for the legacy netboot infrastructure"*.

To add a KIS build to an appliance, create a `type` element with `image` set to `kis` in the `config.xml` as shown below:

```
<preferences>
    <type image="kis"/>
</preferences>
```

With this image type setup, KIWI NG builds a kernel and initrd not associated with any system root file system. Normally, such an image is only useful with certain custom dracut extensions as part of the image description.

The following attributes of the `type` element are often used when building KIS images:

- `filesystem`: Specifies the root filesystem and triggers the build of an additional filesystem image of that filesystem. The generated kernel command-line options file (append file) then also include a `root=` parameter that references this filesystem image UUID. Whther the information from the append file should be used or not is optional.
- `kernelcmdline`: Specifies kernel command-line options that are part of the generated kernel command-line options file (append file). By default, the append file contains neither information nor the reference to the root UUID, if the `filesystem` attribute is used.

All other attributes of the `type` element that applies to an optional root filesystem image remain in effect in the system image of a KIS image as well.

With the appropriate settings present in `config.xml`, you can use KIWI NG to build the image:

```
$ sudo kiwi-ng --type kis system build \
    --description kiwi/build-tests/x86/tumbleweed/test-image-pxe \
    --set-repo https://download.opensuse.org/tumbleweed/repo/oss \
    --target-dir /tmp/myimage
```

The resulting image components are saved in `/tmp/myimage`. Outside of a deployment infrastructure, the example KIS image can be tested with QEMU as follows:

```
$ sudo qemu
    -kernel /tmp/myimage/*.kernel \
    -initrd /tmp/myimage/*.initrd \
    -append "$(cat /tmp/myimage/*.append) rw" \
    -drive file=/tmp/myimage/kiwi-test-image-pxe.*-1.15.6,if=virtio,driver=raw \
    -serial stdio
```

Note

For testing the components of a KIS image normally requires a deployment infrastructure and a deployment process. An example of a deployment infrastructure using PXE is provided by KIWI NG with the `netboot` infrastructure. However, that netboot infrastructure is no longer developed and only kept for compatibility reasons. For details, see *Section 11.14, "Build PXE Root File System Image for the legacy netboot infrastructure"*

10.7 Build an AWS Nitro Enclave

Abstract. This page explains how to build AWS Nitro Enclaves. It covers the following topics:

- how to build an AWS Nitro Enclave
- how to test the enclave via QEMU

AWS Nitro Enclaves enables customers to create isolated compute environments to further protect and securely process highly sensitive data such as personally identifiable information (PII), healthcare, financial, and intellectual property data within their Amazon EC2 instances. Nitro Enclaves uses the same Nitro Hypervisor technology that provides CPU and memory isolation for EC2 instances. For further details please visit https://aws.amazon.com/ec2/nitro/nitro-enclaves

To add an enclave build to your appliance, create a `type` element with `image` set to `enclave` in the `config.xml` file as shown below:

```
<image schemaversion="8.0" name="kiwi-test-image-nitro-enclave">
  <!-- snip -->
  <profiles>
    <profile name="default" description="CPIO: default profile" import="true"/>
    <profile name="std" description="KERNEL: default kernel" import="true"/>
  </profiles>
  <preferences>
    <type image="enclave" enclave_format="aws-nitro" kernelcmdline="reboot=k
 panic=30 pci=off console=ttyS0 i8042.noaux i8042.nomux i8042.nopnp i8042.dumbkbd
 random.trust_cpu=on rdinit=/sbin/init"/>
    <!-- additional preferences -->
  </preferences>
  <packages type="image" profiles="std">
     <package name="kernel"/>
  </packages>
  <!-- more packages -->
  <!-- snip -->
</image>
```

The following attributes of the `type` element are relevant:

- `enclave_format`: Specifies the enclave target
  As of today only the `aws-nitro` enclave target is supported
- `kernelcmdline`: Specifies the kernel commandline suitable for the enclave
  An enclave is a system that runs completely in RAM loaded from an enclave binary format which includes the kernel, initrd and the kernel commandline suitable for the target system.

With the appropriate settings specified in `config.xml`, you can build an image using KIWI NG:

```
$ sudo kiwi-ng system build \
      --description kiwi/build-tests/x86/rawhide/test-image-nitro-enclave \
      --set-repo https://mirrors.fedoraproject.org/metalink?repo=rawhide&arch=x86_64 \
      --target-dir /tmp/myimage
```

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

```
$ sudo qemu-system-x86_64 \
      -M nitro-enclave,vsock=c \
      -m 4G \
      -nographic \
      -chardev socket,id=c,path=/tmp/vhost4.socket \
      -kernel kiwi-test-image-nitro-enclave.eif
```

The image is now complete and ready to use. Access to the system is possible via ssh through a vsock connection into the guest. To establish a vsock connection it's required to forward the connection through the guest AF_VSOCK socket. This can be done via a ProxyCommand setup of the host ssh as follows:

```
$ vi ~/bin/vsock-ssh.sh

#!/bin/bash
CID=$(echo "$1" | cut -d . -f 1)
socat - VSOCK-CONNECT:$CID:22
```

```
$ vi ~/.ssh/config

host *.vsock
  ProxyCommand ~/bin/vsock-ssh.sh %h
```

After the ssh proxy setup login to the enclave with a custom vsock port as follows:

```
$ ssh root@21.vsock
```

11 Working with Images

Note

This document provides a collection of worksheets which describes the creation and setup of appliances to work within a number of different target environments.

11.1 Deploy ISO Image on an USB Stick

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

- *Section 10.1, "Build an ISO Hybrid Live Image"*

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

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

1. Plug in a USB stick

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

   ```
   $ lsblk
   ```

2. Dump the ISO image on the USB stick:

   ### Warning

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

   ```
   $ dd if={exc_image_base_name}.x86_64-1.15.6.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.

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

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

- *Section 10.1, "Build an ISO Hybrid Live Image"*

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

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

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

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

1. Install the `live-grub-stick` package from software.opensuse.org:

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

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

3. Call the `live-grub-stick` command as follows:
   Assuming "/dev/sdz1" was the FAT32 partition selected from the output of the `lsblk` command:

   ```
   $ sudo live-grub-stick {exc_image_base_name}.x86_64-1.15.6.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.

11.3 Booting a Live ISO Images from Grub2

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

- *Section 10.1, "Build an ISO Hybrid Live Image"*

In KIWI NG, all generated ISO images are created to be hybrid. This means, the image can be used as a CD/DVD or as a disk. This works because the ISO image also has a partition table embedded. With more and more computers delivered without a CD/DVD drive this becomes important. The deployment of such an image onto a disk like an USB stick normally destroys all existing data on this device. It is also not possible to use USB stick as a data storage device. Most USB sticks are pre-formatted with a FAT32 or exFAT Windows File System and to keep the existing data on the stick untouched a different deployment needs to be used.

Fortunately Grub2 supports booting directly from ISO files. It does not matter whether it is installed on your computer's hard drive or on a USB stick. The following deployment process copies the ISO image as an additional file to the USB stick or hard drive. The ability to boot from the disk is configured through a Grub2 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. Almost every Linux distribution supports fat32, and more and more of them also support exFAT. For hard drives, Linux filesystems are also supported.

The dracut initrd system used by KIWI NG provides this feature upstream called as “iso-scan/filename”. Therefore all KIWI NG generated live ISO images supports this deployment mode.

The following procedure shows how to setup Grub2 on your hard drive:

1. Copy the ISO image to a folder of your choice on your hard drive.
2. Add the following code to the “grub.cfg” file:
   Be sure to set the path to the ISO image, you can set your own menu name. The drive identification for Grub2 can be checked at boot time by pressing the ‘c’ key and typing ‘ls’.

   ```
   submenu "Boot from openSUSE ISO" {
        iso_path="(hd0,gpt2)/path/to/openSUSE.iso"
        export iso_path
        loopback loop "$iso_path"
        root=(loop)
        source /boot/grub2/loopback.cfg
        loopback --delete loop
   }
   ```
3. Restart your computer and select the added menu.

For USB sticks, the procedure is identical. You would then install Grub2 on the USB drive and follow the steps above. The use of scripts such as “MultiOS-USB” is strongly recommended.

11.4 Image Description for Amazon EC2

Abstract. This page provides further information for handling Amazon EC2 images built with KIWI NG and references the following articles:

- *Section 10.2, “Build a Virtual Disk Image”*

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

- Xen tools and libraries must be installed
- cloud-init package must be installed
- cloud-init configuration for Amazon must be provided
- Grub bootloader modules for Xen must be installed
- AWS tools must be installed

- Disk size must be set to 10G
- Kernel parameters must allow for xen console

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

1. Software packages

   Make sure to add the following packages to the package list

   Note

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

   ```
   <package name="aws-cli"/>
   <package name="grub2-x86_64-xen"/>
   <package name="xen-libs"/>
   <package name="xen-tools-domU"/>
   <package name="cloud-init"/>
   ```

2. Image Type definition

   Update the oem image type setup as follows

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

3. Cloud Init setup

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

   ```
   users:
   ```

```
  - default

disable_root: true
preserve_hostname: false
syslog_fix_perms: root:root

datasource_list: [ NoCloud, Ec2, None ]

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

cloud_config_modules:
  - 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 (https://github.com/SUSE-Enceladus/ec2imgutils)

11.5 Image Description for Microsoft Azure

Abstract. This page provides further information for handling Azure disk images built with KIWI NG and references the following articles:

- *Section 10.2, "Build a Virtual Disk Image"*

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

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

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

1. Software packages
   Make sure to add the following packages to the package list

   Note

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

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

2. Image Type definition

   Update the oem image type setup as follows

```
<type image="oem"
      filesystem="ext4"
      kernelcmdline="console=ttyS0 rootdelay=300 net.ifnames=0"
      devicepersistency="by-uuid"
      format="vhd-fixed"
      formatoptions="force_size"
      bootpartition="true"
      bootpartsize="1024">
  <bootloader name="grub2" timeout="1"/>
  <size unit="M">30720</size>
  <oemconfig>
      <oem-resize>false</oem-resize>
  </oemconfig>
</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 (https://github.com/SUSE-Enceladus/azurectl)

11.6 Image Description for Google Compute Engine

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

- *Section 10.2, "Build a Virtual Disk Image"*

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

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

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

1. Software packages

Make sure to add the following packages to the package list

Note

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

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

2. Image Type definition

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

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

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

11.7 Image Description for Vagrant

Abstract. This page provides further information for handling Vagrant controlled disk images built with KIWI NG and references the following article:

- *Section 10.2, "Build a Virtual Disk Image"*

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

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

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

In addition, you can use the KIWI NG image description as source for the Open Build Service (https://openbuildservice.org) which allows building and maintaining boxes.

Vagrant expects boxes to be setup in a specific way (for details refer to the Vagrant box documentation (https://www.vagrantup.com/docs/boxes/base.html).), applied to the referenced KIWI NG image description from *Section 10.2, "Build a Virtual Disk Image"*, the following steps are required:

1. Update the image type setup

```
<type image="oem" filesystem="ext4" format="vagrant">
    <bootloader name="grub2" timeout="0"/>
    <vagrantconfig provider="libvirt" virtualsize="42"/>
    <size unit="G">42</size>
    <oemconfig>
        <oem-resize>false</oem-resize>
    </oemconfig>
</type>
```

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

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

```
<type image="oem" filesystem="ext4" format="vagrant">
    <bootloader name="grub2" timeout="0"/>
    <vagrantconfig
```

```
      provider="virtualbox"
      virtualbox_guest_additions_present="true"
      virtualsize="42"
    />
    <size unit="G">42</size>
    <oemconfig>
        <oem-resize>false</oem-resize>
    </oemconfig>
</type>
```

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

2. Add mandatory packages

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

3. Add additional packages

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

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

   Otherwise, you must add `rsync`:

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

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

4. Add Vagrant user

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

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

5. Configure SSH, the default shared folder and sudo permissions

Vagrant expects that it can login as the user `vagrant` using an insecure public key . Furthermore, vagrant also usually uses `/vagrant` as the default shared folder and assumes that the `vagrant` user can invoke commands via **`sudo`** without having to enter a password. This can be achieved using the function `baseVagrantSetup` in `config.sh`:

```
baseVagrantSetup
```

6. Additional customizations:
   Additionally to `baseVagrantSetup`, you might want to also ensure the following:

   - If you have installed the Virtualbox guest additions into your box, then also load the `vboxsf` kernel module.
   - When building boxes for libvirt, then ensure that the default wired networking interface is called `eth0` and uses DHCP. This is necessary since libvirt uses `dnsmasq` to issue IPs to the VMs. This step can be omitted for Virtualbox boxes.

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

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

Note

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

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

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

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

11.7.1 Customizing the embedded Vagrantfile

Warning

This is an advanced topic and not required for most users

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

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

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

```
<type image="oem" filesystem="ext4" format="vagrant">
    <bootloader name="grub2" timeout="0"/>
    <vagrantconfig
      provider="libvirt"
      virtualsize="42"
      embedded_vagrantfile="MyVagrantfile"
    />
    <size unit="G">42</size>
    <oemconfig>
        <oem-resize>false</oem-resize>
    </oemconfig>
</type>
```

The option to provide a custom `Vagrantfile` can be combined with the usage of *profiles* (see *Section 7.4, "Image Profiles"*), so that certain builds can use the automatically generated `Vagrantfile` (in the following example that is the Virtualbox build) and others get a customized one (the libvirt profile in the following example):

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

<image schemaversion="8.0" name="{exc_image_base_name}">
  <!-- description goes here -->
  <profiles>
    <profile name="libvirt" description="Vagrant Box for Libvirt"/>
    <profile name="virtualbox" description="Vagrant Box for VirtualBox"/>
  </profiles>
```

```
  <!-- general preferences go here -->

  <preferences profiles="libvirt">
    <type
      image="oem"
      filesystem="ext4"
      format="vagrant">
        <bootloader name="grub2" timeout="0"/>
        <vagrantconfig
          provider="libvirt"
          virtualsize="42"
          embedded_vagrantfile="LibvirtVagrantfile"
        />
        <size unit="G">42</size>
        <oemconfig>
            <oem-resize>false</oem-resize>
        </oemconfig>
  </type>
  </preferences>
  <preferences profiles="virtualbox">
    <type
      image="oem"
      filesystem="ext4"
      format="vagrant">
        <bootloader name="grub2" timeout="0"/>
        <vagrantconfig
          provider="virtualbox"
          virtualbox_guest_additions_present="true"
          virtualsize="42"
        />
        <size unit="G">42</size>
        <oemconfig>
            <oem-resize>false</oem-resize>
        </oemconfig>
    </type>
  </preferences>

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

11.8 Image Description Encrypted Disk

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

- *Section 10.2, "Build a Virtual Disk Image"*

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

Update the KIWI NG image description as follows:

1. Software packages

   Make sure to add the following package to the package list

   Note

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

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

2. Image Type definition

   Update the oem image type setup as follows

   Full disk encryption including `/boot`:

   ```
   <type image="oem" filesystem="ext4" luks="linux" bootpartition="false">
       <oemconfig>
           <oem-resize>false</oem-resize>
       </oemconfig>
   </type>
   ```

   Encrypted root partition with an unencrypted extra `/boot` partition:

   ```
   <type image="oem" filesystem="ext4" luks="linux" bootpartition="true">
       <oemconfig>
           <oem-resize>false</oem-resize>
       </oemconfig>
   </type>
   ```

Note

The value for the `luks` attribute sets the master passphrase for the LUKS keyring. Therefore the XML description becomes security critical and should only be readable by trustworthy people. Alternatively the credentials information can be stored in a key file and referenced as:

```
<type luks="file:///path/to/keyfile"/>
```

11.9 Deploy and Run System in a RamDisk

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

- *Section 10.3, "Build an Expandable Disk Image"*

If a machine should run the OS completely in memory without the need for any persistent storage, the approach to deploy the image into a ramdisk serves this purpose.

Note

brd support

For this functionality to work, brd (Block Ramdisk Driver) must be supported. If brd is available as a kernel module (as opposed to being compiled in the kernel), you must include the driver into the initrd. To do this, create (or edit) the `config.sh` file and add the following command to it:

```
echo 'add_drivers+=" brd "' > /etc/dracut.conf.d/10-brd.conf
```

KIWI NG allows to create a bootable ISO image which deploys the image into a ramdisk and activates that image with the following oem type definition:

```
<type image="oem" filesystem="ext4" installiso="true" initrd_system="dracut"
 installboot="install" kernelcmdline="rd.kiwi.ramdisk">
    <bootloader name="grub2" timeout="1"/>
```

```
    <oemconfig>
        <oem-skip-verify>true</oem-skip-verify>
        <oem-unattended>true</oem-unattended>
        <oem-unattended-id>/dev/ram1</oem-unattended-id>
        <oem-swap>false</oem-swap>
        <oem-multipath-scan>false</oem-multipath-scan>
        <oem-ramdisk-size>2048000</oem-ramdisk-size>
     </oemconfig>
 </type>
```

The type specification above builds an installation ISO image which deploys the System Image into the specified ramdisk device (/dev/ram1). The setup of the ISO image boots with a short boot timeout of 1sec and just runs through the process without asking any questions. In a ramdisk deployment the optional target verification, swap space and multipath targets are out of scope and therefore disabled.

The configured size of the ramdisk via `oem-ramdisk-size` specifies the kB size of the OS disk and it must be at least of the size of the System Image. The disk size can be configured dynamically with the following value in the kernelcmdline attribute:

- ramdisk_size=kbyte-value

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

```
$ sudo qemu -cdrom \
      {exc_image_base_name}.x86_64-1.15.6.install.iso \
      -serial stdio
```

Note

Enough Main Memory

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

Like all other oem KIWI NG images, also the ramdisk setup supports all the deployments methods as explained in *Section 10.3.1, "Deployment Methods"* This means it's also possible to dump the ISO image on a USB stick let the system boot from it and unplug the stick from the machine because the system was deployed into RAM

Note

Limitations Of RamDisk Deployments

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

11.10 Custom Disk Partitions

Abstract. This page provides details about the opportunities and limitations to customize the partition table in addition to the volume management settings from *Section 11.11, "Custom Disk Volumes"*.

KIWI NG has its own partitioning schema which is defined according to several different user configurations: boot firmware, boot partition, expandable layouts, etc. Those supported features have an impact on the partitioning schema.

MBR or GUID partition tables are not flexible, carry limitations and are tied to some specific disk geometry. Because of that the preferred alternative to disk layouts based on traditional partition tables is using flexible approaches like logic volumes.

However, on certain conditions additional entries to the low level partition table are needed. For this purpose the `<partitions>` section exists and allows to add custom entries like shown in the following example:

```
<partitions>
    <partition name="var" size="100" mountpoint="/var" filesystem="ext3"/>
</partitions>
```

Each `<partition>` entry puts a partition of the configured size in the low level partition table, creates a filesystem on it and includes it to the system's fstab file. If parts of the root filesystem are moved into its own partition like it's the case in the above example, this partition will also contain the data that gets installed during the image creation process to that area.

The following attributes must/can be set to configured a partition entry:

name="identifier"

Mandatory name of the partition as handled by KIWI NG.

Note

There are the following reserved names which cannot be used because they are already represented by existing attributes: `root`, `readonly`, `boot`, `prep`, `spare`, `swap`, `efi_csm` and `efi`.

partition_name="name"

Optional name of the partition as it appears when listing the table contents with tools like `gdisk`. If no name is set KIWI NG constructs a name of the form `p.lx(identifier_from_name_attr)`

partition_type="type_identifier"

Optional partition type identifier as handled by KIWI NG. Allowed values are `t.linux` and `t.raid`. If not specified `t.linux` is the default.

size="size_string"

Mandatory size of the partition. A size string can end with `M` or `G` to indicate a mega-Byte or giga-Byte value. Without a unit specification mega-Byte is used.

mountpoint="path"

Optional mountpoint to mount the partition in the system.

filesystem="btrfs|ext2|ext3|ext4|squashfs|xfs

Optional filesystem configuration to create one of the supported filesystems on the partition.

label="string"

Optional filesystem label if a filesystem is provided. If no label is specified the `name` identifier is set as filesystem label

clone="number"

Optional setting to indicate that this partition should be cloned `number` of times. A clone partition is content wise an exact byte for byte copy of the origin. However, to avoid conflicts at boot time the UUID of any cloned partition will be made unique. In the sequence of partitions, the clone(s) will always be created first followed by the partition considered the origin. The origin partition is the one that will be referenced and used by the system

part_id="number"

Specify the partition number for this partition. This is the number used for the partition in the partition table. The value impacts the name of the unix device node associated with the partition. This means if the disk name is e.g. `/dev/sda` and `part_id="42"` is set the unix node for this partition will be `/dev/sda42`. Please note that the number has no impact on the geometry or order of partitions as they get created. The partitions in the `<partitions>` section still gets created in the order as they are listed. Please also note if custom partition ids are used it is strongly recommended that they are used across all partitions created by kiwi. The reason for this is that kiwi uses an incremental counter which by default matches with the number of the partition in the table. If custom partition id's are not used for all partitions this results in a mix of numbers, which works but is not obviously visible by reading the image description. Because of that also see the information about the `<type>` attributes `eficsmpart_id`, `efipart_id`, `rootpart_id` and `bootpart_id`

Despite the customization options of the partition table shown above there are the following limitations:

1. By default the root partition is always the last one
   Disk imags build with KIWI NG are designed to be expandable. For this feature to work the partition containing the system rootfs must always be the last one. If this is unwanted for some reason KIWI NG offers an opportunity for one extra/spare partition with the option to be also placed at the end of the table. For details lookup `spare_part` in *Section 8.1, "Image Description Elements"*

2. By default there are no gaps in the partition table
   The way partitions are configured is done such that there are no gaps in the table of the image. However, leaving some space free at the **end** of the partition geometry is possible in the following ways:

   - **Deploy with unpartitioned free space.**

To leave space unpartitioned on first boot of a disk image it is possible to configure an `<oem-systemsize>` which is smaller than the disk the image gets deployed to. Details about this setting can be found in *Section 8.1, "Image Description Elements"*

- **Build with unpartitioned free space.**
  To leave space unpartitioned at build time of the image it is possible to disable `<oem-resize>` and configure an `<oem-systemsize>` which is smaller than the kiwi calculated disk size or the fixed setting for the disk size via the `size>` element.

- **Build with unpartitioned free space.**
  Setting some unpartitioned free space on the disk can be done using the `unpartitioned` attribute of `size` element in type's section. For details see *Section 10.2.2, "Modifying the Size of the Image"*

- **Resize built image adding unpartitioned free space.**
  A built image can be resized by using the `kiwi-ng image resize` command and set a new extended size for the disk. See KIWI NG commands docs *Section 4.8, "kiwi-ng image resize"*.

11.11 Custom Disk Volumes

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

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

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

```
<image schemaversion="8.0" name="openSUSE-Leap-15.1">
  <type image="oem" filesystem="btrfs">
    <systemdisk name="vgroup" preferlvm="true">
      <volume name="usr/lib" size="1G" label="library"/>
      <volume name="@root" freespace="500M"/>
      <volume name="etc_volume" mountpoint="etc" copy_on_write="false"/>
      <volume name="bin_volume" size="all" mountpoint="/usr/bin" quota="2G"/>
    </systemdisk>
  </type>
</image>
```

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

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

Note

Special name for the root volume

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

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

In addition to the custom size of the root volume it's also possible to setup the name of the root volume as follows:

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

If no name for the root volume is specified the default name: **LVRoot** applies.

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

- `filesystem_check`: Optional attribute to indicate that this filesystem should perform the validation to become filesystem checked. The actual constraints if the check is performed or not depends on systemd and filesystem specific components. If not set or set to `false` no system component will be triggered to run an eventual filesystem check, which results in this filesystem to be never checked. The latter is the default.

- `arch`: Optional attribute to create the volume only if it matches the specified host architecture. Multiple architecture names can be specified as comma separated list.

Warning

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

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

- `name`: The volume group name, by default `kiwiVG` is used. This setting is only relevant for LVM volumes.

- `preferlvm`: Boolean value instructing KIWI NG to prefer LVM even if the used filesystem has its own volume management system.

11.12 Partition Clones

Abstract. This page provides details about the partition clone feature and its use cases

KIWI NG allows to create block level clones of certain partitions used in the image. Clones can be created from the `root`, `boot` and any other partition listed in the `<partitions>` element.

A partition clone is a simple byte dump from one block storage device to another. However, this would cause conflicts during boot of the system because all unique identifiers like the UUID of a filesystem will no longer be unique. The clone feature of KIWI NG takes care of this part and re-creates the relevant unique identifiers per cloned partition. KIWI NG allows this also for complex partitions like LVM, LUKS or RAID.

The partition clone(s) will always appear first in the partition table, followed by the origin partition. The origin partition is the one whose identifier will be referenced and used by the system. By default no cloned partition will be mounted or used by the system at boot time.

Let's take a look at the following example:

```
<type image="oem" root_clone="1" boot_clone="1" firmware="uefi" filesystem="xfs"
 bootpartition="true" bootfilesystem="ext4">
    <partitions>
        <partition name="home" size="10" mountpoint="/home" filesystem="ext3" clone="2"/>
    </partitions>
    <oemconfig>
        <oem-systemsize>2048</oem-systemsize>
        <oem-resize>false</oem-resize>
    </oemconfig>
    <size unit="G">100</size>
</type>
```

With the above setup KIWI NG will create a disk image that contains the following partition table:

```
Number  Start (sector)    End (sector)  Size       Code  Name
   1            2048            6143   2.0 MiB     EF02  p.legacy
   2            6144           47103   20.0 MiB    EF00  p.UEFI
   3           47104          661503   300.0 MiB   8300  p.lxbootclone1
   4          661504         1275903   300.0 MiB   8300  p.lxboot
   5         1275904         1296383   10.0 MiB    8300  p.lxhomeclone1
   6         1296384         1316863   10.0 MiB    8300  p.lxhomeclone2
   7         1316864         1337343   10.0 MiB    8300  p.lxhome
   8         1337344         3864575   2 GiB       8300  p.lxrootclone1
   9         3864576         6287326   2 GiB       8300  p.lxroot
```

When booting the system only the origin partitions `p.lxboot`, `p.lxroot` and `p.lxhome` will be mounted and visible in e.g. `/etc/fstab`, the bootloader or the initrd. Thus partition clones are present as a data source but are not relevant for the operating system from a functional perspective.

As shown in the above example there is one clone request for root and boot and a two clone requests for the home partition. KIWI NG does not sanity- check the provided number of clones (e.g. whether your partition table can hold that many partitions).

Warning

There is a limit how many partitions a partition table can hold. This also limits how many clones can be created.

It's important when using the `root_clone` attribute to specify a size for the part of the system that represents the root partition. As of today KIWI NG does not automatically divide the root partition into two identical pieces. In order to create a clone of the partition a size specification is required. In the above example the size for root is provided via the `oem-systemsize` element. Using a root clone and fixed size values has the following consequences:

1. The resize capability must be disabled. This is done via `oem-resize` element. The reason is that only the last partition in the partition table can be resized without destroying data. If there is a clone of the root partition it should never be resized because then the two partitions will be different in size and no longer clones of each other

2. There can be unpartitioned space left. In the above example the overall disk size is set to 100G. The sum of all partition sizes will be smaller than this value and there is no resize available anymore. Depending on the overall size setup for the disk this will leave unpartitioned space free on the disk.

11.12.1 Use Case

Potential use cases for which a clone of one or more partitions is useful include among others:

Factory Resets:

Creating an image with the option to rollback to the state of the system at deployment time can be very helpful for disaster recovery

System Updates with Rollbacks e.g A/B:

Creating an image which holds extra space allowing to rollback modified data can make a system more robust. For example in a simple A/B update concept, partition A would get updated but would flip to B if A is considered broken after applying the update.

Note

Most probably any use case based on partition clones requires additional software to manage them. KIWI NG provides the option to create the clone layout but it does not provide the software to implement the actual use case for which the partition clones are needed.

Developers writing applications based on a clone layout created with KIWI NG can leverage the metadata file `/config.partids`. This file is created at build time and contains the mapping between the partition `name` and the actual partition number in the partition table. For partition clones, the following naming convention applies:

```
kiwi_(name)PartClone(id)="(partition_number)"
```

The `(name)` is either taken from the `name` attribute of the `<partition>` element or it is a fixed name assigned by KIWI NG. There are the following reserved partition names for which cloning is supported:

- root
- readonly
- boot

For the mentioned example this will result in the following `/config.partids`:

```
kiwi_BiosGrub="1"
kiwi_EfiPart="2"
kiwi_bootPartClone1="3"
kiwi_BootPart="4"
kiwi_homePartClone1="5"
kiwi_homePartClone2="6"
kiwi_HomePart="7"
kiwi_rootPartClone1="8"
kiwi_RootPart="9"
```

11.13 Setting Up a Network Boot Server

Abstract. This page provides general information how to setup a network boot server that provides all services needed for the PXE boot protocol

To be able to deploy a system through the PXE boot protocol, you need to set up a network boot server providing the services DHCP and tftp. With `dnsmasq` an utility exists which allows to setup all needed services at once:

11.13.1 Installing and Configuring DHCP and TFTP with dnsmasq

The following instructions can only serve as an example. Depending on your network structure, the IP addresses, ranges and domain settings needs 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 dnsmasq to work as DHCP and TFTP server.

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

```
# Don't function as a DNS server.
port=0

# Log information about DHCP transactions.
log-dhcp

# Set the root directory for files available via FTP,
# usually "/srv/tftpboot":
tftp-root=TFTP_ROOT_DIR

enable-tftp

dhcp-range=BOOT_SERVER_IP,proxy
```

In the next step it's required to decide for the boot method. There is the PXE loader provided via pxelinux.0 from the syslinux package and there is the GRUB loader provided via the grub package.

Note

Placeholders

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

2.1. insert the following content to use pxelinux.0:

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

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

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

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

Note

On boot of a network client with that configuration the default pxelinux.0 config file is expected at `TFTP_ROOT_DIR/pxelinux.cfg/default`

2.2. insert the following content to use grub:

```
# The boot filename, Server name, Server Ip Address
dhcp-boot=boot/grub2/i386-pc/core.0,,BOOT_SERVER_IP
```

When using grub the referenced dhcp-boot grub module must be genereated. To do this change the directory to `TFTP_ROOT_DIR` and create the `setvars.conf` with the following content:

```
set root=(tftp)
set net_default_server=BOOT_SERVER_IP
set prefix=boot/grub2
```

Now call the following commands to create the grub module

```
$ grub2-mknetdir --net-directory=TFTP_ROOT_DIR --subdir=boot/grub2
$ grub2-mkimage -O i386-pc-pxe \
```

```
    --output boot/grub2/i386-pc/core.0 \
    --prefix=/boot/grub2 \
    -c setvars.conf \
  pxe tftp
```

Note

On boot of a network client with that configuration the grub config file is expected at `TFTP_ROOT_DIR/boot/grub2/grub.cfg`

- Run the dnsmasq server by calling:

```
systemctl start dnsmasq
```

11.14 Build PXE Root File System Image for the legacy netboot infrastructure

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

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

PXE (https://en.wikipedia.org/wiki/Preboot_Execution_Environment) 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 (https://en.wikipedia.org/wiki/Trivial_File_Transfer_Protocol) protocol to download a Kernel and boot instructions. Contrary to other images built with KIWI NG, a PXE image consists of separate boot, kernel and root filesystem images, since those images need to be made available in different locations on the PXE boot server.

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

Many different deployment strategies are possible, e.g root over NBD (https://en.wikipedia.org/wiki/Network_block_device) (network block device), AoE (https://en.wikipedia.org/wiki/ATA_over_Ethernet) (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.

1. Make sure you have checked out the example image descriptions, see *Section 2.4, "Example Appliance Descriptions"*.

2. Build the image with KIWI NG:

```
$ sudo kiwi-ng --profile Flat system build \
    --description kiwi/build-tests/x86/tumbleweed/test-image-pxe \
    --set-repo https://download.opensuse.org/tumbleweed/repo/oss \
    --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 /srv/tftpboot/boot/initrd
$ cp *.kernel /srv/tftpboot/boot/linux
```

5. Copy the system image and its SHA256 sum to `/srv/tftpboot/image`:

```
$ cp kiwi-test-image-pxe.x86_64-1.15.6 /srv/tftpboot/image
$ cp kiwi-test-image-pxe.x86_64-1.15.6.sha256 /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
```

7. Create the image client configuration file:

```
$ vi /srv/tftpboot/KIWI/config.default

IMAGE=/dev/ram1;kiwi-test-image-pxe.x86_64;1.15.6;192.168.100.2;4096
UNIONFS_CONFIG=/dev/ram2,/dev/ram1,overlay
```

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

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

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

11.14.1 PXE Client Setup Configuration

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"
```

```
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.

11.14.1.1 Setup Client with Remote Root

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

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

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

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

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

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

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

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

11.14.1.2 Setup Client with System on Local Disk

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

Note

In the referenced x86/tumbleweed/test-image-pxe 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-tumbleweed"/>
```

```
IMAGE="/dev/sda2;kiwi-test-image-pxe.x86_64;1.15.6;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.

11.14.1.3 Setup Client with System on Local MD RAID Disk

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

Note

In the referenced x86/tumbleweed/test-image-pxe 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-tumbleweed"/>
```

```
RAID="1;/dev/sda;/dev/sdb"
```

```
IMAGE="/dev/md1;kiwi-test-image-pxe.x86_64;1.15.6;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.

11.14.1.4 Setup Loading of Custom Configuration File(s)

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

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

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

11.14.1.5 Setup Client to Force Reload Image

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

```
RELOAD_IMAGE=1
```

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

11.14.1.6 Setup Client to Force Reload Configuration Files

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

```
RELOAD_CONFIG=1
```

By default only configuration files which has changed according to their checksum 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

11.14.1.7 Setup Client for Reboot After Deployment

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

```
REBOOT_IMAGE=1
```

11.14.1.8 Setup custom kernel boot options

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

```
KIWI_KERNEL_OPTIONS="nomodeset"
```

Note

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

11.14.1.9 Setup a Custom Boot Timeout

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

```
KIWI_BOOT_TIMEOUT="10"
```

Note

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

11.14.1.10 Setup a Different Download Protocol and Server

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

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

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

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

Note

Once configured all downloads except for kernel and initrd are now controlled by the given server and protocol. You need to make sure that this server provides the same directory and file structure as initially provided by the `kiwi-pxeboot` package

11.15 Booting a Root Filesystem from Network

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

- *Section 10.6, "Build KIS Image (Kernel, Initrd, System)"*

In KIWI NG, the `kiwi-overlay` dracut module can be used to boot from a remote exported root filesystem. The exported device is visible as block device on the network client. KIWI NG supports the two export backends `NBD` (Network Block Device) and `AoE` (ATA over Ethernet) for this purpose. A system that is booted in this mode will read the contents of the root filesystem from a remote location and targets any write action into RAM by default. The kernel cmdline option `rd.root.overlay.write` can be used to specify an alternative device to use for writing. The two layers (read and write) are combined using the `overlayfs` filesystem.

For remote boot of a network client, the PXE boot protocol is used. This functionality requires a network boot server setup on the system. Details how to setup such a server can be found in *Section 11.13, "Setting Up a Network Boot Server"*.

Before the KIS image can be build, the following configuration step is required:

- Create dracut configuration to include the `kiwi-overlay` module

```
$ cd kiwi/build-tests/x86/tumbleweed/test-image-pxe
$ mkdir -p root/etc/dracut.conf.d
$ cd root/etc/dracut.conf.d
```

```
$ echo 'add_dracutmodules+=" kiwi-overlay "' > overlay.conf
```

Now the KIS image can be build as shown in *Section 10.6, "Build KIS Image (Kernel, Initrd, System)"*. After the build, the following configuration steps are required to boot from the network:

1. Copy initrd/kernel from the KIS build to the PXE server
   The PXE boot process loads the configured kernel and initrd from the PXE server. For this reason, the following files must be copied to the PXE server as follows:

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

2. Export Root FileSystem to the Network
   Access to the root filesystem is implemented using either the AoE or the NBD protocol. This requires the export of the root filesystem image as remote block device:

   Export via AoE:

   Install the `vblade` package on the system which is expected to export the root filesystem

   Note

   Not all versions of AoE are compatible with any kernel. This means the kernel on the system exporting the root filesystem image must provide a compatible aoe kernel module compared to the kernel used inside of the root filesystem image.

   Once done, export the filesystem from the KIS build above as follows:

   ```
   $ vbladed 0 1 IFACE {exc_image_base_name}.x86_64-1.15.6
   ```

   The above command exports the given filesystem image file as a block storage device to the network of the given IFACE. 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`.

Note

Only machines in the same network of the given INTERFACE can see the exported block device.

Export via NBD:

Install the `nbd` package on the system which is expected to export the root filesystem Once done, export the filesystem from the KIS build above as follows:

```
$ losetup /dev/loop0 {exc_image_base_name}.x86_64-1.15.6

$ vi /etc/nbd-server/config

  [generic]
      user = root
      group = root
  [export]
      exportname = /dev/loop0

$ nbd-server
```

3. Setup boot entry in the PXE configuration

Note

The following step assumes that the pxelinux.0 loader has been configured on the boot server to boot up network clients

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

Using NBD:

```
LABEL Overlay-Boot
    kernel boot/linux
    append initrd=boot/initrd root=overlay:nbd=server-ip:export
```

The boot parameter `root=overlay:nbd=server-ip:export` specifies the NBD server IP address and the name of the export as used in `/etc/nbd-server/config`

Using AoE:

```
LABEL Overlay-Boot
    kernel boot/linux
    append initrd=boot/initrd root=overlay:aoe=AOEINTERFACE
```

The boot parameter `root=overlay:aoe=AOEINTERFACE` specifies the interface name as it was exported by the **`vbladed`** command

4. Boot from the Network

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

```
$ sudo qemu -boot n
```

11.16 Booting a Live ISO Image from Network

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

- *Section 10.1, "Build an ISO Hybrid Live Image"*

In KIWI NG, live ISO images can be configured to boot via the PXE boot protocol. This functionality requires a network boot server setup on the system. Details how to setup such a server can be found in *Section 11.13, "Setting Up a Network Boot Server"*.

After the live ISO was built as shown in *Section 10.1, "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 kiwi-test-image-live.x86_64-1.15.6.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 must be provided by either `ftp`, `http`, `https` or `dolly`. The most simple method is to setup a FTP server e.g. `vsftpd` and copy the live ISO file to the data directory:

```
$ mkdir -p /srv/ftp/image
$ cp kiwi-test-image-live.x86_64-1.15.6.iso /srv/ftp/image
```

Note

Check if the image can be downloaded via: `wget ftp://IP/image/kiwi-test-image-live.x86_64-1.15.6.iso` before the next step.

3. Setup live ISO boot entry in PXE configuration

Note

The following step assumes that the pxelinux.0 loader has been configured on the boot server to boot up network clients

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 ip=dhcp root=live:ftp://IP/image/kiwi-test-image-
live.x86_64-1.15.6.iso
```

- The `ip=` parameter controls how the dracut network module sets up the network. Several options exists to control how the network interface should be setup. Please consult the dracut manual for further information.
- The boot parameter `root=live:PROTOCOL://IP/PATH...` specifies the remote end-point and protocol to find the live ISO file. So far `ftp`, `http`, `https` and `dolly` are supported.

4. Boot from the Network

   Within the network which has access to the PXE server and the IP in the root= option, any network client can now boot the live system. A test based on QEMU is done as follows:

   ```
   $ qemu -boot n
   ```

11.16.1 Available Remote Boot Options

There are the following kernel boot options available to control the behavior of the remote boot process.

rd.kiwi.live.curl_options=

Options passed along to the curl call

rd.kiwi.live.dolly_options=

Options passed along to the dolly call

rd.kiwi.live.system=

Block device to use for the system. By default this is set to `/dev/ram0`

ramdisk_size=bytes

Size of the ramdisk in bytes. By default this is set to 2097152 (2G)

rd.kiwi.live.reset

Force reset of the live system. This option only makes sense if the live system device (rd.kiwi.live.system) points to a persistent storage device. In this case KIWI NG loads the system only once and does not overwrite it unless a reset is requested.

11.16.2 Persistent Live System

The remote boot process of a live ISO image, places the ISO file into a ramdisk by default. This means all data lives in memory and is not persistent. In order to boot up the live system from a remote location and keep it on a persistent storage, it's required to pass the `rd.kiwi.live.system` boot option with the device name pointing to that persistent storage.

Warning

All data on the device given via `rd.kiwi.live.system` will be wiped

A test based on QEMU is done as follows:

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 ip=dhcp root=live:ftp://IP/image/kiwi-test-
image-live.x86_64-1.15.6.iso ramdisk_size=3545728 rd.kiwi.live.system=/dev/sda
 rd.live.overlay.persistent rd.live.overlay.cowfs=xfs
```

- The `rd.live.overlay.persistent` and `rd.live.overlay.cowfs=xfs` options are standard KIWI NG live ISO options to control if and how a persistent write partition should be created. The options only take an effect when booting into a persistent storage device.

Next create a persistent storage disk of 3G and attach it to the QEMU instance.

```
$ qemu-img create mydisk.raw 3G
$ qemu -boot n -hda mydisk.raw
```

The live system will be deployed once to the locally attached disk and boots from it. Any subsequent boot process will not modify the local disk unless `rd.kiwi.live.reset` is passed on the kernel command line. If deployed there is also no need for the network anymore and the system could also boot standalone.

11.17 Setting Up YaST at First Boot

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

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

```
yast autoyast
```

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

1. Edit the KIWI NG XML file and add the following package to the `<packages type="image">` section:

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

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

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

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

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

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

11.18 Add or Update the Fstab File

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

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

Possible use cases are for example:

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

Note

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

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

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

Providing an `/etc/fstab.append` file

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

Providing an `/etc/fstab.patch` file

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

Providing an `/etc/fstab.script` file

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

Note

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

11.19 Building Images with Profiles

KIWI NG supports so-called *profiles* inside the XML image description. Profiles act as namespaces for additional settings to be applied on top of the defaults. For further details, see *Section 7.4, "Image Profiles"*.

11.19.1 Local Builds

To execute local KIWI NG builds with a specific, selected profile, add the command line flag `--profile=$PROFILE_NAME`:

```
$ sudo kiwi-ng --type oem --profile libvirt system build \
      --description kiwi/build-tests/x86/leap/test-image-vagrant \
      --set-repo https://download.opensuse.org/distribution/leap/15.6/repo/oss \
      --target-dir /tmp/myimage
```

Consult the manual page of `kiwi` for further details: *Section 4.1.1, "SYNOPSIS"*.

11.19.2 Building with the Open Build Service

The Open Build Service (OBS) support profiles via the multibuild (https://openbuildservice.org/help/manuals/obs-user-guide/cha.obs.multibuild.html) feature.

To enable and use the profiles, follow these steps:

1. Add the following XML comment to your `config.xml`:

   ```
   <!-- OBS-Profiles: @BUILD_FLAVOR@ -->
   ```

   It must be added before the opening `<image>` element and after the `<?xml?>` element, e.g.:

   ```
   <?xml version="1.0" encoding="utf-8"?>
   <!-- OBS-Profiles: @BUILD_FLAVOR@ -->
   <image schemaversion="8.0" name="kiwi-test-image-vagrant">
     <!-- snip -->
   </image>
   ```

2. Add a file `_multibuild` into your package's repository with the following contents:

   ```
   <multibuild>
     <flavor>profile_1</flavor>
     <flavor>profile_2</flavor>
   </multibuild>
   ```

> Add a line `<flavor>$PROFILE</flavor>` for each profile that you want OBS to build.

Note, by default, OBS excludes the build **without** any profile enabled.

Running a build of a multibuild enabled repository via `osc` can be achieved via the `-M $PROFILE` flag:

```
$ osc build -M $PROFILE
```

11.20 Building in the Open Build Service

Note

Abstract

This document gives a brief overview how to build images with KIWI NG in version 10.2.35 inside of the Open Build Service. A tutorial on the Open Buildservice itself can be found here: https://en.opensuse.org/openSUSE:Build_Service_Tutorial

The next generation KIWI NG is fully integrated with the Open Build Service. In order to start it's best to checkout one of the integration test image build projects from the base Testing project `Virtualization:Appliances:Images:Testing_$ARCH:$DISTRO` at:

https://build.opensuse.org

For example the test images for SUSE on x86 can be found here (https://build.opensuse.org/project/show/Virtualization:Appliances:Images:Testing_x86:leap).

11.20.1 Advantages of using the Open Build Service (OBS)

The Open Build Service offers multiple advantages over running KIWI NG locally:

- OBS will host the latest successful build for you without having to setup a server yourself.
- As KIWI NG is fully integrated into OBS, OBS will automatically rebuild your images if one of the included packages or one of its dependencies or KIWI NG itself get updated.
- The builds will no longer have to be executed on your own machine, but will run on OBS, thereby saving you resources. Nevertheless, if a build fails, you get a notification via email (if enabled in your user's preferences).

11.20.2 Differences Between Building Locally and on OBS

Note, there is a number of differences when building images with KIWI NG using the Open Build Service. Your image that build locally just fine, might not build without modifications.

The notable differences to running KIWI NG locally include:

- OBS will pick the KIWI NG package from the repositories configured in your project, which will most likely not be the same version that you are running locally. This is especially relevant when building images for older versions like SUSE Linux Enterprise. Therefore, include the custom appliances repository as described in the following section: Recommendations (#obs-recommended-settings).

- When KIWI NG runs on OBS, OBS will extract the list of packages from `config.xml` and use it to create a build root. In contrast to a local build (where your distributions package manager will resolve the dependencies and install the packages), OBS will **not** build your image if there are multiple packages that could be chosen to satisfy the dependencies of your packages . This shows errors like this:

  ```
  unresolvable: have choice for SOMEPACKAGE: SOMEPAKAGE_1 SOMEPACKAGE_2
  ```

  This can be solved by explicitly specifying one of the two packages in the project configuration via the following setting:

  ```
  Prefer: SOMEPACKAGE_1
  ```

  Place the above line into the project configuration, which can be accessed either via the web interface (click on the tab `Project Config` on your project's main page) or via `osc meta -e prjconf`.

  #### Warning

  We strongly encourage you to remove your repositories from `config.xml` and move them to the repository configuration in your project's settings. This usually prevents the issue of having the choice for multiple package version and results in a much smoother experience when using OBS.

- By default, OBS builds only a single build type and the default profile. If your appliance uses multiple build types, put each build type into a profile, as OBS cannot handle multiple build types.

There are two options to build multiple profiles on OBS:

1. Use the `<image>` element and add it bellow the XML declaration (`<?xml ..?>`):

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

<!-- OBS-Profiles: foo_profile bar_profile -->

<image schemaversion="8.0" name="openSUSE-Leap-15.1">
  <!-- image description with the profiles foo_profile and bar_profile
</image>
```

2. Use the multibuild (https://openbuildservice.org/help/manuals/obs-user-guide/cha.obs.multibuild.html) feature.

The first option is simpler to use, but has the disadvantage that your appliances are built sequentially. The `multibuild` feature allows to build each profile as a single package, thereby enabling parallel execution, but requires an additional `_multibuild` file. For the above example `config.xml` would have to be adapted as follows:

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

<!-- OBS-Profiles: @BUILD_FLAVOR@ -->

<image schemaversion="8.0" name="openSUSE-Leap-15.1">
  <!-- image description with the profiles foo_profile and bar_profile
</image>
```

The file `_multibuild` would have the following contents:

```
<multibuild>
  <flavor>foo_profile</flavor>
  <flavor>bar_profile</flavor>
</multibuild>
```

- Subfolders in OBS projects are ignored by default by **`osc`** and must be explicitly added via **`osc add $FOLDER`**. Bear that in mind when adding the overlay files inside the `root/` directory to your project.

- OBS ignores file permissions. Therefore `config.sh` and `images.sh` will **always** be executed through BASH (see also: *Section 7.6, "User-Defined Scripts"*).

11.20.3 Recommendations

11.20.3.1 Working with OBS

Although OBS is an online service, it is not necessary to test every change by uploading it. OBS will use the same process as `osc build` does, so if your image builds locally via `osc build` it should also build online on OBS.

11.20.3.2 Repository Configuration

When setting up the project, enable the `images` repository: the `images` repository's checkbox can be found at the bottom of the selection screen that appears when clicking `Add from a Distribution` in the `Repositories` tab. Or specify it manually in the project configuration (it can be accessed via `osc meta -e prj`):

```
<repository name="images">
  <arch>x86_64</arch>
</repository>
```

Furthermore, OBS requires additional repositories from which it obtains your dependent packages. These repositories can be provided in two ways:

1. Add the repositories to the project configuration on OBS and omit them from `config.xml`. Provide only the following repository inside the image description:

   ```
   <repository type="rpm-md">
     <source path="obsrepositories:/"/>
   </repository>
   ```

   This instructs OBS to inject the repositories from your project into your appliance. Additional repositories can be added by invoking `osc meta -e prj` and adding a line of the following form as a child of `<repository name="images">`:

   ```
   <path project="$OBS_PROJECT" repository="$REPOSITORY_NAME"/>
   ```

   The order in which you add repositories matters: if a package is present in multiple repositories, then it is taken from the **first** repository. The **last** repository is subject to path expansion: its repository paths are included as well.

Don't forget to add the repository from the `Virtualization:Appliances:Builder` project, providing the latest stable version of KIWI NG (which you are very likely using for your local builds).
The following example repository configuration adds the repositories from the `Virtualization:Appliances:Builder` project and those from the latest snapshot of openSUSE Tumbleweed:

```
<project name="Virtualization:Appliances:Images:openSUSE-Tumbleweed">
  <title>Tumbleweed JeOS images</title>
  <description>Host JeOS images for Tumbleweed</description>
  <repository name="images">
    <path project="Virtualization:Appliances:Builder" repository="Factory"/>
    <path project="openSUSE:Factory" repository="snapshot"/>
    <arch>x86_64</arch>
  </repository>
</project>
```

The above can be simplified further using the path expansion of the last repository to:

```
<project name="Virtualization:Appliances:Images:openSUSE-Tumbleweed">
  <title>Tumbleweed JeOS images</title>
  <description>Host JeOS images for Tumbleweed</description>
  <repository name="images">
    <path project="Virtualization:Appliances:Builder" repository="Factory"/>
    <arch>x86_64</arch>
  </repository>
</project>
```

Now `Virtualization:Appliances:Builder` is the last repository, which' repositories are included into the search path. As `openSUSE:Factory/snapshot` is among these, it can be omitted from the repository list.

2. Keep the repositories in your `config.xml` configuration file. If you have installed the latest stable KIWI NG as described in *Chapter 2, Installation* then you should add the following repository to your projects configuration (accessible via **`osc meta -e prjconf`**), so that OBS will pick the latest stable KIWI NG version too:

```
<repository name="images">
  <path project="Virtualization:Appliances:Builder" repository="$DISTRO"/>
  <arch>x86_64</arch>
</repository>
```

> Replace `$DISTRO` with the appropriate name for the distribution that you are currently building and optionally adjust the architecture.

We recommend to use the first method, as it integrates better into OBS. Note that your image description will then no longer build outside of OBS though. If building locally is required, use the second method.

Warning

Adding the repositories to project's configuration makes it impossible to build images for different distributions from the same project.

Since the repositories are added for every package in your project, all your image builds will share the same repositories, thereby resulting in conflicts for different distributions.

We recommend to create a separate project for each distribution. If that is impossible, you can keep all your repositories (including `Virtualization:Appliances:Builder`) in `config.xml`. That however usually requires a large number of workarounds via `Prefer:` settings in the project configuration and is thus **not** recommended.

11.20.3.3 Project Configuration

The Open Build Service will by default create the same output file as KIWI NG when run locally, but with a custom filename ending (that is unfortunately unpredictable). This has the consequence that the download URL of your image will change with every rebuild (and thus break automated scripts). OBS can create symbolic links with static names to the latest build by adding the following line to the project configuration:

```
Repotype: staticlinks
```

If build Vagrant images (see *Section 11.7, "Image Description for Vagrant"*) add the repository-type `vagrant`. OBS creates a `boxes/` subdirectory in your download repositories, which contains JSON files for Vagrant .

If you have added your repositories to `config.xml`, you probably see errors of the following type:

```
unresolvable: have choice for SOMEPACKAGE: SOMEPAKAGE_1 SOMEPACKAGE_2
```

Instead of starting from scratch and manually adding `Prefer:` statements to the project configuration, we recommend to copy the current project configuration of the testing project `Virtualization:Appliances:Images:Testing_$ARCH:$DISTRO` into your own project. It provides a good starting point and can be adapted to your OBS project.

11.21 Using SUSE Product ISO To Build

Abstract. This page provides information how to use the SUSE media ISO with KIWI NG

When building an image with KIWI NG, the image description usually points to a number of public/private package source repositories from which the new image root tree will be created. Alternatively the vendor provided product ISO image(s) can be used. The contents of the ISO (DVD) media also provides package source repositories but organized in a vendor specific structure. As a user it's important to know about this structure such that the KIWI NG image description can consume it.

To use a SUSE product media the following steps are required:

- Mount the ISO media from file or DVD drive:

```
$ sudo mount Product_ISO_file.iso|DVD_drive /media/suse
```

- Lookup all `Product` and `Module` directories:
  Below `/media/suse` there is a directory structure which provides package repositories in directories starting with `Product-XXX` and `Module-XXX`. It depends on the package list in the KIWI NG image description from which location a package or a dependency of the package is taken. Therefore it is best practice to browse through all the directories and create a `<repository>` definition for each of them in the KIWI NG image description like the following example shows:

  ```
  <repository alias="DVD-1-Product-SLES">
      <source path="file:///media/suse/Product-SLES"/>
  </repository>

  <repository alias="DVD-1-Module-Basesystem">
      <source path="file:///media/suse/Module-Basesystem"/>
  </repository>
  ```

Once all the individual product and module repos has been created in the KIWI NG image description, the build process can be started as usual.

Note

Because of the manual mount process the `/media/suse` location stays busy after KIWI NG has created the image. The cleanup of this resource is a responsibility of the user and not done by KIWI NG

11.22 Circumvent Debian Bootstrap

Abstract. This page provides information how to build Debian based images without an extra bootstrap process.

When building Debian based images KIWI NG uses `apt` in the bootstrap and the system phase to create the image root tree. However, `apt` does not support a native way to bootstrap an empty root tree. Therefore the bootstrap phase uses apt only to resolve the given bootstrap packages and to download these packages from the given repositories. The list of packages is then manually extracted into the new root tree which is not exactly the same as if `apt` would have installed them natively. For the purpose of creating an initial tree to begin with, this procedure is acceptable though.

If, for some reasons, this bootstrap procedure is not applicable, KIWI NG allows for an alternative process which is based on a prebuilt bootstrap-root archive provided as a package.

To make use of a `bootstrap_package`, the name of that package needs to be referenced in the KIWI NG description as follows:

```
<packages type="bootstrap" bootstrap_package="bootstrap-root">
    <package name="a"/>
    <package name="b"/>
</packages>
```

The boostrap process now changes in a way that the provided bootstrap_package `bootstrap-root` will be installed on the build host machine. Next KIWI NG searches for a tar archive file `/var/lib/bootstrap/bootstrap-root.ARCH.tar.xz`, where `ARCH` is the name of the host architecture e.g `x86_64`. If found the archive gets unpacked and serves as the bootstrap root tree to begin with. The optionally provided additional bootstrap packages, `a` and `b` in this example will be installed like system packages via `chroot` and `apt`. Usually no additional bootstrap packages are needed as they could all be handled as system packages.

11.22.1 How to Create a bootstrap_package

Changing the setup in KIWI NG to use a `bootstrap_package` rather then using KIWI NG's debian bootstrap method to do the job, comes with the task to create that package providing the bootstrap root tree. There are more than one way to do this. The following procedure is just one example and requires some background knowledge about the Open Build Service OBS (https://build.opensuse.org) and its KIWI NG integration.

1. Create an OBS project and repository setup that matches your image target
2. Create an image build package

```
osc mkpac bootstrap-root
```

3. Create the following `appliance.kiwi` file

```
<image schemaversion="7.4" name="bootstrap-root">
    <description type="system">
        <author>The Author</author>
        <contact>author@example.com</contact>
        <specification>prebuilt bootstrap rootfs for ...</specification>
    </description>

    <preferences>
        <version>1.0.1</version>
        <packagemanager>apt</packagemanager>
        <type image="tbz"/>
    </preferences>

    <repository type="rpm-md">
        <source path="obsrepositories:/"/>
    </repository>

    <packages type="image">
        <package name="gawk"/>
        <package name="apt-utils"/>
        <package name="debconf"/>
        <package name="mawk"/>
        <package name="libpam-runtime"/>
        <package name="util-linux"/>
        <package name="systemd"/>
        <package name="init"/>
        <package name="gnupg"/>
        <package name="iproute2"/>
        <package name="iptables"/>
        <package name="iputils-ping"/>
```

```
        <package name="ifupdown"/>
        <package name="isc-dhcp-client"/>
        <package name="netbase"/>
        <package name="dbus"/>
        <package name="xz-utils"/>
        <package name="usrmerge"/>
        <package name="language-pack-en"/>
    </packages>

    <packages type="bootstrap"/>
</image>
```

```
osc add appliance.kiwi
osc ci
```

4. Package the image build results into a debian package

   In step 3. the bootstrap root tarball was created but not yet packaged. A debian package is needed such that it can be referenced with the `bootstrap_package` attribute and the repository providing it. The simplest way to package the `bootstrap-root` tarball is to create another package in OBS and use the tarball file as its source.

12 Contributing

Note

Abstract

This document describes the development process of KIWI NG and how you can be part of it. This description applies to version 10.2.35.

12.1 Using KIWI NG in a Python Project

Note

Abstract

KIWI NG is provided as a Python module under the **kiwi** namespace. It is available for the Python 3 version. The following description applies for KIWI NG version 10.2.35.

KIWI NG can also function as a module for other Python projects. The following example demonstrates how to read an existing image description, add a new repository definition, and export the modified description on stdout.

```
import sys
import logging

from kiwi.xml_description import XMLDescription
from kiwi.xml_state import XMLState

description = XMLDescription('path/to/kiwi/XML/config.xml')

xml_data = description.load()

xml_state = XMLState(
    xml_data=xml_data, profiles=[], build_type='iso'
)

xml_state.add_repository(
    repo_source='http://repo',
    repo_type='rpm-md',
    repo_alias='myrepo',
```

```
    repo_prio=99
)

xml_data.export(
    outfile=sys.stdout, level=0
)
```

Each class in the example is responsible for a single tasks, so they can be reused in other user cases. Therefore it is possible to use KIWI NG beyond the main image building scope, for example to manage setup of loop devices, filesystems, partitions, etc.

This means KIWI NG offers a way to describe a system, but you can choose whether you want to use the KIWI NG description format or not. The following example shows how to use KIWI NG to create a simple filesystem image which contains your host `tmp` directory.

```
import logging

from kiwi.storage.loop_device import LoopDevice
from kiwi.filesystem import FileSystem

with LoopDevice(
    filename='my_tmp.ext4', filesize_mbytes=100
) as loop_provider:
    loop_provider.create()

    filesystem = FileSystem.new(
        'ext4', loop_provider, '/tmp/'
    )
    filesystem.create_on_device(
        label='TMP'
    )
    filesystem.sync_data()
```

12.2 Plugin Architecture

Each command provided by KIWI NG is written as a task plugin under the **kiwi.tasks** namespace. As a developer, you can extend KIWI NG with custom task plugins, following the conventions below.

12.2.1 Naming conventions

Task plugin file name

The file name of a task plugin must follow the pattern `<service>_<command>.py`. This allows you to invoke the task with **`kiwi-ng service command ...`**

Task plugin option handling

KIWI NG uses the docopt module to handle options. Each task plugin must use docopt to allow option handling.

Task plugin class

The implementation of the plugin must be a class that matches the naming convention `<Service><Command>Task`. The class must inherit from the `CliTask` base class. On the plugin startup, KIWI NG expects an implementation of the `process` method.

Task plugin entry point

Registration of the plugin must be done in `pyproject.toml` using the `tool.poetry.plugins` concept.

```
[tool.poetry]
name = "kiwi_plugin"

packages = [
    { include = "kiwi_plugin"},
]

[tool.poetry.plugins]
[tool.poetry.plugins."kiwi.tasks"]
service_command = "kiwi_plugin.tasks.service_command"
```

12.2.2 Example plugin

Note

The following example assumes an existing Python project which was set up using poetry and pyproject.toml.

1. Assuming the project namespace is **kiwi_relax_plugin**, create the task plugin directory `kiwi_relax_plugin/tasks`

2. Create the entry point in **`pyproject.toml`**.
   Assuming we want to create the service named **relax** that has the command **justdoit**, this is the required plugin definition in `pyproject.toml`:

```
[tool.poetry]
name = "kiwi_relax_plugin"

packages = [
    { include = "kiwi_relax_plugin"},
]

[tool.poetry.plugins]
[tool.poetry.plugins."kiwi.tasks"]
relax_justdoit = "kiwi_relax_plugin.tasks.relax_justdoit"
```

3. Create the plugin code in the file `kiwi_relax_plugin/tasks/relax_justdoit.py` with the following content:

```
"""
usage: kiwi-ng relax justdoit -h | --help
       kiwi-ng relax justdoit --now

commands:
    justdoit
        time to relax

options:
    --now
        right now. For more details about docopt
        see: http://docopt.org
"""
# These imports requires kiwi to be part of your environment
# It can be either installed from pip into a virtual development
# environment or from the distribution package manager
from kiwi.tasks.base import CliTask
from kiwi.help import Help

class RelaxJustdoitTask(CliTask):
    def process(self):
        self.manual = Help()
        if self.command_args.get('help') is True:
            # The following will invoke man to show the man page
            # for the requested command. Thus for the call to
            # succeed a manual page needs to be written and
            # installed by the plugin
            return self.manual.show('kiwi::relax::justdoit')
```

```
        print(
            'https://genius.com/Frankie-goes-to-hollywood-relax-lyrics'
        )
```

4. Test the plugin

```
$ poetry run kiwi-ng relax justdoit --now
```

12.3 Write Integration Tests for the Scripts

KIWI NG comes with a set of helper functions that can be used in `config.sh` (see also: *Section 7.6, "User-Defined Scripts"*). These functions utilize containers to run the individual tasks and verify the final result.

Ensure that you have either **`podman`** or **`docker`** installed and configured on your system. With Podman, the integration tests use **`podman`** in **rootless mode** by default. You can select **`docker`** instead by setting the environment variable `CONTAINER_RUNTIME` to `docker`. Then you can run the integration tests via the `test_scripts` Makefile target:

```
$ sudo make test_scripts
```

The tests are written using the pytest-container (https://github.com/dcermak/pytest_container) plugin. If applicable, use the utility functions and fixtures of the plugin. For example, the `auto_container` and `auto_container_per_test` fixtures in conjunction with testinfra (https://testinfra.readthedocs.io/).

12.3.1 Test setup

The script tests can be run inside different containers specified in `test/scripts/conftest.py`. This file contains the `CONTAINERS` list with all currently present images. These images are pulled and built as needed, and the `functions.sh` is copied into `/bin/`, so it is available in `PATH`.

To use any of these containers, you can either define the global variable `CONTAINER_IMAGES` in a test module and use the `auto_container` fixture, or parametrize (https://docs.pytest.org/en/stable/parametrize.html) the `container` fixture indirectly:

```
@pytest.mark.parametrize("container_per_test", (TUMBLEWEED, LEAP_15_3), indirect=True)
def test_RmWorks(container_per_test):
```

```
    # create the file /root/foobar
    container_per_test.connection.run_expect([0], "touch /root/foobar")
    assert container_per_test.connection.file("/root/foobar").exists

    # source the functions and execute our function under test
    container_per_test.connection.run_expect([0], ". /bin/functions.sh && Rm /root/
foobar")

    # verify the result
    assert not container_per_test.connection.file("/root/foobar").exists
```

The example above uses the `_per_test` variant of the `container` fixture. It ensures that the container is used only in a single test function. Use this variant for tests that mutate the system under test, because otherwise it may lead race conditions that are difficult to debug. For tests that only perform reads, you can omit the `_per_test` suffix, so that the container environment can shared with other tests. This improves execution speed, but comes at the expense of safety in case of mutation.

For further information, refer to pytest-container (https://github.com/dcermak/pytest_container).

12.4 Extending KIWI NG with Custom Operations

Note

Abstract

Extension plugins in KIWI NG offer a mechanism for adding information outside the standard KIWI NG schema.

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

The described schema extension can be used in an additional KIWI NG task plugin to provide a new subcommand for KIWI NG. At the present moment, there is no other plugin interface except for providing additional KIWI NG commands.

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

This description applies for version 10.2.35.

12.4.1 The <extension> Section

The main KIWI NG schema supports an extension section that allows you to specify any XML structure and attributes, as long as they are attached to a namespace. This means that any custom XML structure can be implemented similar to the the example below:

```
<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 be unique
- Multiple different toplevel namespaces are allowed, for example: my_plugin_a, my_plugin_b

12.4.2 RELAX NG Schema for the Extension

If an extension section is found, KIWI NG looks up its namespace and uses 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 save the schema as `/usr/share/xml/kiwi/my_plugin.rng`

For the example above, the RELAX NG Schema in the compressed format `my_plugin.rnc` looks as follows:

```
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
```

12.4.3 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 is as follows:

```
<?xml version="1.0"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
    <system
        systemId="http://www.my_plugin.com"
        uri="file:////usr/share/xml/kiwi/my_plugin.rng"/>
</catalog>
```

For resolving the catalog, KIWI NG uses the **`xmlcatalog`** command and the main XML catalog from the system `/etc/xml/catalog`.

Note

How the main catalog is informed about the existence of the KIWI NG extension catalog file depends on the distribution and its version. Refer to the distribution documentation for information on 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
```

12.4.4 Using the extension

In order to test the extension, insert the example extension into one of your image description's `config.xml` file.

The following example reads 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'))
```

12.5 The Basics

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

If you want to implement a bigger feature, consider opening an issue on GitHub first to discuss the changes. Or join the discussion in the `#kiwi` channel on Riot.im (https://about.riot.im).

12.6 Fork the upstream repository

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

12.7 Create a local clone of the forked repository

```
$ git clone https://github.com/YOUR-USERNAME/kiwi

$ git remote add upstream https://github.com/OSInside/kiwi.git
```

12.8 Install Required Operating System Packages

KIWI NG requires additional packages at runtime which are not provided by **pip**. Those will be pulled in by installing the following package:

- kiwi-systemdeps

The package is provided on the Open Build Service in the Virtualization:Appliances:Builder (https://download.opensuse.org/repositories/Virtualization:/Appliances:/Builder) project. For manual inspection of the packages that are pulled in from the above kiwi-systemdeps package, please refer to the `package/python-kiwi-spec-template` spec file from the checked out Git repository.

12.9 Create a Python Virtual Development Environment

The following commands initializes and activates a development environment for Python 3:

```
$ poetry install
```

The command above automatically creates the application script called **kiwi-ng**, which allows you to run KIWI NG from the Python sources inside the virtual environment using Poetry:

```
$ poetry run kiwi-ng --help
```

12.10 Running the Unit Tests

Before submitting your changes via a pull request, ensure that all tests pass and that the code has the required test coverage via the command:

```
$ make check
$ make test
```

12.11 Create a Branch for each Feature or Bugfix

Code changes should be done in an extra Git branch. This allows for creating GitHub pull requests in a clean way. See also: Collaborating with issues and pull requests (https://help.github.com/en/categories/collaborating-with-issues-and-pull-requests)

```
$ git checkout -b my-topic-branch
```

Make and commit your changes.

Note

You can make multiple commits which is generally useful to give your changes a clear structure and to allow us to better review your work.

Note

Your work is important and must be signed to ensure the integrity of the repository and the code. Thus we recommend to setup a signing key as documented in Signing Git Patches (#signing-git-patches).

```
$ git commit -S -a
```

Once everything is done, push your local branch to your forked repository and create a pull request into the upstream repository.

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

Thank you much for contributing to KIWI NG. Your time and work effort is very much appreciated!

12.12 Coding Style

KIWI NG follows the general PEP8 guidelines with the following exceptions:

- We do not use free functions at all. Even utility functions must be part of a class, but should be either prefixed with the `@classmethod` or `@staticmethod` decorators (whichever is more appropriate).
- Do not set module and class level variables, put these into the classes' `__init__` method.
- The names of constants are not written in all capital letters.

12.12.1 Documentation

KIWI NG uses Sphinx (https://www.sphinx-doc.org/en/master/) for the API, user documentation and manual pages

```
$ make docs
```

Document all your classes, methods, their parameters and their types using the standard reStructuredText (https://www.sphinx-doc.org/en/master/usage/restructuredtext/index.html) syntax as supported by Sphinx, an example class is documented as follows:

```
class Example:
    """
    **Example class**

    :param str param: A parameter
    :param bool : Source file name to compress
    :param list supported_zipper: List of supported compression tools
    :attr Optional[str] attr: A class attribute
    """
    def __init__(self, param, param_w_default=False):
        self.attr = param if param_w_default else None

    def method(self, param):
        """
        A method that takes a parameter.

        :param list param: a parameter
        :return: whether param is very long
        :rtype: bool
        """
        return len(param) > 50
```

Try to stick to the following guidelines when documenting source code:

- Classes should be documented directly in their main docstring and not in `__init__`.
- Document **every** function parameter and every public attribute including their types.
- Only public methods should be documented, private methods don't have to, unless they are complex and it is not easy to grasp what they do (which should be avoided anyway).

Please also document any user-facing changes that you implementing (e.g. adding a new build type) in the user documentation, which can be found in `doc/source`. General documentation should be put into the `working_with_kiwi/` subfolder, whereas documentation about more specialized topics would belong into the `building/` subfolder.

Adhere to a line limit of 75 characters when writing the user facing documentation .

12.13 Additional Information

The following sections provides further information about the repository integrity, version, package and documentation management.

12.13.1 Signing Git Patches

To ensure the integrity of the repository and the code base, patches sent for inclusion should be signed with a GPG key.

To prepare Git to sign commits, follow these instructions:

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

   ```
   $ gpg2 --expert --full-gen-key
   ```

2. Either choose a RSA key for signing (option `(4)`) or an ECC key for signing (option `(10)`). For a RSA key choose a key size of 4096 bits and for a ECC key choose Curve 25519 (option `(1)`). Enter a reasonable validity period (we recommend 2 to 5 years). Complete the key generation by entering your name and email address.

3. Add the key ID to your git configuration, by running the following **`git config`** commands:

   ```
   $ git config --local user.signingkey $YOUR_SIGN_KEY_ID
   ```

```
$ git config --local commit.gpgSign true
```

Omitting the flag `--local` will make these settings global for all repositories (they will be added to `~/.gitconfig`). You can find your signkey's ID via:

```
$ gpg2 --list-keys --keyid-format long $YOUR_EMAIL
pub   rsa4096/AABBCCDDEEFF0011 2019-04-26 [S] [expires: 2021-04-16]
AAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBB
uid                 [ultimate] YOU <$YOUR_EMAIL>
```

The key's ID in this case would be `AABBCCDDEEFF0011`. Note that your signkey will have only a `[S]` after the creation date, not a `[SC]` (then you are looking at your ordinary GPG key that can also encrypt).

12.13.2 Bumping the Version

The KIWI NG project follows the Semantic Versioning (https://semver.org) scheme. We use the **`bumpversion`** tool for consistent versioning.

Follow these instructions to bump the major, minor, or patch part of the KIWI NG version. Ensure that your repository is clean (i.e. no modified and unknown files exist) beforehand running **`bumpversion`**.

- For backwards-compatible bug fixes:

  ```
  $ bumpversion patch
  ```

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

  ```
  $ bumpversion minor
  ```

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

  ```
  $ bumpversion major
  ```

12.13.3 Creating a RPM Package

We provide a template for a RPM spec file in `package/python-kiwi-spec-template` alongside with a rpmlint configuration file and an automatically updated `python-kiwi.changes`.

To create the necessary files to build a RPM package via `rpmbuild`, run:

```
$ make build
```

The sources are collected in the `dist/` directory. These can be directly build it with **`rpmbuild`**, **`fedpkg`**, or submitted to the Open Build Service using **`osc`**.

13 Building Build Tests

KIWI NG provides a collection of integration test images for different architectures and distributions. The test descriptions covers a number of appliance features that can be build with KIWI NG. The test descriptions are stored in a directory structure of the following layout: `build-tests/ARCH/DISTRIBUTION/test-image-NAME`. To build integration test image(s), the script `build-tests.sh` exists.

The implementation of `build-tests.sh` calls kiwi's `boxbuild` command in container mode, which allows calling the script on actually any host system that allows to run container instances via `podman`.

Warning

Architectures

Cross architecture image building is possible and also supported via boxbuild, but the performance impact is big even in containers using qemu-binfmt and even bigger in full qemu arch emulation. This would slow down building the integration tests a lot and therefore `build-tests.sh` requires the host architecture to match with the image target architecture.

Prior calling `build-tests.sh` the following requirements must be met:

`Tools`

Install the packages providing the following tools:

- tree
- git
- xmllint
- podman
- pip

`Source Checkout`

Checkout the kiwi git repo which provides the test descriptions as well as the `build-tests.sh` script

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

`kiwi-boxed-plugin`

Fetch the kiwi-boxed-plugin from pip. It provides the boxbuild command used by build-tests.sh

```
$ pip install --upgrade kiwi-boxed-plugin
```

Warning

Make sure to be able to execute `kiwi-ng`. In case there was no KIWI NG installed on your host you will get notified by the kiwi-boxed-plugin installation to update your path to `export PATH:~/.local/bin/kiwi-ng:$PATH`. If in doubt about all this just install kiwi from pip too. `pip install --upgrade kiwi`

Building a specific integration test can be done as follows:

```
$ cd kiwi
$ ./build-tests.sh \
      --test-dir build-tests/x86/tumbleweed/ \
      --test-name test-image-disk
```

Building all integration tests for a specific arch and distribution can be done as follows:

```
$ cd kiwi
$ ./build-tests.sh \
      --test-dir build-tests/x86/tumbleweed/
```

Note

Building all integration tests can take some time and depends on the number of tests provided as well as on the build power of the host system. In general the tests can also run in parallel or distributed to multiple hosts

14 Python API

Note

This API documentation covers KIWI NG 10.2.35

14.1 kiwi Package

14.1.1 Submodules

14.1.2 kiwi.app Module

`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

14.1.3 kiwi.cli Module

`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

kiwi.cli.Cli.get_command

Extract selected command name

Returns

command name

Return type

str

kiwi.cli.Cli.get_command_args

Extract argument dict for selected command

Returns

Contains dictionary of command arguments

```
{
    '--command-option': 'value'
}
```

Return type

dict

kiwi.cli.Cli.get_global_args

Extract argument dict for global arguments

Returns

Contains dictionary of global arguments

```
{
    '--global-option': 'value'
}
```

Return type

dict

kiwi.cli.Cli.get_servicename

Extract service name from argument parse result

Returns

service name

Return type

str

kiwi.cli.Cli.load_command

Loads task class plugin according to service and command name

Returns

loaded task module

Return type

object

kiwi.cli.Cli.show_and_exit_on_help_request

Execute man to show the selected manual page

14.1.4 kiwi.command Module

`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

kiwi.command.Command.call

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` () – command and arguments
- `custom_env` () – 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

kiwi.command.Command.run ,

Execute a program and block the caller. The return value is a CommandT namedtuple 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. If raise_on_command_not_found is `False` and the command is not found, then `None` is returned.
Example:

```
result = Command.run(['ls', '-l'])
```

Parameters

- `command` () – command and arguments
- `custom_env` () – custom os.environ
- `raise_on_error` () – control error behaviour
- `stderr_to_stdout` () – redirects stderr to stdout

Returns

Contains call results in command type

```
CommandT(output='string', error='string', returncode=int)
```

Return type

`kiwi.command.CommandT`

`kiwi.command.CommandCallT`

Bases: `NamedTuple`

kiwi.command.CommandCallT.error

Alias for field number 2

kiwi.command.CommandCallT.error_available

Alias for field number 3

kiwi.command.CommandCallT.output

Alias for field number 0

kiwi.command.CommandCallT.output_available

Alias for field number 1

kiwi.command.CommandCallT.process

Alias for field number 4

`kiwi.command.CommandT`

Bases: `NamedTuple`

kiwi.command.CommandT.error

Alias for field number 1

kiwi.command.CommandT.output

Alias for field number 0

kiwi.command.CommandT.returncode

Alias for field number 2

14.1.5 `kiwi.command_process` Module

`kiwi.command_process.CommandIterator`

Bases: `object`

Implements an Iterator for Instances of Command

Parameters

`command` () – instance of subprocess

kiwi.command_process.CommandIterator.get_error_code

Provide return value from processed command

Returns

errorcode

Return type

int

kiwi.command_process.CommandIterator.get_error_output

Provide data which was sent to the stderr channel

Returns

stderr data

Return type

str

kiwi.command_process.CommandIterator.get_pid

Provide process ID of command while running

Returns

pid

Return type

int

kiwi.command_process.CommandIterator.kill

Send kill signal SIGTERM to command process

`kiwi.command_process.CommandProcess`

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` () – instance of subprocess
- `log_topic` () – topic string for logging

kiwi.command_process.CommandProcess.create_match_method

create a matcher function pointer which calls the given method as method(item_to_match, data) on dereference

Parameters

`method` () – function reference

Returns

function pointer

Return type

object

kiwi.command_process.CommandProcess.poll

Iterate over process, raise on error and log output

kiwi.command_process.CommandProcess.poll_and_watch

Iterate over process don't raise on error and log stdout and stderr

kiwi.command_process.CommandProcess.poll_show_progress

Iterate over process and show progress in percent raise on error and log output

Parameters

- `items_to_complete` () – all items
- `match_method` () – method matching item

kiwi.command_process.CommandProcess.returncode

`kiwi.command_process.PollT`

Bases: `NamedTuple`

kiwi.command_process.PollT.stderr_line

Alias for field number 1

kiwi.command_process.PollT.stdout_line

Alias for field number 0

14.1.6 `kiwi.defaults` Module

`kiwi.defaults.Defaults`

Bases: `object`

Implements default values

Provides static methods for default values and state information

kiwi.defaults.Defaults.get

Implements get method for profile elements

Parameters

`key` () – profile keyname

Returns

key value

Return type

str

kiwi.defaults.Defaults.get_apk_repo_config

Repository file for apk

Returns

file path name

Return type

str

kiwi.defaults.Defaults.get_archive_image_types

Provides list of supported archive image types

Returns

archive names

Return type

list

kiwi.defaults.Defaults.get_bls_loader_entries_dir

Provide default loader entries directory for BLS loaders

Returns

directory name

Return type

str

kiwi.defaults.Defaults.get_boot_image_description_path

Provides the path to find custom kiwi boot descriptions

Returns

directory path

Return type

str

kiwi.defaults.Defaults.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

kiwi.defaults.Defaults.get_buildservice_env_name

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

Returns

file basename

Return type

str

kiwi.defaults.Defaults.get_ca_target_distributions

kiwi.defaults.Defaults.get_ca_update_map

kiwi.defaults.Defaults.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

kiwi.defaults.Defaults.get_container_base_image_tag

Provides the tag used to identify base layers during the build of derived images.

Returns

tag

Return type

str

kiwi.defaults.Defaults.get_container_compression

Provides default container compression

Returns

True

Return type

bool

kiwi.defaults.Defaults.get_container_image_types

Provides list of supported container image types

Returns

container names

Return type

list

kiwi.defaults.Defaults.get_custom_rpm_bootstrap_macro_name

Returns the rpm bootstrap macro file name created in the custom rpm macros path

Returns

filename

Return type

str

kiwi.defaults.Defaults.get_custom_rpm_image_macro_name

Returns the rpm image macro file name created in the custom rpm macros path

Returns

filename

Return type

str

kiwi.defaults.Defaults.get_custom_rpm_macros_path

Returns the custom macros directory for the rpm database.

Returns

path name

Return type

str

kiwi.defaults.Defaults.get_default_boot_mbytes

Provides default boot partition size in mbytes

Returns

mbsize value

Return type

int

kiwi.defaults.Defaults.get_default_boot_timeout_seconds

Provides default boot timeout in seconds

Returns

seconds

Return type

int

kiwi.defaults.Defaults.get_default_bootloader

Return default bootloader name which is grub2

Returns

bootloader name

Return type

str

kiwi.defaults.Defaults.get_default_container_created_by

Provides the default 'created by' history entry for containers.

Returns

the specific kiwi version used for the build

Return type

str

kiwi.defaults.Defaults.get_default_container_name

Provides the default container name.

Returns

name

Return type

str

kiwi.defaults.Defaults.get_default_container_subcommand

Provides the default container subcommand.

Returns

command as a list of arguments

Return type

list

kiwi.defaults.Defaults.get_default_container_tag

Provides the default container tag.

Returns

tag

Return type

str

kiwi.defaults.Defaults.get_default_disk_start_sector

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

Returns

sector value

Return type

int

kiwi.defaults.Defaults.get_default_efi_boot_mbytes

Provides default EFI partition size in mbytes

Returns

mbsize value

Return type

int

kiwi.defaults.Defaults.get_default_efi_partition_table_type

Provides the default partition table type for efi firmwares.

Returns

partition table type name

Return type

str

kiwi.defaults.Defaults.get_default_firmware

Provides default firmware for specified architecture

Parameters

`arch` () – machine architecture name

Returns

firmware name

Return type

str

kiwi.defaults.Defaults.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

kiwi.defaults.Defaults.get_default_legacy_bios_mbytes

Provides default size of bios_grub partition in mbytes

Returns

mbsize value

Return type

int

kiwi.defaults.Defaults.get_default_live_iso_root_filesystem

Provides default live iso root filesystem type

Returns

filesystem name

Return type

str

kiwi.defaults.Defaults.get_default_live_iso_type

Provides default live iso union type

Returns

live iso type

Return type

str

kiwi.defaults.Defaults.get_default_package_manager

Returns the default package manager name if none is configured in the image description

Returns

package manager name

Return type

str

kiwi.defaults.Defaults.get_default_packager_tool

Provides the packager tool according to the package manager

Parameters

`package_manager` () – package manger name

Returns

packager tool binary name

Return type

str

kiwi.defaults.Defaults.get_default_prep_mbytes

Provides default size of prep partition in mbytes

Returns

mbsize value

Return type

int

kiwi.defaults.Defaults.get_default_uri_type

Provides default URI type

Absolute path specifications used in the context of an URI will apply the specified default mime type

Returns

URI mime type

Return type

str

kiwi.defaults.Defaults.get_default_video_mode

Uses auto mode for default video. See get_video_mode_map for details on the value depending which bootloader is used

Returns

auto

Return type

str

kiwi.defaults.Defaults.get_default_volume_group_name

Provides default LVM volume group name

Returns

name

Return type

str

kiwi.defaults.Defaults.get_discoverable_partition_ids

Provides arch specific partition UUIDs as defined by the UAPI group

Returns

partition UUIDs

Return type

dict

kiwi.defaults.Defaults.get_disk_format_types

Provides supported disk format types

Returns

disk types

Return type

list

kiwi.defaults.Defaults.get_disk_image_types

Provides supported disk image types

Returns

disk image type names

Return type

list

kiwi.defaults.Defaults.get_dracut_conf_name

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

Returns

file path name

Return type

str

kiwi.defaults.Defaults.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

kiwi.defaults.Defaults.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

kiwi.defaults.Defaults.get_efi_image_name

Provides architecture specific EFI boot binary name

Parameters

`arch` () – machine architecture name

Returns

name

Return type

str

kiwi.defaults.Defaults.get_efi_module_directory_name

Provides architecture specific EFI directory name which stores the EFI binaries for the desired architecture.

Parameters

`arch` () – machine architecture name

Returns

directory name

Return type

str

kiwi.defaults.Defaults.get_efi_vendor_directory

Provides EFI vendor directory if present

Looks up distribution specific EFI vendor directory

Parameters

`root_path` () – path to efi mountpoint

Returns

directory path or None

Return type

str

kiwi.defaults.Defaults.get_enclaves_image_types

Provides supported enclave(initrd-only) image types

Returns

enclave image type names

Return type

list

kiwi.defaults.Defaults.get_exclude_list_for_non_physical_devices

Provides the list of folders that are not associated with a physical device. KIWI returns the basename of the folders typically used as mountpoint for those devices.

Returns

list of file and directory names

Return type

list

kiwi.defaults.Defaults.get_exclude_list_for_removed_files_detection

Provides list of files/dirs to exclude from the removed files detection in a delta root build

kiwi.defaults.Defaults.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

kiwi.defaults.Defaults.get_exclude_list_from_custom_exclude_files

Gets the list of excluded items for the root filesystem from the optional metadata file image/exclude_files.yaml

Returns

list of paths and glob patterns

Parameters

`root_dir` () – image root directory

Return type

list

kiwi.defaults.Defaults.get_exclude_list_from_custom_exclude_files_for_efifatimage

Gets the list of excluded items for the ESP's EFI folder from the optional metadata file image/exclude_files_efifatimage.yaml

Excluded items must be relative to the ESP's /EFI directory.

Returns

list of paths and glob patterns

Parameters

`root_dir` () – EFI root directory

Return type

list

kiwi.defaults.Defaults.get_failsafe_kernel_options

Provides failsafe boot kernel options

Returns

list of kernel options

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

Return type

list

kiwi.defaults.Defaults.get_filesystem_image_types

Provides list of supported filesystem image types

Returns

filesystem names

Return type

list

kiwi.defaults.Defaults.get_firmware_types

Provides supported architecture specific firmware types

Returns

firmware types per architecture

Return type

dict

kiwi.defaults.Defaults.get_grub_basic_modules

Provides list of basic grub modules

Parameters

`multiboot` () – grub multiboot mode

Returns

list of module names

Return type

list

kiwi.defaults.Defaults.get_grub_boot_directory_name

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

kiwi.defaults.Defaults.get_grub_chrp_loader

Lookup CHRP boot loader (ppc)

Parameters

`boot_path` () – boot path

Returns

file base name

Return type

str

kiwi.defaults.Defaults.get_grub_custom_arguments

kiwi.defaults.Defaults.get_grub_efi_font_directory

Provides distribution specific EFI font directory used with grub.

Parameters

`root_path` () – image root path

Returns

file path or None

Return type

str

kiwi.defaults.Defaults.get_grub_efi_modules

Provides list of grub efi modules

Parameters

`multiboot` () – grub multiboot mode

Returns

list of module names

Return type

list

kiwi.defaults.Defaults.get_grub_ofw_modules

Provides list of grub ofw modules (ppc)

Returns

list of module names

Return type

list

kiwi.defaults.Defaults.get_grub_path

Provides grub path to given search file

Depending on the distribution grub could be installed below a grub2 or grub directory. grub could also reside in /usr/lib as well as in /usr/share. Therefore this information needs to be dynamically looked up

Parameters

- `root_path` () – root path to start the lookup from
- `filename` () – filename to search
- `raise_on_error` () – raise on not found, defaults to True

The method returns the path to the given grub search file. By default it raises a KiwiBootLoaderGrubDataError exception if the file could not be found in any of the search locations. If raise_on_error is set to False and no file could be found the function returns None

Returns

filepath

Return type

str

kiwi.defaults.Defaults.get_grub_platform_core_loader

Provides grub bios image

Searches distribution specific locations to find the core bios image below the given root path

Parameters

`root_path` () – image root path

Returns

file path or None

Return type

str

kiwi.defaults.Defaults.get_grub_platform_image_name

Provides platform specific core boot binary name

Returns

name

Return type

str

kiwi.defaults.Defaults.get_grub_platform_module_directory_name

Provides grub platform specific directory name which stores the grub module binaries

Returns

directory name

Return type

str

kiwi.defaults.Defaults.get_grub_platform_modules

Provides list of platform specific grub modules

Parameters

`multiboot` () – grub multiboot mode

Returns

list of module names

Return type

list

kiwi.defaults.Defaults.get_grub_s390_modules

Provides list of grub ofw modules (s390)

Returns

list of module names

Return type

list

kiwi.defaults.Defaults.get_imported_root_image

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` () – image root directory

Returns

file path name

Return type

str

kiwi.defaults.Defaults.get_install_volume_id

Provides default value for ISO volume ID for install media

Returns

name

Return type

str

kiwi.defaults.Defaults.get_iso_boot_path

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

Returns

relative path name

Return type

str

kiwi.defaults.Defaults.get_iso_grub_loader

Return name of eltorito grub image used as ISO loader

Returns

file base name

Return type

str

kiwi.defaults.Defaults.get_iso_grub_mbr

Return name of hybrid MBR image used as ISO boot record

Returns

file base name

Return type

str

kiwi.defaults.Defaults.get_iso_media_tag_tool

Provides default iso media tag tool

Returns

name

Return type

str

kiwi.defaults.Defaults.get_iso_tool_category

Provides default iso tool category

Returns

name

Return type

str

kiwi.defaults.Defaults.get_kis_image_types

Provides supported kis image types

Returns

kis image type names

Return type

list

kiwi.defaults.Defaults.get_live_dracut_modules_from_flag

Provides flag_name to dracut modules name map

Depending on the value of the flag attribute in the KIWI image description specific dracut modules need to be selected

Returns

dracut module names as list

Return type

list

kiwi.defaults.Defaults.get_live_image_types

Provides supported live image types

Returns

live image type names

Return type

list

kiwi.defaults.Defaults.get_live_iso_persistent_boot_options

Provides list of boot options passed to the dracut kiwi-live module to setup persistent writing

Returns

list of boot options

Return type

list

kiwi.defaults.Defaults.get_luks_key_length

Provides key length to use for random luks keys

kiwi.defaults.Defaults.get_lvm_overhead_mbytes

Provides empiric LVM overhead size in mbytes

Returns

mbsize value

Return type

int

kiwi.defaults.Defaults.get_min_partition_mbytes

Provides default minimum partition size in mbytes

Returns

mbsize value

Return type

int

kiwi.defaults.Defaults.get_min_volume_mbytes

Provides default minimum LVM volume size in mbytes per filesystem

Returns

mbsize value

Return type

int

kiwi.defaults.Defaults.get_mok_manager

Provides Mok Manager file path

Searches distribution specific locations to find the Mok Manager EFI binary

Parameters

`root_path` () – image root path

Returns

file path or None

Return type

str

kiwi.defaults.Defaults.get_obs_api_server_url

Provides the default API server url to access the public open buildservice API

Returns

url path

Return type

str

kiwi.defaults.Defaults.get_obs_download_server_url

Provides the default download server url hosting the public open buildservice repositories

Returns

url path

Return type

str

kiwi.defaults.Defaults.get_oci_archive_tool

Provides the default OCI archive tool name.

Returns

name

Return type

str

kiwi.defaults.Defaults.get_part_mapper_tool

Provides the default partition mapper tool name.

Returns

name

Return type

str

kiwi.defaults.Defaults.get_platform_name

Provides the machine architecture name as used by KIWI

This is the architecture name as it is returned by ‘uname -m’ with one exception for the 32bit x86 architecture which is handled as ‘ix86’ in general

Returns

architecture name

Return type

str

kiwi.defaults.Defaults.get_preparer

Provides ISO preparer name

Returns

name

Return type

str

kiwi.defaults.Defaults.get_profile_file

Return name of profile file for given root directory

kiwi.defaults.Defaults.get_publisher

Provides ISO publisher name

Returns

name

Return type

str

kiwi.defaults.Defaults.get_recovery_spare_mbytes

Provides spare size of recovery partition in mbytes

Returns

mbsize value

Return type

int

kiwi.defaults.Defaults.get_removed_files_name

Provides base file name to store removed files in a delta root build

kiwi.defaults.Defaults.get_runtime_checker_metadata

kiwi.defaults.Defaults.get_schema_file

Provides file path to kiwi RelaxNG schema

Returns

file path

Return type

str

kiwi.defaults.Defaults.get_schematron_module_name

Provides module name for XML SchemaTron validations

Returns

python module name

Return type

str

kiwi.defaults.Defaults.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

kiwi.defaults.Defaults.get_shim_loader

Provides shim loader file path

Searches distribution specific locations to find shim.efi below the given root path

Parameters

`root_path` () – image root path

Returns

list of shim_loader_type

Return type

list

kiwi.defaults.Defaults.get_shim_vendor_directory

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` () – image root path

Returns

directory path or None

Return type

str

kiwi.defaults.Defaults.get_signed_grub_loader

Provides shim signed grub loader file path

Searches distribution specific locations to find a grub EFI binary within the given root path

Parameters

`root_path` () – image root path

Returns

list of grub_loader_type

Return type

list

kiwi.defaults.Defaults.get_snapper_config_template_file

Provides the default configuration template file for snapper. The location in etc/ are preferred over files in usr/

Returns

file path

Return type

str

kiwi.defaults.Defaults.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

kiwi.defaults.Defaults.get_swapsize_mbytes

Provides swapsize in MB

kiwi.defaults.Defaults.get_sync_options

Provides list of default data sync options

Returns

list of rsync options

Return type

list

kiwi.defaults.Defaults.get_system_files_name

Provides base file name to store system files in a container build

kiwi.defaults.Defaults.get_temp_location

Provides the base temp directory location

This is the directory used to store any temporary files and directories created by kiwi during runtime

Returns

directory path

Return type

str

kiwi.defaults.Defaults.get_unsigned_grub_loader

Provides unsigned grub efi loader file path

Searches distribution specific locations to find a distro grub EFI binary within the given root path

Parameters

`root_path` () – image root path

Returns

list of grub_loader_type

Return type

list

kiwi.defaults.Defaults.get_vagrant_config_virtualbox_guest_additions

Provides the default value for `vagrantconfig.virtualbox_guest_additions_present`

Returns

whether guest additions are expected to be present in the vagrant box

Return type

bool

kiwi.defaults.Defaults.get_vendor_grubenv

kiwi.defaults.Defaults.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')}
```

Return type

dict

kiwi.defaults.Defaults.get_volume_id

Provides default value for ISO volume ID

Returns

name

Return type

str

kiwi.defaults.Defaults.get_xsl_stylesheet_file

Provides the file path to the KIWI XSLT style sheets

Returns

file path

Return type

str

kiwi.defaults.Defaults.get_xz_compression_options

Provides compression options for the xz compressor

Returns

Contains list of options

```
['--option=value']
```

Return type

list

kiwi.defaults.Defaults.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

kiwi.defaults.Defaults.is_ppc64_arch

Checks if machine architecture is ppc64 based

Any arch that matches little endian or big endian ppc64 architecture causes the method to return True. Anything else will cause the method to return False

Return type

bool

kiwi.defaults.Defaults.is_x86_arch

Checks if machine architecture is x86 based

Any arch that matches 32bit and 64bit x86 architecture causes the method to return True. Anything else will cause the method to return False

Return type

bool

kiwi.defaults.Defaults.project_file

Provides the python module base directory search path

The method uses the importlib.resources.path method to identify files and directories from the application

Parameters

`filename` () – relative project file

Returns

absolute file path name

Return type

str

kiwi.defaults.Defaults.set_custom_runtime_config_file

Sets the runtime config file once

Parameters

`filename` () – a file path name

kiwi.defaults.Defaults.set_platform_name

Sets the platform architecture once

Parameters

`name` () – an architecture name

kiwi.defaults.Defaults.set_runtime_checker_metadata

Sets the runtime checker metadata filename

Parameters

`filename` () – a file path name

kiwi.defaults.Defaults.set_shared_cache_location

Sets the shared cache location once

Parameters

`location` () – a location path

kiwi.defaults.Defaults.set_temp_location

Sets the temp directory location once

Parameters

`location` () – a location path

kiwi.defaults.Defaults.to_profile

Implements method to add list of profile keys and their values to the specified instance of a Profile class

Parameters

`profile` () – Profile instance

`kiwi.defaults.grub_loader_type`

Bases: `tuple`

kiwi.defaults.grub_loader_type.binaryname

Alias for field number 1

kiwi.defaults.grub_loader_type.filename

Alias for field number 0

kiwi.defaults.grub_loader_type.targetname

Alias for field number 2

`kiwi.defaults.shim_loader_type`

Bases: `tuple`

kiwi.defaults.shim_loader_type.binaryname

Alias for field number 1

kiwi.defaults.shim_loader_type.filename

Alias for field number 0

`kiwi.defaults.unit_type`

Bases: `tuple`

kiwi.defaults.unit_type.byte

Alias for field number 0

kiwi.defaults.unit_type.gb

Alias for field number 3

kiwi.defaults.unit_type.kb

Alias for field number 1

kiwi.defaults.unit_type.mb

Alias for field number 2

14.1.7 `kiwi.exceptions` Module

`kiwi.exceptions.KiwiAnyMarkupPluginError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the python anymarkup module failed to load.

`kiwi.exceptions.KiwiArchiveSetupError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if an unsupported image archive type is used.

`kiwi.exceptions.KiwiArchiveTarError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if impossible to determine which tar command version is installed on the underlying system

`kiwi.exceptions.KiwiBootImageSetupError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if an unsupported initrd system type is used.

`kiwi.exceptions.KiwiBootLoaderConfigSetupError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if a configuration for an unsupported bootloader is requested.

`kiwi.exceptions.KiwiBootLoaderDiskPasswordError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the disk password could not be set

`kiwi.exceptions.KiwiBootLoaderGrubDataError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if no grub installation was found.

`kiwi.exceptions.KiwiBootLoaderGrubFontError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if no grub unicode font was found.

`kiwi.exceptions.KiwiBootLoaderGrubInstallError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if grub install to master boot record has failed.

`kiwi.exceptions.KiwiBootLoaderGrubModulesError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the synchronisation of modules from the grub installation to the boot space has failed.

`kiwi.exceptions.KiwiBootLoaderGrubPlatformError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

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

`kiwi.exceptions.KiwiBootLoaderGrubSecureBootError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the Microsoft signed shim loader or grub2 loader could not be found in the image root system

kiwi.exceptions.KiwiBootLoaderInstallSetupError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if an installation for an unsupported bootloader is requested.

kiwi.exceptions.KiwiBootLoaderTargetError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the target to read the bootloader path from is not a disk or an iso image.

kiwi.exceptions.KiwiBootLoaderZiplInstallError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the installation of zipl has failed.

kiwi.exceptions.KiwiBootLoaderZiplPlatformError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if a configuration for an unsupported zipl architecture is requested.

kiwi.exceptions.KiwiBootLoaderZiplSetupError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the data set to configure the zipl bootloader is incomplete.

kiwi.exceptions.KiwiBootStrapPhaseFailed

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the bootstrap phase of the system prepare command has failed.

kiwi.exceptions.KiwiBuildahError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised on inconsistent buildah class calls

kiwi.exceptions.KiwiBundleError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the system bundle command has failed.

kiwi.exceptions.KiwiCATargetDistributionError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if no CA target distribution can be found but the request to import custom CA certificates was issued

kiwi.exceptions.KiwiCommandCapabilitiesError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception is raised when some the CommandCapabilities methods fails, usually meaning there is some issue trying to parse some command output.

`kiwi.exceptions.KiwiCommandError`

Bases: `KiwiError` (#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.

`kiwi.exceptions.KiwiCommandNotFound`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if any executable command cannot be found in the evironment PATH variable.

`kiwi.exceptions.KiwiCommandNotLoaded`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if a kiwi command task module could not be loaded.

`kiwi.exceptions.KiwiCompressionFormatUnknown`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the compression format of the data could not be detected.

`kiwi.exceptions.KiwiConfigFileFormatNotSupported`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if kiwi description file format is not supported.

`kiwi.exceptions.KiwiConfigFileNotFound`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if no kiwi XML description was found.

`kiwi.exceptions.KiwiContainerBuilderError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception is raised when something fails during a container image build procedure.

`kiwi.exceptions.KiwiContainerImageSetupError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if an attempt to create a container instance for an unsupported container type is performed.

`kiwi.exceptions.KiwiContainerSetupError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if an error in the creation of the container archive happened.

`kiwi.exceptions.KiwiCredentialsError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if required credentials information is missing

`kiwi.exceptions.KiwiCustomPartitionConflictError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the entry in a custom partition setup conflicts with an existing partition table layout setting

`kiwi.exceptions.KiwiDataStructureError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the XML description failed to parse the data structure.

`kiwi.exceptions.KiwiDebianBootstrapError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the bootstrap installation for Debian based systems has failed

`kiwi.exceptions.KiwiDecodingError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception is raised on decoding literals failure

`kiwi.exceptions.KiwiDescriptionInvalid`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the XML description failed to validate the XML schema.

`kiwi.exceptions.KiwiDeviceProviderError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if a storage provide is asked for its managed device but no such device exists.

`kiwi.exceptions.KiwiDiskBootImageError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if a kiwi boot image does not provide the requested data, e.g kernel, or hypervisor files.

`kiwi.exceptions.KiwiDiskFormatSetupError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if an attempt was made to create a disk format instance of an unsupported disk format.

`kiwi.exceptions.KiwiDiskGeometryError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the disk geometry (partition table) could not be read or evaluated against their expected geometry and capabilities.

`kiwi.exceptions.KiwiDistributionNameError`

Bases: `KiwiError` (#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.

`kiwi.exceptions.KiwiEnclaveBootImageError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if no kernel image was found while building an enclave image.

`kiwi.exceptions.KiwiEnclaveFormatError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if no enclave_format attribute specified for the selected build type

`kiwi.exceptions.KiwiEnvImportError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if extending os.environ with another env file caused an issue

`kiwi.exceptions.KiwiError`

Bases: `Exception`

Base class to handle all known exceptions

Specific exceptions are implemented as sub classes of KiwiError

Attributes

Parameters

`message` () – Exception message text

`kiwi.exceptions.KiwiExtensionError`

Bases: `KiwiError` (#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

kiwi.exceptions.KiwiFileAccessError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if accessing a file or its metadata failed

kiwi.exceptions.KiwiFileNotFound

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the requested file could not be found.

kiwi.exceptions.KiwiFileSystemSetupError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

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

kiwi.exceptions.KiwiFileSystemSyncError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the data sync from the system into the loop mounted filesystem image failed.

kiwi.exceptions.KiwiFormatSetupError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the requested disk format could not be created.

kiwi.exceptions.KiwiHelpNoCommandGiven

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the request for the help page is executed without a command to show the help for.

kiwi.exceptions.KiwiImageResizeError

Bases: KiwiError (#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.

kiwi.exceptions.KiwiImportDescriptionError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the XML description data and scripts could not be imported into the root of the image.

kiwi.exceptions.KiwiIncludFileNotFoundError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the file reference in an <include> statement could not be found

kiwi.exceptions.KiwiInstallBootImageError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the required files to boot an installation image could not be found, e.g kernel or hypervisor.

kiwi.exceptions.KiwiInstallMediaError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if a request for an installation media is made but the system image type is not an oem type.

kiwi.exceptions.KiwiInstallPhaseFailed

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the install phase of a system prepare command has failed.

kiwi.exceptions.KiwiIsoMetaDataError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if an inconsistency in the ISO header was found such like invalid eltorito specification or a broken path table.

kiwi.exceptions.KiwiIsoToolError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if an iso helper tool such as isoinfo could not be found on the build system.

kiwi.exceptions.KiwiKernelLookupError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the search for the kernel image file failed

kiwi.exceptions.KiwiKisBootImageError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if a required boot file e.g the kernel could not be found in the process of building a kis image.

kiwi.exceptions.KiwiLiveBootImageError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if an attempt was made to use an unsupported live iso type.

kiwi.exceptions.KiwiLoadCommandUndefined

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if no command is specified for a given service on the commandline.

`kiwi.exceptions.KiwiLogFileSetupFailed`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the log file could not be created.

`kiwi.exceptions.KiwiLogSocketSetupFailed`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the Unix Domain log socket could not be created.

`kiwi.exceptions.KiwiLoopSetupError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if not enough user data to create a loop device is specified.

`kiwi.exceptions.KiwiLuksSetupError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if not enough user data is provided to setup the luks encryption on the given device.

`kiwi.exceptions.KiwiMappedDeviceError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the device to become mapped does not exist.

`kiwi.exceptions.KiwiMarkupConversionError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the markup format conversion is not possible.

`kiwi.exceptions.KiwiMountKernelFileSystemsError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if a kernel filesystem such as proc or sys could not be mounted.

`kiwi.exceptions.KiwiMountSharedDirectoryError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the host <-> image shared directory could not be mounted.

`kiwi.exceptions.KiwiNotImplementedError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if a functionality is not yet implemented.

`kiwi.exceptions.KiwiOCIArchiveToolError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the requested OCI archive tool is not supported

kiwi.exceptions.KiwiOSReleaseImportError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if reading etc/os-release caused an issue

kiwi.exceptions.KiwiOffsetError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the offset for a seek operation does not match the expected data to write

kiwi.exceptions.KiwiPackageManagerSetupError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if an attempt was made to create a package manager instance for an unsupported package manager.

kiwi.exceptions.KiwiPackagesDeletePhaseFailed

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the packages deletion phase in system prepare fails.

kiwi.exceptions.KiwiPartitionTooSmallError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the specified partition size is smaller than the required bytes to store the data

kiwi.exceptions.KiwiPartitionerGptFlagError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if an attempt was made to set an unknown partition flag for an entry in the GPT table.

kiwi.exceptions.KiwiPartitionerMsDosFlagError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if an attempt was made to set an unknown partition flag for an entry in the MSDOS table.

kiwi.exceptions.KiwiPartitionerSetupError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if an attempt was made to create an instance of a partitioner for an unsupporte partitioner.

kiwi.exceptions.KiwiPrivilegesError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if root privileges are required but not granted.

`kiwi.exceptions.KiwiProfileNotFound`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if a specified profile does not exist in the XML configuration.

`kiwi.exceptions.KiwiRaidSetupError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if invalid or not enough user data is provided to create a raid array on the specified storage device.

`kiwi.exceptions.KiwiRepositorySetupError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if an attempt was made to create an instance of a repository for an unsupported package manager.

`kiwi.exceptions.KiwiRequestError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if a package request could not be processed by the corresponding package manager instance.

`kiwi.exceptions.KiwiRequestedTypeError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if an attempt was made to build an image for an unsupported image type.

`kiwi.exceptions.KiwiResizeRawDiskError`

Bases: `KiwiError` (#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.

`kiwi.exceptions.KiwiResultError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the image build result pickle information could not be created or loaded.

`kiwi.exceptions.KiwiRootDirExists`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the specified image root directory already exists and should not be re-used.

`kiwi.exceptions.KiwiRootImportError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception is raised when something fails during the root import procedure.

`kiwi.exceptions.KiwiRootInitCreationError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the initialization of a new image root directory has failed.

`kiwi.exceptions.KiwiRpmDirNotRemoteError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the provided rpm-dir repository is not local

`kiwi.exceptions.KiwiRuntimeConfigFileError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the provided custom runtime config file could not be found

`kiwi.exceptions.KiwiRuntimeConfigFormatError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the expected format in the yaml KIWI runtime config file does not match

`kiwi.exceptions.KiwiRuntimeError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if a runtime check has failed.

`kiwi.exceptions.KiwiSatSolverJobError`

Bases: `KiwiError` (#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

`kiwi.exceptions.KiwiSatSolverJobProblems`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the sat solver operations returned with solver problems e.g package conflicts

`kiwi.exceptions.KiwiSatSolverPluginError`

Bases: `KiwiError` (#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

`kiwi.exceptions.KiwiSchemaImportError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the schema file could not be read by lxml.RelaxNG.

`kiwi.exceptions.KiwiScriptFailed`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if a user script returned with an exit code != 0.

`kiwi.exceptions.KiwiSetupIntermediateConfigError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the setup of the temporary image system configuration for the duration of the build process has failed.

`kiwi.exceptions.KiwiShellVariableValueError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if a given python value cannot be converted into a string representation for use in shell scripts

`kiwi.exceptions.KiwiSizeError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception is raised when the convertion from a given size in string format to a number.

`kiwi.exceptions.KiwiSolverRepositorySetupError`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the repository type is not supported for the creation of a SAT solvable

`kiwi.exceptions.KiwiSystemDeletePackagesFailed`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the deletion of a package has failed in the corresponding package manager instance.

`kiwi.exceptions.KiwiSystemInstallPackagesFailed`

Bases: `KiwiError` (#kiwi.exceptions.KiwiError)

Exception raised if the installation of a package has failed in the corresponding package manager instance.

kiwi.exceptions.KiwiSystemUpdateFailed

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the package upgrade has failed in the corresponding package manager instance.

kiwi.exceptions.KiwiTargetDirectoryNotFound

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the specified target directory to store the image results was not found.

kiwi.exceptions.KiwiTemplateError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the substitution of variables in a configuration file template has failed.

kiwi.exceptions.KiwiTypeNotFound

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if no build type was found in the XML description.

kiwi.exceptions.KiwiUmountBusyError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the attempt to umount a resource has failed

kiwi.exceptions.KiwiUnknownServiceName

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if an unknown service name was provided on the commandline.

kiwi.exceptions.KiwiUriOpenError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the urllib urlopen request has failed

kiwi.exceptions.KiwiUriStyleUnknown

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if an unsupported URI style was used in the source definition of a repository.

kiwi.exceptions.KiwiUriTypeUnknown

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the protocol type of an URI is unknown in the source definition of a repository.

kiwi.exceptions.KiwiValidationError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the XML validation against the schema has failed.

kiwi.exceptions.KiwiVhdTagError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the GUID tag is not provided in the expected format.

kiwi.exceptions.KiwiVolumeGroupConflict

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the requested LVM volume group already is in use on the build system.

kiwi.exceptions.KiwiVolumeManagerSetupError

Bases: KiwiError (#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.

kiwi.exceptions.KiwiVolumeRootIDError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the root volume can not be found. This concept currently exists only for the btrfs subvolume system.

kiwi.exceptions.KiwiVolumeTooSmallError

Bases: KiwiError (#kiwi.exceptions.KiwiError)

Exception raised if the specified volume size is smaller than the required bytes to store the data

14.1.8 kiwi.firmware Module

kiwi.firmware.FirmWare

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`

kiwi.firmware.FirmWare.bios_mode

Check if BIOS mode is requested

Returns

True or False

Return type

bool

kiwi.firmware.FirmWare.ec2_mode

Check if EC2 mode is requested

Returns

True or False

Return type

bool

kiwi.firmware.FirmWare.efi_mode

Check if EFI mode is requested

Returns

The requested EFI mode or None if no EFI mode requested

Return type

str

kiwi.firmware.FirmWare.get_efi_partition_size

Size of EFI partition. Returns 0 if no such partition is needed

Returns

mbsize value

Return type

int

kiwi.firmware.FirmWare.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

kiwi.firmware.FirmWare.get_partition_table_type

Provides partition table type according to architecture and firmware

Returns

partition table name

Return type

str

kiwi.firmware.FirmWare.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

kiwi.firmware.FirmWare.legacy_bios_mode

Check if the legacy boot from BIOS systems should be activated

Returns

True or False

Return type

bool

kiwi.firmware.FirmWare.ofw_mode

Check if OFW mode is requested

Returns

True or False

Return type

bool

kiwi.firmware.FirmWare.opal_mode

Check if Opal mode is requested

Returns

True or False

Return type

bool

14.1.9 kiwi.help Module

`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.

kiwi.help.Help.show

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` () – man page name

14.1.10 kiwi.kiwi Module

`kiwi.kiwi.extras`

Overwritten method from docopt

Shows our own usage message for -h|–help

Parameters

- `help` () – indicate to show help
- `version` () – version string
- `options` ()

list of option tuples

```
[option(name='name', value='value')]
```

Parameters

`doc` () – 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`

Instead of the docopt way to show the usage information we provide a kiwi specific usage information. The usage data now always consists out of:

1. the generic call kiwi-ng [global options] service <command> [<args>]
2. the command specific usage defined by the docopt string short form by default, long form with -h | –help
3. the global options

Parameters

`command_usage` () – usage data

14.1.11 `kiwi.logger` Module

`kiwi.logger.Logger`

Bases: `Logger`

Extended logging facility based on Python logging

Parameters

`name` () – name of the logger

kiwi.logger.Logger.getLogFlags

Return logging flags

Returns

Dictionary with flags and their activation status

Return type

dict

kiwi.logger.Logger.getLogLevel

Return currently used log level

Returns

log level number

Return type

int

kiwi.logger.Logger.get_logfile

Return file path name of logfile

Returns

file path

Return type

str

kiwi.logger.Logger.progress

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` () – current item
- `total` () – total number of items
- `prefix` () – prefix name
- `bar_length` () – length of progress bar

kiwi.logger.Logger.setLogFlag

Set logging flag for further properties of the logging facility Available flags are:

- run-scripts-in-screen

Parameters

`flag` () – name

kiwi.logger.Logger.setLogLevel

Set custom log level for all console handlers

Parameters

- `level` () – log level number
- `except_for` () – set log level to all handlers except for the given list
- `only_for` () – set log level to the given handlers only

if both except_for and only_for handlers are specified, the except_for list will be ignored

kiwi.logger.Logger.set_color_format

Set color format for all console handlers

kiwi.logger.Logger.set_log_socket

Set log socket handler

Parameters

`filename` () – UDS socket file path. Note if there is no server listening on the socket the log handler setup will fail

kiwi.logger.Logger.set_logfile

Set logfile handler

Parameters

`filename` () – logfile file path

14.1.12 `kiwi.logger_color_formatter` Module

`kiwi.logger_color_formatter.ColorFormatter`

Bases: `Formatter`

Extended standard logging Formatter

Extended format supporting text with color metadata

Example:

```
ColorFormatter(message_format, '%H:%M:%S')
```

kiwi.logger_color_formatter.ColorFormatter.format

Creates a logging Formatter with support for color messages

Parameters

`record` () – logging message record

Returns

result from format_message

Return type

str

`kiwi.logger_color_formatter.ColorMessage`

Bases: `object`

Implements color messages for Python logging facility

Has to implement the format_message method to serve as message formatter

kiwi.logger_color_formatter.ColorMessage.format_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` () – color level name
- `message` () – text

Returns

color message with escape sequences

Return type

str

14.1.13 `kiwi.logger_filter` Module

`kiwi.logger_filter.DebugFilter`

Bases: `Filter`

Extended standard debug logging Filter

kiwi.logger_filter.DebugFilter.filter

Only messages with record level DEBUG can pass for messages with another level an extra handler is used

Parameters

`record` () – logging message record

Returns

True|False

Return type

bool

`kiwi.logger_filter.ErrorFilter`

Bases: `Filter`

Extended standard error logging Filter

kiwi.logger_filter.ErrorFilter.filter

Only messages with record level DEBUG can pass for messages with another level an extra handler is used

Parameters

`record` () – logging message record

Returns

True|False

Return type

bool

`kiwi.logger_filter.InfoFilter`

Bases: `Filter`

Extended standard logging Filter

kiwi.logger_filter.InfoFilter.filter

Only messages with record level INFO can pass for messages with another level an extra handler is used

Parameters

`record` () – logging message record

Returns

True|False

Return type

bool

`kiwi.logger_filter.LoggerSchedulerFilter`

Bases: `Filter`

Extended standard logging Filter

kiwi.logger_filter.LoggerSchedulerFilter.filter

Messages from apscheduler scheduler instances are filtered out They conflict with console progress information

Parameters

`record` () – logging message record

Returns

True|False

Return type

bool

`kiwi.logger_filter.WarningFilter`

Bases: `Filter`

Extended standard warning logging Filter

kiwi.logger_filter.WarningFilter.filter

Only messages with record level WARNING can pass for messages with another level an extra handler is used

Parameters

`record` () – logging message record

Returns

True|False

Return type

bool

14.1.14 `kiwi.mount_manager` Module

`kiwi.mount_manager.MountManager`

Bases: `object`

Implements methods for mounting, umounting and mount checking

The caller is responsible for unmounting the device if the MountManager is used as is. The class also supports to be used as a context manager, where the device is unmounted once the context manager's with block is left

- **param string device**
  device node name
- **param string mountpoint**
  mountpoint directory name
- **param dict attributes**
  optional attributes to store

kiwi.mount_manager.MountManager.bind_mount

Bind mount the device to the mountpoint

kiwi.mount_manager.MountManager.get_attributes

Return attributes dict for this mount manager

kiwi.mount_manager.MountManager.is_mounted

Check if mounted

Returns

True or False

Return type

bool

kiwi.mount_manager.MountManager.mount

Standard mount the device to the mountpoint

Parameters

`options` () – mount options

kiwi.mount_manager.MountManager.overlay_mount

kiwi.mount_manager.MountManager.tmpfs_mount

tmpfs mount the device to the mountpoint

kiwi.mount_manager.MountManager.umount

Umount by the mountpoint directory

Wait up to 10sec trying to umount. If the resource stays busy the call will raise an exception unless raise_on_busy is set to False. In case the umount failed and raise_on_busy is set to False, the method returns False to indicate the error condition.

Returns

True or False

Return type

bool

kiwi.mount_manager.MountManager.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

14.1.15 kiwi.path Module

`kiwi.path.Path`

Bases: `object`

Directory path helpers

kiwi.path.Path.access

Check whether path can be accessed with the given mode.

Parameters

- `path` () – The path that should be checked for access.
- `mode` () – Which access mode should be checked. This value must be a bitwise or of one or more of the following constants: `os.F_OK` (note that this one is zero), `os.X_OK`, `os.R_OK` and `os.W_OK`
- `kwargs` – further keyword arguments are forwarded to `os.access()`

Returns

Boolean value whether this access mode is allowed

Return type

bool

Raises

- `ValueError` – if the supplied mode is invalid
- `kiwi.exceptions.KiwiFileNotFound` (#kiwi.exceptions.KiwiFileNotFound) – if the path does not exist or is not accessible by the current user

kiwi.path.Path.create

Create path and all sub directories to target

Parameters

`path` () – path name

kiwi.path.Path.first_exists

Lookup first path that exists in the given path hierarchy

kiwi.path.Path.move_to_root

Change the given path elements to a new root directory

Parameters

- `root` () – the root path to trim
- `elements` () – list of path names

Returns

changed elements

Return type

list

kiwi.path.Path.rebase_to_root

Include the root prefix for the given paths elements

Parameters

- `root` () – the new root path
- `elements` () – list of path names

Returns

changed elements

Return type

list

kiwi.path.Path.remove_hierarchy

Recursively remove an empty path and its sub directories starting at a given root directory. Ignore non empty or protected paths and leave them untouched

Parameters

- `root` () – start at directory
- `path` () – path name below root

kiwi.path.Path.sort_by_hierarchy

Sort given list of path names by their hierachy in the tree
Example:

```
result = Path.sort_by_hierarchy(['/var/lib', '/var'])
```

Parameters

`path_list` () – list of path names

Returns

hierachy sorted path_list

Return type

list

kiwi.path.Path.which

Lookup file name in PATH

Parameters

- `filename` () – file base name
- `alternative_lookup_paths` () – list of additional lookup paths
- `custom_env` () – a custom os.environ used to obtain `$PATH`
- `access_mode` () – one of the os access modes or a combination of them (os.R_OK, os.W_OK and os.X_OK). If the provided access mode does not match the file is considered not existing
- `root_dir` () – the root path to look at

Returns

absolute path to file or None

Return type

str

kiwi.path.Path.wipe

Delete path and all contents

Parameters

`path` () – path name

14.1.16 `kiwi.privileges` Module

`kiwi.privileges.Privileges`

Bases: `object`

Implements check for root privileges

kiwi.privileges.Privileges.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

14.1.17 `kiwi.runtime_checker` Module

`kiwi.runtime_checker.RuntimeChecker`

Bases: `object`

Implements build consistency checks at runtime

kiwi.runtime_checker.RuntimeChecker.check_appx_naming_conventions_valid

When building wsl images there are some naming conventions that must be fulfilled to run the container on Microsoft Windows

kiwi.runtime_checker.RuntimeChecker.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

kiwi.runtime_checker.RuntimeChecker.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

kiwi.runtime_checker.RuntimeChecker.check_container_tool_chain_installed

When creating container images the specific tools are used in order to import and export OCI or Docker compatible images. This check searches for those tools to be installed in the build system and fails if it can't find them

kiwi.runtime_checker.RuntimeChecker.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.

kiwi.runtime_checker.RuntimeChecker.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.

kiwi.runtime_checker.RuntimeChecker.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.

kiwi.runtime_checker.RuntimeChecker.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.

kiwi.runtime_checker.RuntimeChecker.check_dracut_module_versions_compatible_to_kiwi

KIWI images which makes use of kiwi dracut modules has to use module versions compatible with the version of this KIWI builder code base. This is important to avoid inconsistencies between the way how kiwi includes its own dracut modules and former version of those dracut modules which could be no longer compatible with the builder. Therefore this runtime check maintains a min_version constraint for which we know this KIWI builder to be compatible with.

kiwi.runtime_checker.RuntimeChecker.check_efi_fat_image_has_correct_size

Verify that the efifatimagesize does not exceed the max El Torito load size of 65535 * 512 bytes

kiwi.runtime_checker.RuntimeChecker.check_image_include_repos_publicly_resolvable

Verify that all repos marked with the imageinclude attribute can be resolved into a http based web URL

kiwi.runtime_checker.RuntimeChecker.check_image_type_unique

Verify that the selected image type is unique within the range of the configured types and profiles.

kiwi.runtime_checker.RuntimeChecker.check_image_version_provided

Kiwi requires a <version> element to be specified as part of at least one <preferences> section.

kiwi.runtime_checker.RuntimeChecker.check_include_references_unresolvable

Raise for still included <include> statements as not resolvable. The KIWI XSLT processing replaces the specified include directive(s) with the given file reference(s). If this action did not happen for example on nested includes, it can happen that they stay in the document as sort of waste.

kiwi.runtime_checker.RuntimeChecker.check_initrd_selection_required

If the boot attribute is used without selecting kiwi as the initrd_system, the setting of the boot attribute will not have any effect. We assume that configurations which explicitly specify the boot attribute wants to use the custom kiwi initrd system and not dracut.

kiwi.runtime_checker.RuntimeChecker.check_luksformat_options_valid

Options set via the luksformat element are passed along to the cryptsetup tool. Only options that are known to the tool should be allowed. Thus this runtime check looks up the provided option names if they exist in the cryptsetup version used on the build host

kiwi.runtime_checker.RuntimeChecker.check_mediacheck_installed

If the image description enables the mediacheck attribute the required tools to run this check must be installed on the image build host

kiwi.runtime_checker.RuntimeChecker.check_partuuid_persistency_type_used_with_mbr

The devicepersistency setting by-partuuid can only be used in combination with a partition table type that supports UUIDs. In any other case Linux creates artificial values for PTUUID and PARTUUID from the disk signature which can change without touching the actual partition table. We consider this unsafe and only allow the use of by-partuuid in combination with partition tables that actually supports it properly.

kiwi.runtime_checker.RuntimeChecker.check_repositories_configured

Verify that there are repositories configured

kiwi.runtime_checker.RuntimeChecker.check_swap_name_used_with_lvm

The optional oem-swapname is only effective if used together with the LVM volume manager. A name for the swap space can only be set if it is created as a LVM volume. In any other case the name does not apply to the system

kiwi.runtime_checker.RuntimeChecker.check_target_dir_on_unsupported_filesystem

Raise if the given target dir does not reside on a filesystem that supports all important features like extended permissions(fscaps), ACLs or xattrs.

kiwi.runtime_checker.RuntimeChecker.check_target_directory_not_in_shared_cache

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` () – path name

kiwi.runtime_checker.RuntimeChecker.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

kiwi.runtime_checker.RuntimeChecker.check_volume_setup_defines_multiple_fullsize_volumes

The volume size specification ‘all’ makes this volume to take the rest space available on the system. It’s only allowed to specify one all size volume

kiwi.runtime_checker.RuntimeChecker.check_volume_setup_defines_reserved_labels

kiwi.runtime_checker.RuntimeChecker.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

kiwi.runtime_checker.RuntimeChecker.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

`kiwi.runtime_checker.dracut_module_type`

Bases: `tuple`

kiwi.runtime_checker.dracut_module_type.min_version

Alias for field number 1

kiwi.runtime_checker.dracut_module_type.package

Alias for field number 0

14.1.18 `kiwi.runtime_config` Module

`kiwi.runtime_config.RuntimeConfig`

Bases: `object`

Implements reading of runtime configuration file:

1. Check for –config provided from the CLI
2. ~/.config/kiwi/config.yml
3. /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.

Parameters

`reread` () – reread runtime config

kiwi.runtime_config.RuntimeConfig.get_bundle_compression

Return boolean value to express if the image bundle should contain XZ compressed image results or not.

bundle:

- compress: true|false

If compression of image build results is activated the size of the bundle is smaller and the download speed increases. However the image must be uncompressed before use If no compression is explicitly configured, the provided default value applies

Parameters

`default` () – Default value

Returns

True or False

Return type

bool

kiwi.runtime_config.RuntimeConfig.get_container_compression

Return compression for container images

container:

- compress: xz|none|true|false

if no or invalid configuration data is provided, the default compression from the Defaults class is returned

Returns

True or False

Return type

bool

kiwi.runtime_config.RuntimeConfig.get_credentials_verification_metadata_signing_key_file

Return verification metadata signing key file, used for signature creation of rootfs verification metadata:

credentials:

- verification_metadata_signing_key_file: …

There is no default value for this setting available

Returns

file path name or ‘’

Return type

str

kiwi.runtime_config.RuntimeConfig.get_disabled_runtime_checks

Returns disabled runtime checks. Checks can be disabled with:

runtime_checks:

- disable: check_container_tool_chain_installed

if the provided string does not match any RuntimeChecker method it is just ignored.

kiwi.runtime_config.RuntimeConfig.get_iso_media_tag_tool

Return media tag tool used to checksum iso images

iso:

- media_tag_tool: checkmedia

if no or invalid configuration exists the default media tagger from the Defaults class is returned

Returns

A name

Return type

str

kiwi.runtime_config.RuntimeConfig.get_iso_tool_category

Return tool category which should be used to build iso images

iso:

- tool_category: xorriso

if no or invalid configuration exists the default tool category from the Defaults class is returned

Returns

A name

Return type

str

kiwi.runtime_config.RuntimeConfig.get_mapper_tool

Return partition mapper tool

mapper:

- part_mapper: partx

if no configuration exists the default tool from the Defaults class is returned

Returns

A name

Return type

str

kiwi.runtime_config.RuntimeConfig.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

kiwi.runtime_config.RuntimeConfig.get_obs_api_credentials

Return OBS API credentials if configured:

obs:

- **user:**
  - user_name: user_credentials

Returns

List of Dicts with credentials per user

Return type

list

kiwi.runtime_config.RuntimeConfig.get_obs_api_server_url

Return URL of buildservice API server in:

obs:

- api_url: …

if no configuration exists the API server from the Defaults class is returned

Returns

URL type data

Return type

str

kiwi.runtime_config.RuntimeConfig.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

kiwi.runtime_config.RuntimeConfig.get_oci_archive_tool

Return OCI archive tool which should be used on creation of container archives for OCI compliant images, e.g docker

oci:

- archive_tool: umoci

if no configuration exists the default tool from the Defaults class is returned

Returns

A name

Return type

str

kiwi.runtime_config.RuntimeConfig.get_package_changes

Return boolean value to express if the image build and bundle should contain a .changes file. The .changes file contains the package changelog information from all packages installed into the image.

bundle:

- has_package_changes: true|false

By default the creation is switched on. When building in the Open Build Service the default is switched off because obs provides a .report file containing the same information.

Parameters

`default` () – Default value

Returns

True or False

Return type

bool

kiwi.runtime_config.RuntimeConfig.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

kiwi.runtime_config.RuntimeConfig.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

14.1.19 kiwi.version Module

Global version information used in kiwi and the package

14.1.20 kiwi.xml_description Module

`kiwi.xml_description.XMLDescription`

Bases: `object`

Implements data management for the image description

Supported description markup languages are XML, YAML, JSON and INI. The provided input file is converted into XML, transformed to the current RelaxNG schema via XSLT and validated against this result.

- 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` () – path to description file
- `derived_from` () – path to base description file

kiwi.xml_description.XMLDescription.get_extension_xml_data

Return the xml etree parse result for the specified extension namespace

Parameters

`namespace_name` () – name of the extension namespace

Returns

result of etree.parse

Return type

object

kiwi.xml_description.XMLDescription.load

Read XML description, validate it against the schema and the schematron rules and pass it to the autogenerated(generateDS) parser.

Returns

instance of XML toplevel domain (image)

Return type

object

14.1.21 `kiwi.xml_state` Module

`kiwi.xml_state.ContainerT`

Bases: `NamedTuple`

kiwi.xml_state.ContainerT.backend

Alias for field number 1

kiwi.xml_state.ContainerT.container_file

Alias for field number 2

kiwi.xml_state.ContainerT.fetch_command

Alias for field number 4

kiwi.xml_state.ContainerT.fetch_only

Alias for field number 3

kiwi.xml_state.ContainerT.load_command

Alias for field number 5

kiwi.xml_state.ContainerT.name

Alias for field number 0

`kiwi.xml_state.DracutT`

Bases: `NamedTuple`

kiwi.xml_state.DracutT.drivers

Alias for field number 2

kiwi.xml_state.DracutT.modules

Alias for field number 1

kiwi.xml_state.DracutT.uefi

Alias for field number 0

`kiwi.xml_state.FileT`

Bases: `NamedTuple`

kiwi.xml_state.FileT.owner

Alias for field number 1

kiwi.xml_state.FileT.permissions

Alias for field number 2

kiwi.xml_state.FileT.target

Alias for field number 0

`kiwi.xml_state.XMLState`

Bases: `object`

Implements methods to get stateful information from the XML data

Parameters

- `xml_data` () – parse result from XMLDescription.load()
- `profiles` () – list of used profiles
- `build_type` () – build <type> section reference

kiwi.xml_state.XMLState.add_certificate

Add <certificate name="cert_file"> to main <certificates> section The main section will be created if it does not exist. Also setup the target_distribution in the resulting main section.

kiwi.xml_state.XMLState.add_container_config_label

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` () – the string representing the label name
- `value` () – the value of the label

kiwi.xml_state.XMLState.add_repository

Add a new repository section at the end of the list

Parameters

- `repo_source` () – repository URI
- `repo_type` () – type name defined by schema
- `repo_alias` () – alias name
- `repo_prio` () – priority number, package manager specific
- `repo_imageinclude` () – setup repository inside of the image
- `repo_package_gpgcheck` () – enable/disable package gpg checks
- `repo_signing_keys` () – list of signing key file names
- `components` () – component names for debian repos

- `distribution` () – base distribution name for debian repos
- `repo_gpgcheck` () – enable/disable repo gpg checks

kiwi.xml_state.XMLState.btrfs_default_volume_requested

Check if setting a default volume for btrfs is requested

kiwi.xml_state.XMLState.collection_matches_host_architecture

Tests if the given namedcollection 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

kiwi.xml_state.XMLState.container_matches_host_architecture

Tests if the given container section is applicable for the current host architecture. If no arch attribute is provided in the section it is considered as a match and returns: True.

Parameters

`section` – XML section object

Returns

True or False

Return type

bool

kiwi.xml_state.XMLState.containers_matches_host_architecture

Tests if the given containers section is applicable for the current host architecture. If no arch attribute is provided in the section it is considered as a match and returns: True.

Parameters

`section` – XML section object

Returns

True or False

Return type

bool

kiwi.xml_state.XMLState.copy_bootdelete_packages

Copy packages marked as bootdelete to the packages type = delete section in the target xml state

Parameters

`target_state` () – XMLState instance

kiwi.xml_state.XMLState.copy_bootincluded_archives

Copy archives marked as bootinclude to the packages type = bootstrap section in the target xml state

Parameters

`target_state` () – XMLState instance

kiwi.xml_state.XMLState.copy_bootincluded_packages

Copy packages marked as bootinclude to the packages type = bootstrap 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` () – XMLState instance

kiwi.xml_state.XMLState.copy_bootloader_section

Copy bootloader section from this xml state to the target xml state

Parameters

`target_state` () – XMLState instance

kiwi.xml_state.XMLState.copy_build_type_attributes

Copy specified attributes from this build type section to the target xml state build type section

Parameters

- `attribute_names` () – type section attributes
- `target_state` () – XMLState instance

kiwi.xml_state.XMLState.copy_displayname

Copy image displayname from this xml state to the target xml state

Parameters

`target_state` () – XMLState instance

kiwi.xml_state.XMLState.copy_drivers_sections

Copy drivers sections from this xml state to the target xml state

Parameters

`target_state` () – XMLState instance

kiwi.xml_state.XMLState.copy_machine_section

Copy machine sections from this xml state to the target xml state

Parameters

`target_state` () – XMLState instance

kiwi.xml_state.XMLState.copy_name

Copy image name from this xml state to the target xml state

Parameters

`target_state` () – XMLState instance

kiwi.xml_state.XMLState.copy_oemconfig_section

Copy oemconfig sections from this xml state to the target xml state

Parameters

`target_state` () – XMLState instance

kiwi.xml_state.XMLState.copy_preferences_subsections

Copy subsections of the preferences sections, matching given section names, from this xml state to the target xml state

Parameters

- `section_names` () – preferences subsection names
- `target_state` () – XMLState instance

kiwi.xml_state.XMLState.copy_repository_sections

Copy repository sections from this xml state to the target xml state

Parameters

- `target_state` () – XMLState instance
- `wipe` () – delete all repos in target prior to copy

kiwi.xml_state.XMLState.copy_strip_sections

Copy strip sections from this xml state to the target xml state

Parameters

`target_state` () – XMLState instance

kiwi.xml_state.XMLState.copy_systemdisk_section

Copy systemdisk sections from this xml state to the target xml state

Parameters

`target_state` () – XMLState instance

kiwi.xml_state.XMLState.delete_repository_sections

Delete all repository sections matching configured profiles

kiwi.xml_state.XMLState.delete_repository_sections_used_for_build

Delete all repository sections used to build the image matching configured profiles

kiwi.xml_state.XMLState.get_archives_target_dirs

Dict of archive names and target dirs for packages section(s), if any :return: archive names and its target dir :rtype: dict

kiwi.xml_state.XMLState.get_bootloader_config_options

List of custom options used in the bootloader configuration

kiwi.xml_state.XMLState.get_bootloader_install_options

List of custom options used in the bootloader installation

kiwi.xml_state.XMLState.get_bootloader_options

List of custom options used in the process to run bootloader setup workloads

kiwi.xml_state.XMLState.get_bootloader_shim_options

List of custom options used in the process to setup secure boot

kiwi.xml_state.XMLState.get_bootstrap_archives

List of archive names from the type = ”bootstrap” packages section(s)

Returns

archive names

Return type

list

kiwi.xml_state.XMLState.get_bootstrap_archives_target_dirs

Dict of archive names and target dirs from the type = ”bootstrap” packages section(s) :return: archive names and its target dir :rtype: dict

kiwi.xml_state.XMLState.get_bootstrap_collection_type

Collection type for packages sections matching type = ”bootstrap”

Returns

collection type name

Return type

str

kiwi.xml_state.XMLState.get_bootstrap_collections

List of collection names from the packages sections matching type = ”bootstrap”

Returns

collection names

Return type

list

kiwi.xml_state.XMLState.get_bootstrap_files

List of file names from the type = ”bootstrap” packages section(s)

Returns

file names

Return type

dict

kiwi.xml_state.XMLState.get_bootstrap_ignore_packages

List of ignore package names from the packages sections matching type = ”image” and type = build_type

Returns

package names

Return type

list

kiwi.xml_state.XMLState.get_bootstrap_package_name

bootstrap_package name from type = ”bootstrap” packages section

Returns

bootstrap_package name

Return type

str

kiwi.xml_state.XMLState.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

Parameters

`plus_packages` () – list of additional packages

Returns

package names

Return type

list

kiwi.xml_state.XMLState.get_bootstrap_packages_sections

List of packages sections matching type = ”bootstrap”

Returns

list of < packages > section reference(s)

Return type

list

kiwi.xml_state.XMLState.get_bootstrap_products

List of product names from the packages sections matching type = ”bootstrap”

Returns

product names

Return type

list

kiwi.xml_state.XMLState.get_build_type_bootloader_bls

Return bootloader bls setting for selected build type

Returns

True or False

Return type

bool

kiwi.xml_state.XMLState.get_build_type_bootloader_console

Return bootloader console setting for selected build type

Returns

list of console settings for output (first element) and input (second element)

Return type

list

kiwi.xml_state.XMLState.get_build_type_bootloader_name

Return bootloader name for selected build type

Returns

bootloader name

Return type

str

kiwi.xml_state.XMLState.get_build_type_bootloader_section

First bootloader section from the build type section

Returns

<bootloader> section reference

Return type

xml_parse::bootloader

kiwi.xml_state.XMLState.get_build_type_bootloader_securelinux_section

First securelinux section from the build type bootloader section

Returns

<securelinux> section reference

Return type

xml_parse::securelinux

kiwi.xml_state.XMLState.get_build_type_bootloader_serial_line_setup

Return bootloader serial line setup parameters for the selected build type

Returns

serial line setup

Return type

str

kiwi.xml_state.XMLState.get_build_type_bootloader_settings_section

First bootloadersettings section from the build type bootloader section

Returns

<bootloadersettings> section reference

Return type

xml_parse::bootloadersettings

kiwi.xml_state.XMLState.get_build_type_bootloader_targettype

Return bootloader target type setting. Only relevant for the zipl bootloader because zipl is installed differently depending on the storage target it runs later

Returns

target type string

Return type

str

kiwi.xml_state.XMLState.get_build_type_bootloader_timeout

Return bootloader timeout setting for selected build type

Returns

timeout string

Return type

str

kiwi.xml_state.XMLState.get_build_type_bootloader_timeout_style

Return bootloader timeout style setting for selected build type

Returns

timeout_style string

Return type

str

kiwi.xml_state.XMLState.get_build_type_bootloader_use_disk_password

Indicate whether the bootloader configuration should use the password protecting the encrypted root volume.

Returns

True|False

Return type

bool

kiwi.xml_state.XMLState.get_build_type_bundle_format

Return bundle_format for build type

The bundle_format string is validated against the available name tags from kiwi.system.result::result_name_tags.

Returns

bundle format string

Return type

str

kiwi.xml_state.XMLState.get_build_type_containerconfig_section

First containerconfig section from the build type section

Returns

<containerconfig> section reference

Return type

xml_parse::containerconfig

kiwi.xml_state.XMLState.get_build_type_format_options

Disk format options returned as a dictionary

Returns

format options

Return type

dict

kiwi.xml_state.XMLState.get_build_type_machine_section

First machine section from the build type section

Returns

<machine> section reference

Return type

xml_parse::machine

kiwi.xml_state.XMLState.get_build_type_name

Default build type name

Returns

Content of image attribute from build type

Return type

str

kiwi.xml_state.XMLState.get_build_type_oemconfig_section

First oemconfig section from the build type section

Returns

<oemconfig> section reference

Return type

xml_parse::oemconfig

kiwi.xml_state.XMLState.get_build_type_partitions_section

First partitions section from the build type section

Returns

<partitions> section reference

Return type

xml_parse::partitions

kiwi.xml_state.XMLState.get_build_type_size

Size information from the build type section. If no unit is set the value is treated as mbytes

Parameters

`include_unpartitioned` () – sets if the unpartitioned area should be included in the computed size or not

Returns

mbytes

Return type

int

kiwi.xml_state.XMLState.get_build_type_spare_part_fs_attributes

Build type specific list of filesystem attributes applied to the spare partition.

Returns

list of strings or empty list

Return type

list

kiwi.xml_state.XMLState.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

kiwi.xml_state.XMLState.get_build_type_system_disk_section

First system disk section from the build type section

Returns

<systemdisk> section reference

Return type

xml_parse::systemdisk

kiwi.xml_state.XMLState.get_build_type_unpartitioned_bytes

Size of the unpartitioned area for image in megabytes

Returns

mbytes

Return type

int

kiwi.xml_state.XMLState.get_build_type_vagrant_config_section

First vagrantconfig section from the build type section

Returns

<vagrantconfig> section reference

Return type

xml_parse::vagrantconfig

kiwi.xml_state.XMLState.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

kiwi.xml_state.XMLState.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

kiwi.xml_state.XMLState.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

kiwi.xml_state.XMLState.get_build_type_vmnic_entries

vmnic section(s) from the first machine section in the build type section

Returns

list of <vmnic> section reference(s)

Return type

list

kiwi.xml_state.XMLState.get_certificates

Read list of certificates

kiwi.xml_state.XMLState.get_certificates_target_distribution

Read CA target distribution

kiwi.xml_state.XMLState.get_collection_modules

Dict of collection modules to enable and/or disable

Returns

Dict of the form:

```
{
    'enable': [
        "module:stream", "module"
    ],
    'disable': [
        "module"
    ]
}
```

Return type

dict

kiwi.xml_state.XMLState.get_collection_type

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` () – type name from packages section

Returns

collection type name

Return type

str

kiwi.xml_state.XMLState.get_collections

List of collection names from the packages sections matching type = section_type and type = build_type

Returns

collection names

Return type

list

kiwi.xml_state.XMLState.get_container_config

Dictionary of containerconfig information

Takes attributes and subsection data from the selected <containerconfig> section and stores it in a dictionary

kiwi.xml_state.XMLState.get_containers

kiwi.xml_state.XMLState.get_containers_sections

List of all containers sections for the selected profiles that matches the host architecture

Returns

<containers> section reference(s)

Return type

list

kiwi.xml_state.XMLState.get_derived_from_image_uri

Uri object(s) of derived image if configured

Specific image types can be based on one ore more derived images. This method returns the location of this image(s) when configured in the XML description

Returns

List of Uri instances

Return type

list

kiwi.xml_state.XMLState.get_description_section

The description section

Returns

description_type tuple providing the elements author contact and specification

Return type

tuple

kiwi.xml_state.XMLState.get_disk_start_sector

First disk sector number to be used by the first disk partition.

Returns

number

Return type

int

kiwi.xml_state.XMLState.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

kiwi.xml_state.XMLState.get_dracut_config

Get dracut initrd config for the specified action

kiwi.xml_state.XMLState.get_drivers_list

List of driver names from all drivers sections matching configured profiles

Returns

driver names

Return type

list

kiwi.xml_state.XMLState.get_fs_create_option_list

List of root filesystem creation options

The list contains elements with the information from the fscreateoptions attribute string that got split into its substring components

Returns

list with create options

Return type

list

kiwi.xml_state.XMLState.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

kiwi.xml_state.XMLState.get_host_key_certificates

kiwi.xml_state.XMLState.get_ignore_packages

List of ignore package names from the packages sections matching section_type and type = build_type

Returns

package names

Return type

list

kiwi.xml_state.XMLState.get_image_packages_sections

List of packages sections matching type = ”image”

Returns

list of <packages> section reference(s)

Return type

list

kiwi.xml_state.XMLState.get_image_version

Image version from preferences section.

Multiple occurences 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

kiwi.xml_state.XMLState.get_include_section_reference_file_names

List of all <include> section file name references

Returns

List[str]

Return type

list

kiwi.xml_state.XMLState.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

kiwi.xml_state.XMLState.get_installmedia_initrd_drivers

Gets the list of drivers to append in installation initrds

Returns

a list of dracut driver names

Return type

list

kiwi.xml_state.XMLState.get_installmedia_initrd_modules

Gets the list of modules to append in installation initrds

Returns

a list of dracut module names

Return type

list

kiwi.xml_state.XMLState.get_locale

Gets list of locale names if configured. Takes the first locale setup from the existing preferences sections into account.

Returns

List of names or None

Return type

list|None

kiwi.xml_state.XMLState.get_luks_credentials

Return key or passphrase credentials to open the luks pool

Returns

data

Return type

str

kiwi.xml_state.XMLState.get_luks_format_options

Return list of luks format options

Returns

list of options

Return type

list

kiwi.xml_state.XMLState.get_oemconfig_oem_multipath_scan

State value to activate multipath maps. Returns a boolean value if specified or False

Returns

Content of <oem-multipath-scan> section value

Return type

bool

kiwi.xml_state.XMLState.get_oemconfig_oem_resize

State value to activate/deactivate disk resize. Returns a boolean value if specified or True to set resize on by default

Returns

Content of <oem-resize> section value

Return type

bool

kiwi.xml_state.XMLState.get_oemconfig_oem_systemsize

State value to retrieve root partition size

Returns

Content of <oem-systemsize> section value

Return type

int

kiwi.xml_state.XMLState.get_oemconfig_swap_mbytes

Return swapsize in MB if requested or None

Operates on the value of oem-swap and if set to true returns the given size or the default value.

Returns

Content of <oem-swapsize> section value or default

Return type

int

kiwi.xml_state.XMLState.get_oemconfig_swap_name

Return the swap space name

Operates on the value of oem-swapname and if set returns the configured name or the default name: LVSwap

The name of the swap space is used only if the image is configured to use the LVM volume manager. In this case swap is a volume and the volume takes a name. In any other case the given name will have no effect.

Returns

Content of <oem-swapname> section value or default

Return type

str

kiwi.xml_state.XMLState.get_package_manager

Get configured package manager from selected preferences section

Returns

Content of the <packagemanager> section

Return type

str

kiwi.xml_state.XMLState.get_package_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 specfies an architecture, it is only taken if the host architecture matches the configured architecture

Parameters

`packages_sections` () – <packages>

Returns

Contains list of package_type tuples

```
[package_type(packages_section=object, package_section=object)]
```

Return type

list

kiwi.xml_state.XMLState.get_packages_sections

List of packages sections matching given section type(s)

Parameters

`section_types` () – type name(s) from packages sections

Returns

list of <packages> section reference(s)

Return type

list

kiwi.xml_state.XMLState.get_partitions

Dictionary of configured partitions.

Each entry in the dict references a ptable_entry_type Each key in the dict references the name of the partition entry as handled by KIWI

Returns

Contains dict of ptable_entry_type tuples

```
{
    'NAME': ptable_entry_type(
        mbsize=int,
        clone=int,
        partition_name=str,
        partition_type=str,
        partition_id=Optional[int],
        mountpoint=str,
        filesystem=str,
        label=str
    )
}
```

Return type

dict

kiwi.xml_state.XMLState.get_preferences_sections

All preferences sections for the selected profiles that match the host architecture

Returns

list of <preferences> section reference(s)

Return type

list

kiwi.xml_state.XMLState.get_products

List of product names from the packages sections matching type = section_type and type = build_type

Parameters

`section_type` () – type name from packages section

Returns

product names

Return type

list

kiwi.xml_state.XMLState.get_release_version

Get configured release version from selected preferences section

Returns

Content of the <release-version> section or ''

Return type

str

kiwi.xml_state.XMLState.get_repositories_signing_keys

Get list of signing keys specified on the repositories

kiwi.xml_state.XMLState.get_repository_sections

List of all repository sections for the selected profiles that matches the host architecture

Returns

<repository> section reference(s)

Return type

list

kiwi.xml_state.XMLState.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

kiwi.xml_state.XMLState.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

kiwi.xml_state.XMLState.get_root_filesystem_uuid

Return preserved UUID

kiwi.xml_state.XMLState.get_root_partition_uuid

Return preserved PARTUUID

kiwi.xml_state.XMLState.get_rpm_check_signatures

Gets the rpm-check-signatures configuration flag. Returns False if not present.

Returns

True or False

Return type

bool

kiwi.xml_state.XMLState.get_rpm_excludedocs

Gets the rpm-excludedocs configuration flag. Returns False if not present.

Returns

True or False

Return type

bool

kiwi.xml_state.XMLState.get_rpm_locale

Gets list of locale names to filter out by rpm if rpm-locale-filtering is switched on the the list always contains: [POSIX, C, C.UTF-8] and is extended by the optionaly configured locale

Returns

List of names or None

Return type

list|None

kiwi.xml_state.XMLState.get_rpm_locale_filtering

Gets the rpm-locale-filtering configuration flag. Returns False if not present.

Returns

True or False

Return type

bool

kiwi.xml_state.XMLState.get_strip_files_to_delete

Items to delete from strip section

Returns

item names

Return type

list

kiwi.xml_state.XMLState.get_strip_libraries_to_keep

Libraries to keep from strip section

Returns

librarie names

Return type

list

kiwi.xml_state.XMLState.get_strip_list

List of strip names matching the given section type and profiles

Parameters

`section_type` () – type name from packages section

Returns

strip names

Return type

list

kiwi.xml_state.XMLState.get_strip_tools_to_keep

Tools to keep from strip section

Returns

tool names

Return type

list

kiwi.xml_state.XMLState.get_system_archives

List of archive names from the packages sections matching type = ”image” and type = build_type

Returns

archive names

Return type

list

kiwi.xml_state.XMLState.get_system_archives_target_dirs

Dict of archive names and its target dir from the packages sections matching type = ”image” and type = build_type :return: archive names and its target dir :rtype: dict

kiwi.xml_state.XMLState.get_system_collection_type

Collection type for packages sections matching type = ”image”

Returns

collection type name

Return type

str

kiwi.xml_state.XMLState.get_system_collections

List of collection names from the packages sections matching type = ”image”

Returns

collection names

Return type

list

kiwi.xml_state.XMLState.get_system_files

List of file names from the packages sections matching type = ”image” and type = build_type

Returns

file names

Return type

dict

kiwi.xml_state.XMLState.get_system_files_ignore_packages

List of ignore package names from the type = ”systemfiles” packages section(s)

Returns

package names

Return type

list

kiwi.xml_state.XMLState.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

kiwi.xml_state.XMLState.get_system_packages

List of package names from the packages sections matching type = ”image” and type = build_type

Returns

package names

Return type

list

kiwi.xml_state.XMLState.get_system_products

List of product names from the packages sections matching type = ”image”

Returns

product names

Return type

list

kiwi.xml_state.XMLState.get_to_become_deleted_packages

List of package names from the type = ”delete” or type = ”uninstall” packages section(s)

Parameters

`force` () – return “delete” type if True, “uninstall” type otherwise

Returns

package names

Return type

list

kiwi.xml_state.XMLState.get_user_groups

List of group names matching specified user

Each entry in the list is the name of a group and optionally its group ID separated by a colon, 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

kiwi.xml_state.XMLState.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

kiwi.xml_state.XMLState.get_users_sections

All users sections for the selected profiles

Returns

list of <users> section reference(s)

Return type

list

kiwi.xml_state.XMLState.get_vagrant_config_virtualbox_guest_additions

Attribute virtualbox_guest_additions_present from the first vagrantconfig section.

Returns

True|False

Return type

bool

kiwi.xml_state.XMLState.get_volume_group_name

Volume group name from selected <systemdisk> section

Returns

volume group name

Return type

str

kiwi.xml_state.XMLState.get_volume_management

Provides information which volume management system is used

Returns

name of volume manager

Return type

str

kiwi.xml_state.XMLState.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,
        parent=volume_parent,
        size=volume_size,
        realpath=path,
        mountpoint=path,
        fullsize=True,
        label=volume_label,
        attributes=['no-copy-on-write'],
        is_root_volume=True|False
    )
]
```

Return type

list

kiwi.xml_state.XMLState.is_xen_guest

Check if build type setup specifies a Xen Guest (domX) The check is based on the architecture, the firmware and xen_loader configuration values:

- We only support Xen setup on the x86_64 architecture
- Firmware pointing to ec2 means the image is targeted to run in Amazon EC2 which is a Xen guest
- Machine setup with a xen_loader attribute also indicates a Xen guest target

Returns

True or False

Return type

bool

kiwi.xml_state.XMLState.is_xen_server

Check if build type domain setup specifies a Xen Server (dom0)

Returns

True or False

Return type

bool

kiwi.xml_state.XMLState.package_matches_host_architecture

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

kiwi.xml_state.XMLState.preferences_matches_host_architecture

Tests if the given preferences 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

kiwi.xml_state.XMLState.profile_matches_host_architecture

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

kiwi.xml_state.XMLState.repository_matches_host_architecture

Tests if the given repository 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

kiwi.xml_state.XMLState.resolve_this_path

Resolve any this:// repo source path into the path representing the target inside of the image description directory

kiwi.xml_state.XMLState.set_container_config_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` () – tag name

kiwi.xml_state.XMLState.set_derived_from_image_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` () – URI

kiwi.xml_state.XMLState.set_repository

Overwrite repository data of the first repository

Parameters

- `repo_source` () – repository URI
- `repo_type` () – type name defined by schema

- `repo_alias` () – alias name
- `repo_prio` () – priority number, package manager specific
- `repo_imageinclude` () – setup repository inside of the image
- `repo_package_gpgcheck` () – enable/disable package gpg checks
- `repo_signing_keys` () – list of signing key file names
- `components` () – component names for debian repos
- `distribution` () – base distribution name for debian repos
- `repo_gpgcheck` () – enable/disable repo gpg checks

kiwi.xml_state.XMLState.set_root_filesystem_uuid

Store UUID provided in uuid as state information

Parameters

`uuid` () – UUID

kiwi.xml_state.XMLState.set_root_partition_uuid

Store PARTUUID provided in uuid as state information

Parameters

`uuid` () – PARTUUID

kiwi.xml_state.XMLState.users_matches_host_architecture

Tests if the given users 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

kiwi.xml_state.XMLState.volume_matches_host_architecture

Tests if the given volume 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

`kiwi.xml_state.description_type`

Bases: `tuple`

kiwi.xml_state.description_type.author

Alias for field number 0

kiwi.xml_state.description_type.contact

Alias for field number 1

kiwi.xml_state.description_type.specification

Alias for field number 2

`kiwi.xml_state.package_type`

Bases: `tuple`

kiwi.xml_state.package_type.package_section

Alias for field number 1

kiwi.xml_state.package_type.packages_section

Alias for field number 0

`kiwi.xml_state.size_type`

Bases: `tuple`

kiwi.xml_state.size_type.additive

Alias for field number 1

kiwi.xml_state.size_type.mbytes

Alias for field number 0

`kiwi.xml_state.volume_type`

Bases: `tuple`

kiwi.xml_state.volume_type.attributes
: Alias for field number 7

kiwi.xml_state.volume_type.fullsize
: Alias for field number 5

kiwi.xml_state.volume_type.is_root_volume
: Alias for field number 8

kiwi.xml_state.volume_type.label
: Alias for field number 6

kiwi.xml_state.volume_type.mountpoint
: Alias for field number 4

kiwi.xml_state.volume_type.name
: Alias for field number 0

kiwi.xml_state.volume_type.parent
: Alias for field number 1

kiwi.xml_state.volume_type.realpath
: Alias for field number 3

kiwi.xml_state.volume_type.size
: Alias for field number 2

14.1.22 Module Contents

14.2 kiwi.archive Package

14.2.1 Submodules

14.2.2 `kiwi.archive.cpio` Module

`kiwi.archive.cpio.ArchiveCpio`
: Bases: `object`

Extraction/Creation of cpio archives

Parameters

`filename` () – filename to use for archive extraction or creation

kiwi.archive.cpio.ArchiveCpio.create

Create cpio archive

Parameters

- `source_dir` () – data source directory
- `exclude` () – list of excluded items

kiwi.archive.cpio.ArchiveCpio.extract

Extract cpio archive contents

Parameters

`dest_dir` () – target data directory

14.2.3 `kiwi.archive.tar` Module

`kiwi.archive.tar.ArchiveTar`

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` () – filename to use for archive extraction or creation
- `create_from_file_list` () – use file list not entire directory to create the archive
- `file_list` () – list of files and directorie names to archive

kiwi.archive.tar.ArchiveTar.append_files

Append files to an already existing uncompressed tar archive

Parameters

- `source_dir` () – data source directory
- `files_to_append` () – list of items to append
- `options` () – custom options

kiwi.archive.tar.ArchiveTar.create

Create uncompressed tar archive

Parameters

- `source_dir` () – data source directory
- `exclude` () – list of excluded items
- `options` () – custom creation options

kiwi.archive.tar.ArchiveTar.create_gnu_gzip_compressed

Create gzip compressed tar archive

Parameters

- `source_dir` () – data source directory
- `exclude` () – list of excluded items

kiwi.archive.tar.ArchiveTar.create_xz_compressed

Create XZ compressed tar archive

Parameters

- `source_dir` () – data source directory
- `exclude` () – list of excluded items
- `options` () – custom tar creation options
- `xz_options` () – custom xz compression options

kiwi.archive.tar.ArchiveTar.extract

Extract tar archive contents

Parameters

`dest_dir` () – target data directory

14.2.4 Module Contents

14.3 kiwi.boot.image Package

14.3.1 Submodules

14.3.2 kiwi.boot.image.base Module

`kiwi.boot.image.base.BootImageBase`

Bases: `object`

Base class for boot image(initrd) task

kiwi.boot.image.base.BootImageBase.add_argument

Add caller argument to boot image creation tool

Parameters

- `option` () – argument name
- `value` () – optional argument value

kiwi.boot.image.base.BootImageBase.cleanup

Cleanup temporary boot image data if any

kiwi.boot.image.base.BootImageBase.create_initrd

Implements creation of the initrd

Parameters

- `mbrid` (`kiwi.system.identifier.SystemIdentifier`) – instance of SystemIdentifier
- `basename` () – base initrd file name
- `install_initrd` () – installation media initrd

Implementation in specialized boot image class

kiwi.boot.image.base.BootImageBase.create_uki

Create UKI EFI binary

Implementation in specialized boot image class

kiwi.boot.image.base.BootImageBase.get_boot_description_directory

Provide path to the boot image XML description

Returns

path name

Return type

str

kiwi.boot.image.base.BootImageBase.get_boot_names

Provides kernel and initrd names for the boot image

Returns

Contains boot_names_type tuple

```
boot_names_type(
    kernel_name='INSTALLED_KERNEL',
    initrd_name='DRACUT_OUTPUT_NAME',
    kernel_version='KERNEL_VERSION',
    kernel_filename='KERNEL_FILE_NAME'
)
```

Return type

kiwi.boot.image.base.boot_names_type

kiwi.boot.image.base.BootImageBase.has_initrd_support

Indicates if this instance supports actual creation of an initrd

The method needs to be overwritten by the subclass implementing preparation and creation of an initrd

kiwi.boot.image.base.BootImageBase.import_system_description_elements

Copy information from the system image relevant to create the boot image to the boot image state XML description

kiwi.boot.image.base.BootImageBase.include_driver

Include driver to the boot image

For kiwi boot no drivers configuration is required. Thus in such a case this method is a noop.

Parameters

- `driver` () – driver to include
- `install_media` () – include the driver for install initrds

kiwi.boot.image.base.BootImageBase.include_file

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` () – file path name
- `install_media` () – include also for installation media initrd

kiwi.boot.image.base.BootImageBase.include_module

Include module to boot image

For kiwi boot no modules configuration is required. Thus in such a case this method is a noop.

Parameters

- `module` () – module to include
- `install_media` () – include the module for install initrds

kiwi.boot.image.base.BootImageBase.is_prepared

Check if initrd system is prepared.

Returns

True or False

Return type

bool

kiwi.boot.image.base.BootImageBase.load_boot_xml_description

Load the boot image description referenced by the system image description boot attribute

kiwi.boot.image.base.BootImageBase.omit_driver

Omit driver to the boot image

For kiwi boot no drivers configuration is required. Thus in such a case this method is a noop.

Parameters

- `driver` () – driver to omit
- `install_media` () – omit the driver for install initrds

kiwi.boot.image.base.BootImageBase.omit_module

Omit module to boot image

For kiwi boot no modules configuration is required. Thus in such a case this method is a noop.

Parameters

- `module` () – module to omit
- `install_media` () – omit the module for install initrds

kiwi.boot.image.base.BootImageBase.post_init

Post initialization method

Implementation in specialized boot image class

kiwi.boot.image.base.BootImageBase.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.base.BootImageBase.set_static_modules

Set static modules list for boot image

For kiwi boot no modules configuration is required. Thus in such a case this method is a noop.

Parameters

- `modules` () – list of modules to include
- `install_media` () – lists the modules for install initrds

kiwi.boot.image.base.BootImageBase.write_system_config_file

Writes relevant boot image configuration into configuration file that will be part of the system image.

This is used to configure any further boot image rebuilds after deployment. For instance, initrds recreated on kernel update.
For kiwi boot no specific configuration is required for initrds recreation, thus this method is a noop in that case.

Parameters

- `config` () – dictonary including configuration parameters
- `config_file` () – configuration file to write

`kiwi.boot.image.base.boot_names_type`

Bases: `tuple`

kiwi.boot.image.base.boot_names_type.initrd_name

Alias for field number 1

kiwi.boot.image.base.boot_names_type.kernel_filename

Alias for field number 3

kiwi.boot.image.base.boot_names_type.kernel_name

Alias for field number 0

kiwi.boot.image.base.boot_names_type.kernel_version

Alias for field number 2

14.3.3 `kiwi.boot.image.dracut` Module

`kiwi.boot.image.dracut.BootImageDracut`

Bases: `BootImageBase` (#kiwi.boot.image.base.BootImageBase)

Implements creation of dracut boot(initrd) images.

kiwi.boot.image.dracut.BootImageDracut.add_argument

Add caller argument to boot image creation tool

Parameters

- `option` () – argument name
- `value` () – optional argument value

kiwi.boot.image.dracut.BootImageDracut.create_initrd

Create kiwi .profile environment to be included in dracut initrd. Call dracut as chroot operation to create the initrd and move the result into the image build target directory

Parameters

- `mbrid` (`kiwi.system.identifier.SystemIdentifier`) – unused
- `basename` () – base initrd file name
- `install_initrd` () – unused

kiwi.boot.image.dracut.BootImageDracut.create_uki

Create UKI EFI binary

kiwi.boot.image.dracut.BootImageDracut.has_initrd_support

This instance supports initrd preparation and creation

kiwi.boot.image.dracut.BootImageDracut.include_driver

Include driver to dracut boot image

Parameters

- `driver` () – driver to include
- `install_media` () – unused

kiwi.boot.image.dracut.BootImageDracut.include_file

Include file to dracut boot image

Parameters

- `filename` () – file path name
- `install_media` () – unused

kiwi.boot.image.dracut.BootImageDracut.include_module

Include module to dracut boot image

Parameters

- `module` () – module to include
- `install_media` () – unused

kiwi.boot.image.dracut.BootImageDracut.omit_driver

Omit driver to dracut boot image

Parameters

- `driver` () – driver to omit
- `install_media` () – unused

kiwi.boot.image.dracut.BootImageDracut.omit_module

Omit module to dracut boot image

Parameters

- `module` () – module to omit
- `install_media` () – unused

kiwi.boot.image.dracut.BootImageDracut.post_init

Post initialization method

Initialize empty list of dracut caller options

kiwi.boot.image.dracut.BootImageDracut.prepare

Prepare dracut caller environment

- Setup machine_id(s) to be generic and rebuild by dracut on boot

kiwi.boot.image.dracut.BootImageDracut.set_static_modules

Set static dracut modules list for boot image

Parameters

- `modules` () – list of the modules to include
- `install_media` () – unused

kiwi.boot.image.dracut.BootImageDracut.write_system_config_file

Writes modules configuration into a dracut configuration file.

Parameters

- `config` () – a dictionary containing the modules to add and omit
- `conf_file` () – configuration file to write

14.3.4 kiwi.boot.image.builtin_kiwi Module

kiwi.boot.image.builtin_kiwi.BootImageKiwi

Bases: BootImageBase (#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.

kiwi.boot.image.builtin_kiwi.BootImageKiwi.cleanup

Cleanup temporary boot image data if any

kiwi.boot.image.builtin_kiwi.BootImageKiwi.create_initrd

Create initrd from prepared boot system tree and compress the result

Parameters

- mbrid (kiwi.system.identifier.SystemIdentifier) – instance of ImageIdentifier
- basename () – base initrd file name
- install_initrd () – installation media initrd

kiwi.boot.image.builtin_kiwi.BootImageKiwi.has_initrd_support

This instance supports initrd preparation and creation

kiwi.boot.image.builtin_kiwi.BootImageKiwi.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

kiwi.boot.image.builtin_kiwi.BootImageKiwi.prepare

Prepare new root system suitable to create a kiwi initrd from it

14.3.5 Module Contents

kiwi.boot.image.BootImage

Bases: object

BootImge Factory

Parameters

- `xml_state` () – Instance of `XMLState`
- `target_dir` () – target dir to store the initrd
- `root_dir` () – system image root directory
- `signing_keys` () – list of package signing keys

kiwi.boot.image.BootImage.new

14.4 kiwi.bootloader.config Package

14.4.1 Submodules

14.4.2 `kiwi.bootloader.config.base` Module

`kiwi.bootloader.config.base.BootLoaderConfigBase`

Bases: `ABC`

Base class for bootloader configuration

Parameters

- `xml_state` () – instance of `XMLState`
- `root_dir` () – root directory path name
- `custom_args` () – custom bootloader arguments dictionary

kiwi.bootloader.config.base.BootLoaderConfigBase.create_efi_path

Create standard EFI boot directory structure

Parameters

`in_sub_dir` () – toplevel directory

Returns

Full qualified EFI boot path

Return type

str

kiwi.bootloader.config.base.BootLoaderConfigBase.failsafe_boot_entry_requested

Check if a failsafe boot entry is requested

Returns

True or False

Return type

bool

kiwi.bootloader.config.base.BootLoaderConfigBase.get_boot_cmdline

Boot commandline arguments passed to the kernel

Parameters

- `boot_device` () – boot device node. If no extra boot device exists then boot device equals root device. In case of an overlay setup the boot device equals the readonly root device
- `write_device` () – optional overlay write device node

Returns

kernel boot arguments

Return type

str

kiwi.bootloader.config.base.BootLoaderConfigBase.get_boot_path

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` () – target name: disk|iso

Returns

path name

Return type

str

kiwi.bootloader.config.base.BootLoaderConfigBase.get_boot_theme

Bootloader Theme name

Returns

theme name

Return type

str

kiwi.bootloader.config.base.BootLoaderConfigBase.get_boot_timeout_seconds

Bootloader timeout in seconds

If no timeout is specified the default timeout applies

Returns

timeout seconds

Return type

int

kiwi.bootloader.config.base.BootLoaderConfigBase.get_continue_on_timeout

Check if the boot should continue after boot timeout or not

Returns

True or False

Return type

bool

kiwi.bootloader.config.base.BootLoaderConfigBase.get_gfxmode

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` () – bootloader name

Returns

boot graphics mode

Return type

str

kiwi.bootloader.config.base.BootLoaderConfigBase.get_install_image_boot_default

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 controlls 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` () – bootloader name

Returns

menu name or id

Return type

str

kiwi.bootloader.config.base.BootLoaderConfigBase.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

kiwi.bootloader.config.base.BootLoaderConfigBase.get_menu_entry_title

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` () – indicate to add built type into title text

Returns

title text

Return type

str

kiwi.bootloader.config.base.BootLoaderConfigBase.post_init

Post initialization method

Store custom arguments by default

Parameters

`custom_args` () – custom bootloader arguments

kiwi.bootloader.config.base.BootLoaderConfigBase.quote_title

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` () – title name

Returns

quoted text

Return type

str

kiwi.bootloader.config.base.BootLoaderConfigBase.setup_disk_boot_images

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` () – boot device UUID
- `efi_uuid` () – EFI device UUID
- `lookup_path` () – custom module lookup path

Implementation in specialized bootloader class required

kiwi.bootloader.config.base.BootLoaderConfigBase.setup_disk_image_config

Create boot config file to boot from disk.

Parameters

- `boot_uuid` () – boot device UUID
- `root_uuid` () – root device UUID
- `hypervisor` () – hypervisor name
- `kernel` () – kernel name
- `initrd` () – initrd name
- `boot_options` () – custom options dictionary required to setup the bootloader. The scope of the options covers all information needed to setup and configure the bootloader and gets effective in the individual implementation. boot_options should not be mixed up with commandline options used at boot time. This information is provided from the get_*_cmdline methods. The contents of the dictionary can vary between bootloaders or even not be needed

Implementation in specialized bootloader class required

kiwi.bootloader.config.base.BootLoaderConfigBase.setup_install_boot_images

Create bootloader images for ISO boot an install media

Parameters

- `mbrid` () – mbrid file name on boot device
- `lookup_path` () – custom module lookup path

Implementation in specialized bootloader class required

kiwi.bootloader.config.base.BootLoaderConfigBase.setup_install_image_config

Create boot config file to boot from install media in EFI mode.

Parameters

- `mbrid` () – mbrid file name on boot device
- `hypervisor` () – hypervisor name
- `kernel` () – kernel name
- `initrd` () – initrd name

Implementation in specialized bootloader class required

kiwi.bootloader.config.base.BootLoaderConfigBase.setup_live_boot_images

Create bootloader images for ISO boot a live ISO image

Parameters

- `mbrid` () – mbrid file name on boot device
- `lookup_path` () – custom module lookup path

Implementation in specialized bootloader class required

kiwi.bootloader.config.base.BootLoaderConfigBase.setup_live_image_config

Create boot config file to boot live ISO image in EFI mode.

Parameters

- `mbrid` () – mbrid file name on boot device
- `hypervisor` () – hypervisor name
- `kernel` () – kernel name
- `initrd` () – initrd name

Implementation in specialized bootloader class required

kiwi.bootloader.config.base.BootLoaderConfigBase.setup_sysconfig_bootloader

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

Implementation in specialized bootloader class required

kiwi.bootloader.config.base.BootLoaderConfigBase.write

Write config data to config file.

Implementation in specialized bootloader class required

kiwi.bootloader.config.base.BootLoaderConfigBase.write_meta_data

Write bootloader setup meta data files

Parameters

- `root_device` () – root device node
- `write_device` () – overlay root write device node
- `boot_options` () – kernel options as string

Implementation in specialized bootloader class optional

14.4.3 kiwi.bootloader.config.grub2 Module

kiwi.bootloader.config.grub2.BootLoaderConfigGrub2

Bases: BootLoaderConfigBase (#kiwi.bootloader.config.base.BootLoaderConfigBase)

grub2 bootloader configuration.

kiwi.bootloader.config.grub2.BootLoaderConfigGrub2.post_init

grub2 post initialization method

Parameters

custom_args () –
Contains grub config arguments

```
{'grub_directory_name': 'grub|grub2'}
```

kiwi.bootloader.config.grub2.BootLoaderConfigGrub2.setup_disk_boot_images

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 () – boot device UUID
- lookup_path () – custom module lookup path

kiwi.bootloader.config.grub2.BootLoaderConfigGrub2.setup_disk_image_config

Create grub2 config file to boot from disk using grub2-mkconfig

Parameters

- boot_uuid () – unused
- root_uuid () – unused
- hypervisor () – unused
- kernel () – unused

- `initrd` () – unused
- `boot_options` ()

options dictionary that has to contain the root and boot device and optional volume configuration. KIWI has to mount the system prior to run grub2-mkconfig.

```
{
    'root_device': string,
    'boot_device': string,
    'efi_device': string,
    'system_volumes':
        volume_manager_instance.get_volumes(),
    'system_root_volume':
        volume_manager_instance.get_root_volume_name()
}
```

kiwi.bootloader.config.grub2.BootLoaderConfigGrub2.setup_install_boot_images

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` () – mbrid file name on boot device
- `lookup_path` () – custom module lookup path

kiwi.bootloader.config.grub2.BootLoaderConfigGrub2.setup_install_image_config

Create grub2 config file to boot from an ISO install image

Parameters

- `mbrid` () – mbrid file name on boot device
- `hypervisor` () – hypervisor name
- `kernel` () – kernel name
- `initrd` () – initrd name

kiwi.bootloader.config.grub2.BootLoaderConfigGrub2.setup_live_boot_images

Create/Provide grub2 boot images and metadata

Calls setup_install_boot_images because no different action required

kiwi.bootloader.config.grub2.BootLoaderConfigGrub2.setup_live_image_config

Create grub2 config file to boot a live media ISO image

Parameters

- `mbrid` () – mbrid file name on boot device
- `hypervisor` () – hypervisor name
- `kernel` () – kernel name
- `initrd` () – initrd name

kiwi.bootloader.config.grub2.BootLoaderConfigGrub2.setup_sysconfig_bootloader

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

Implementation in specialized bootloader class required

kiwi.bootloader.config.grub2.BootLoaderConfigGrub2.write

Write bootloader configuration

- writes grub.cfg template by KIWI if template system is used
- creates an embedded fat efi image for EFI ISO boot

kiwi.bootloader.config.grub2.BootLoaderConfigGrub2.write_meta_data

Write bootloader setup meta data files

- cmdline arguments initialization
- etc/default/grub setup file
- etc/default/zipl2grub.conf.in (s390 only)
- etc/sysconfig/bootloader

Parameters

- `root_device` () – root device node
- `write_device` () – overlay root write device node
- `boot_options` () – kernel options as string
- `iso_boot` () – indicate target is an ISO

`kiwi.bootloader.config.grub2.func_name`

helper method to return name of caller

14.4.4 kiwi.bootloader.config.systemd_boot Module

`kiwi.bootloader.config.systemd_boot.BootLoaderSystemdBoot`

Bases: `BootLoaderSpecBase`

kiwi.bootloader.config.systemd_boot.BootLoaderSystemdBoot.create_loader_image

Handle EFI images

For systemd boot EFI images are provided along with systemd. Thus the only action taken care of is the creation of the ESP path

Parameters

`target` () – target identifier, one of disk, live(iso) or install(iso)

kiwi.bootloader.config.systemd_boot.BootLoaderSystemdBoot.set_loader_entry

Setup/update loader entries

Parameters

`target` () – target identifier, one of disk, live(iso) or install(iso)

kiwi.bootloader.config.systemd_boot.BootLoaderSystemdBoot.setup_loader

Setup ESP for systemd-boot using bootctl

For disk images only, setup the ESP layout for systemd boot using bootctl. All data will be loaded from the ESP.

Parameters

`target` () – target identifier, one of disk, live(iso) or install(iso) Currently only the disk identifier is supported

14.4.5 kiwi.bootloader.config.zipl Module

`kiwi.bootloader.config.zipl.BootLoaderZipl`

Bases: `BootLoaderSpecBase`

kiwi.bootloader.config.zipl.BootLoaderZipl.create_loader_image

Nothing to be done here for zipl

Parameters

`target` () – unused

kiwi.bootloader.config.zipl.BootLoaderZipl.set_loader_entry

Setup/update loader entries of the form {boot_path}/loader/entries/{get_entry_name}

Parameters

`target` () – target identifier, one of disk, live(iso) or install(iso)

kiwi.bootloader.config.zipl.BootLoaderZipl.setup_loader

Setup temporary zipl config and install zipl for supported targets. Please note we are not touching the main zipl.conf file provided by the distributors

Parameters

`target` () – target identifier, one of disk, live(iso) or install(iso) Currently only the disk identifier is supported

14.4.6 Module Contents

`kiwi.bootloader.config.create_boot_loader_config`,

,

,

,

14.5 kiwi.bootloader.install Package

14.5.1 Submodules

14.5.2 `kiwi.bootloader.install.base` Module

`kiwi.bootloader.install.base.BootLoaderInstallBase`

Bases: `object`

Base class for bootloader installation on device

Parameters

- `root_dir` () – root directory path name
- `device_provider` () – instance of `DeviceProvider`
- `custom_args` () – custom arguments dictionary

kiwi.bootloader.install.base.BootLoaderInstallBase.install

Install bootloader on self.device

Implementation in specialized bootloader install class required

kiwi.bootloader.install.base.BootLoaderInstallBase.install_required

Check if bootloader needs to be installed

Implementation in specialized bootloader install class required

kiwi.bootloader.install.base.BootLoaderInstallBase.post_init

Post initialization method

Store custom arguments by default

Parameters

`custom_args` () – custom bootloader arguments

kiwi.bootloader.install.base.BootLoaderInstallBase.secure_boot_install

Run shim-install in self.device for secure boots

Implementation in specialized bootloader install class required

14.5.3 `kiwi.bootloader.install.grub2` Module

`kiwi.bootloader.install.grub2.BootLoaderInstallGrub2`

Bases: `BootLoaderInstallBase` (#kiwi.bootloader.install.base.BootLoaderInstallBase)

grub2 bootloader installation

kiwi.bootloader.install.grub2.BootLoaderInstallGrub2.install

Install bootloader on disk device

kiwi.bootloader.install.grub2.BootLoaderInstallGrub2.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

kiwi.bootloader.install.grub2.BootLoaderInstallGrub2.post_init

grub2 post initialization method

Parameters

`custom_args` () –

Contains custom grub2 bootloader arguments

```
{
    'target_removable': bool,
    'system_volumes': list_of_volumes,
    'system_root_volume': root volume name if required
    'firmware': FirmWare_instance,
    'efi_device': string,
    'boot_device': string,
    'root_device': string
}
```

kiwi.bootloader.install.grub2.BootLoaderInstallGrub2.secure_boot_install

Run shim-install in self.device for secure boots

Implementation in specialized bootloader install class required

kiwi.bootloader.install.grub2.BootLoaderInstallGrub2.set_disk_password

14.5.4 `kiwi.bootloader.install.systemd_boot` Module

`kiwi.bootloader.install.systemd_boot.BootLoaderInstallSystemdBoot`

Bases: `BootLoaderInstallBase` (#kiwi.bootloader.install.base.BootLoaderInstallBase)

systemd-boot bootloader installation

kiwi.bootloader.install.systemd_boot.BootLoaderInstallSystemdBoot.install_required

Check if systemd-boot needs to install boot code

systemd-boot supports EFI boot only and does not need to install boot code since it's expected that the firmware can read from the EFI partition

Returns

True or False

Return type

bool

kiwi.bootloader.install.systemd_boot.BootLoaderInstallSystemdBoot.post_init

systemd-boot post initialization method

Parameters

`custom_args` () – Contains custom systemd-boot bootloader arguments

kiwi.bootloader.install.systemd_boot.BootLoaderInstallSystemdBoot.secure_boot_install

Run shim installation for secure boot setup

For systemd-boot this is currently skipped since details for secure boot are not yet clear

14.5.5 `kiwi.bootloader.install.zipl` Module

`kiwi.bootloader.install.zipl.BootLoaderInstallZipl`

Bases: `BootLoaderInstallBase` (#kiwi.bootloader.install.base.BootLoaderInstallBase)

zipl bootloader installation

kiwi.bootloader.install.zipl.BootLoaderInstallZipl.install_required

Check if zipl needs to install boot code

zipl requires boot code installation, but it is done as part of the BLS implementation in bootloader/config/zipl.py Thus this method always returns: False

Returns

False

Return type

bool

kiwi.bootloader.install.zipl.BootLoaderInstallZipl.post_init

zipl post initialization method

Parameters

`custom_args` () – unused

kiwi.bootloader.install.zipl.BootLoaderInstallZipl.secure_boot_install

Run shim installation for secure boot setup

For zipl this is skipped since details for secure boot are not yet clear.

14.5.6 Module Contents

`kiwi.bootloader.install.BootLoaderInstall`

Bases: `object`

BootLoaderInstall Factory

Parameters

- `name` () – bootloader name
- `root_dir` () – root directory path name
- `device_provider` () – instance of `DeviceProvider`
- `custom_args` () – custom arguments dictionary

kiwi.bootloader.install.BootLoaderInstall.new

14.6 kiwi.bootloader Package

14.6.1 Module Contents

14.7 kiwi.bootloader.template Package

14.7.1 Submodules

14.7.2 `kiwi.bootloader.template.grub2` Module

`kiwi.bootloader.template.grub2.BootLoaderTemplateGrub2`

Bases: `object`

grub2 configuraton file templates

kiwi.bootloader.template.grub2.BootLoaderTemplateGrub2.get_install_template

Bootloader configuration template for install media

Parameters

- `failsafe` () – with failsafe true|false
- `has_graphics` () – supports graphics terminal
- `has_serial` () – supports serial terminal

Returns

instance of `Template`

Return type

Template

kiwi.bootloader.template.grub2.BootLoaderTemplateGrub2.get_iso_template

Bootloader configuration template for live ISO media

Parameters

- `failsafe` () – with failsafe true|false
- `has_graphics` () – supports graphics terminal
- `has_serial` () – supports serial terminal

Returns

instance of `Template`

Return type

Template

kiwi.bootloader.template.grub2.BootLoaderTemplateGrub2.get_multiboot_install_template

Bootloader configuration template for install media with hypervisor, e.g Xen dom0

Parameters

- `failsafe` () – with failsafe true|false
- `has_graphics` () – supports graphics terminal
- `has_serial` () – supports serial terminal

Returns

instance of `Template`

Return type

Template

kiwi.bootloader.template.grub2.BootLoaderTemplateGrub2.get_multiboot_iso_template

Bootloader configuration template for live ISO media with hypervisor, e.g Xen dom0

Parameters

- `failsafe` () – with failsafe true|false
- `has_graphics` () – supports graphics terminal
- `has_serial` () – supports serial terminal

Returns

instance of `Template`

Return type

Template

14.7.3 Module Contents

14.8 kiwi.boot Package

14.8.1 Module Contents

14.9 kiwi.builder Package

14.9.1 Submodules

14.9.2 `kiwi.builder.archive` Module

`kiwi.builder.archive.ArchiveBuilder`

Bases: `object`

Root archive image builder

Parameters

- `xml_state` () – Instance of `XMLState`
- `target_dir` () – target directory path name
- `root_dir` () – root directory path name
- `custom_args` () – Custom processing arguments defined as hash keys: * xz_options: string of XZ compression parameters

kiwi.builder.archive.ArchiveBuilder.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"
- image = "cpio"

Returns

result

Return type

instance of `Result`

14.9.3 `kiwi.builder.container` Module

`kiwi.builder.container.ContainerBuilder`

Bases: `object`

Container image builder

Parameters

- `xml_state` () – Instance of `XMLState`
- `target_dir` () – target directory path name
- `root_dir` () – root directory path name
- `custom_args` () – Custom processing arguments defined as hash keys: * xz_options: string of XZ compression parameters

kiwi.builder.container.ContainerBuilder.create

Builds a container image which is usually a data archive including container specific metadata.

Image types which triggers this builder are:

- image = "docker"
- image = "oci"
- image = "appx"
- image = "wsl"

Returns

result

Return type

instance of `Result`

14.9.4 `kiwi.builder.disk` Module

`kiwi.builder.disk.DiskBuilder`

Bases: `object`

Disk image builder

Parameters

- `xml_state` () – Instance of `XMLState`
- `target_dir` () – Target directory path name
- `root_dir` () – Root directory path name
- `custom_args` () – Custom processing arguments defined as hash keys: * signing_keys: list of package signing keys * xz_options: string of XZ compression parameters

kiwi.builder.disk.DiskBuilder.append_unpartitioned_space

Extends the raw disk if an unpartitioned area is specified

kiwi.builder.disk.DiskBuilder.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”

Returns

result

Return type

instance of `Result`

kiwi.builder.disk.DiskBuilder.create_disk

Build a bootable raw disk image

Raises

- `kiwi.exceptions.KiwiInstallMediaError` – if install media is required and image type is not oem
- `kiwi.exceptions.KiwiVolumeManagerSetupError` – root overlay at the same time volumes are defined is not supported

Returns

result

Return type

instance of `Result`

kiwi.builder.disk.DiskBuilder.create_disk_format

Create a bootable disk format from a previously created raw disk image

Parameters

`result_instance` () – instance of `Result`

Returns

updated result_instance

Return type

instance of `Result`

kiwi.builder.disk.DiskBuilder.create_install_media

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` () – instance of `Result`

Returns

updated result_instance with installation media

Return type

instance of `Result`

`kiwi.builder.disk.StorageMap`

Bases: `TypedDict`

kiwi.builder.disk.StorageMap.integrity_root

kiwi.builder.disk.StorageMap.luks_root

kiwi.builder.disk.StorageMap.raid_root

kiwi.builder.disk.StorageMap.system

kiwi.builder.disk.StorageMap.system_boot

kiwi.builder.disk.StorageMap.system_custom_parts

kiwi.builder.disk.StorageMap.system_efi

kiwi.builder.disk.StorageMap.system_spare

14.9.5 `kiwi.builder.filesystem` Module

`kiwi.builder.filesystem.FileSystemBuilder`

Bases: `object`

Filesystem image builder

Parameters

- `xml_state` () – Instance of `XMLState`
- `target_dir` () – target directory path name
- `root_dir` () – root directory path name
- `custom_args` () – Custom processing arguments defined as hash keys: * None

kiwi.builder.filesystem.FileSystemBuilder.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`

14.9.6 `kiwi.builder.install` Module

`kiwi.builder.install.InstallImageBuilder`

Bases: `object`

Installation image builder

Parameters

- `xml_state` () – instance of `XMLState`
- `root_dir` () – system image root directory
- `target_dir` () – target directory path name
- `boot_image_task` () – instance of `BootImage`
- `custom_args` () – Custom processing arguments defined as hash keys: * xz_options: string of XZ compression parameters

kiwi.builder.install.InstallImageBuilder.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”

kiwi.builder.install.InstallImageBuilder.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”

14.9.7 kiwi.builder.live Module

`kiwi.builder.live.LiveImageBuilder`

Bases: `object`

Live image builder

Parameters

- `xml_state` () – instance of `XMLState`
- `target_dir` () – target directory path name
- `root_dir` () – root directory path name
- `custom_args` () – Custom processing arguments

kiwi.builder.live.LiveImageBuilder.create

Build a bootable hybrid live ISO image

Image types which triggers this builder are:

- image = ”iso”

Raises

`kiwi.exceptions.KiwiLiveBootImageError` – if no kernel or hipervisor is found in boot image tree

Returns

result

Return type

instance of `Result`

14.9.8 `kiwi.builder.kis` Module

`kiwi.builder.kis.KisBuilder`

Bases: `object`

Filesystem based image builder.

Parameters

- `xml_state` () – instance of `XMLState`
- `target_dir` () – target directory path name
- `root_dir` () – system image root directory
- `custom_args` () – Custom processing arguments defined as hash keys: * signing_keys: list of package signing keys * xz_options: string of XZ compression parameters

kiwi.builder.kis.KisBuilder.create

Build a component image consisting out of a boot image(initrd) plus its appropriate kernel files and the root filesystem image with a checksum.

Image types which triggers this builder are:

- image = ”kis”
- image = ”pxe”

Raises

`kiwi.exceptions.KiwiKisBootImageError` – if no kernel or hipervisor is found in boot image tree

Returns

result

Return type

instance of `Result`

14.9.9 Module Contents

`kiwi.builder.ImageBuilder`

Bases: `object`

Image builder factory

kiwi.builder.ImageBuilder.new

14.10 kiwi.container Package

14.10.1 Submodules

14.10.2 kiwi.container.oci Module

`kiwi.container.oci.ContainerImageOCI`

Bases: `ContainerImageBase`

Create oci container from a root directory

Parameters

- `root_dir` () – root directory path name
- `custom_args` ()

Custom processing arguments defined as hash keys:
Example

```
{
    'container_name': 'name',
    'container_tag': '1.0',
    'additional_names': ['current', 'foobar'],
    'entry_command': ['/bin/bash', '-x'],
    'entry_subcommand': ['ls', '-l'],
    'maintainer': 'tux',
    'user': 'root',
    'workingdir': '/root',
    'expose_ports': ['80', '42'],
    'volumes': ['/var/log', '/tmp'],
    'environment': {'PATH': '/bin'},
    'labels': {'name': 'value'},
    'history': {
        'created_by': 'some explanation here',
        'comment': 'some comment here',
        'author': 'tux'
    }
}
```

kiwi.container.oci.ContainerImageOCI.create

Create compressed oci system container tar archive

Parameters

- `filename` () – archive file name
- `base_image` () – archive used as a base image
- `ensure_empty_tmpdirs` () – exclude system tmp directories
- `compress_archive` () – compress container archive

`kiwi.container.oci.OciConfig`

Bases: `TypedDict`

kiwi.container.oci.OciConfig.additional_names

kiwi.container.oci.OciConfig.container_name

kiwi.container.oci.OciConfig.container_tag

kiwi.container.oci.OciConfig.entry_command

kiwi.container.oci.OciConfig.entry_subcommand

kiwi.container.oci.OciConfig.environment

kiwi.container.oci.OciConfig.expose_ports

kiwi.container.oci.OciConfig.history

kiwi.container.oci.OciConfig.labels

kiwi.container.oci.OciConfig.maintainer

kiwi.container.oci.OciConfig.user

kiwi.container.oci.OciConfig.volumes

kiwi.container.oci.OciConfig.workingdir

14.10.3 Module Contents

`kiwi.container.ContainerImage`

Bases: `object`

Container Image factory

Parameters

- `name` () – container system name
- `root_dir` () – root directory path name
- `custom_args` () – custom arguments

kiwi.container.ContainerImage.new

14.11 kiwi.container.setup Package

14.11.1 Submodules

14.11.2 `kiwi.container.setup.base` Module

`kiwi.container.setup.base.ContainerSetupBase`

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

kiwi.container.setup.base.ContainerSetupBase.deactivate_bootloader_setup

Container bootloader setup

Tell the system there is no bootloader configuration it needs to care for. A container does not boot

kiwi.container.setup.base.ContainerSetupBase.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

kiwi.container.setup.base.ContainerSetupBase.deactivate_systemd_service

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` () – systemd service name

kiwi.container.setup.base.ContainerSetupBase.get_container_name

Container name

Returns

name

Return type

str

kiwi.container.setup.base.ContainerSetupBase.post_init

Post initialization method

Implementation in specialized container setup class

Parameters

`custom_args` () – unused

kiwi.container.setup.base.ContainerSetupBase.setup

Setup container metadata

Implementation in specialized bootloader class required

kiwi.container.setup.base.ContainerSetupBase.setup_root_console

Container console setup

/dev/console should be allowed to login by root

14.11.3 `kiwi.container.setup.docker` Module

`kiwi.container.setup.docker.ContainerSetupDocker`

Bases: `ContainerSetupOCI`

Docker container setup

14.11.4 Module Contents

`kiwi.container.setup.ContainerSetup`

Bases: `object`

container setup factory

kiwi.container.setup.ContainerSetup.new

14.12 kiwi.filesystem Package

14.12.1 Submodules

14.12.2 `kiwi.filesystem.base` Module

`kiwi.filesystem.base.FileSystemBase`

Bases: `object`

Implements base class for filesystem interface

Parameters

- `device_provider` () – 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` () – root directory path name
- `custom_args` () – custom filesystem arguments

kiwi.filesystem.base.FileSystemBase.create_on_device

Create filesystem on block device

Implement in specialized filesystem class for filesystems which requires a block device for creation, e.g ext4.

Parameters

- `label` () – label name
- `size` () – size value, can also be counted from the end via -X The value is interpreted in units of: unit
- `unit` () – unit name. Default unit is set to: defaults.UNIT.kb
- `uuid` () – UUID name

kiwi.filesystem.base.FileSystemBase.create_on_file

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` () – result file path name
- `label` () – label name
- `exclude` () – list of exclude dirs/files

kiwi.filesystem.base.FileSystemBase.create_verification_metadata

Write verification block at the end of the device

Parameters

`device_node` () – Target device node, if not specified the root device from this instance is used

kiwi.filesystem.base.FileSystemBase.create_verity_layer

Create veritysetup on device

Parameters

- `block` () – Number of blocks to use for veritysetup. If not specified the entire root device is used
- `filename` () – Target filename to use for VeritySetup. If not specified the filename or block special provided at object construction time is used

kiwi.filesystem.base.FileSystemBase.get_fstab

Consistency layer with regards to VolumeManager classes

Raises

kiwi.filesystem.base.FileSystemBase.get_mountpoint

Provides mount point directory

Effective use of the directory is guaranteed only after sync_data

Returns

directory path name

Return type

string

kiwi.filesystem.base.FileSystemBase.get_root_volume_name

Consistency layer with regards to VolumeManager classes

Raises

kiwi.filesystem.base.FileSystemBase.get_volumes

Consistency layer with regards to VolumeManager classes

Raises

kiwi.filesystem.base.FileSystemBase.mount

Mount the filesystem

kiwi.filesystem.base.FileSystemBase.mount_volumes

Consistency layer with regards to VolumeManager classes

Invokes mount

kiwi.filesystem.base.FileSystemBase.post_init

Post initialization method

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

Parameters

`custom_args` () –

custom arguments

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

kiwi.filesystem.base.FileSystemBase.set_property_readonly_root

Consistency layer with regards to VolumeManager classes

Raises

kiwi.filesystem.base.FileSystemBase.set_uuid

Create new random filesystem UUID

Implement in specialized filesystem class for filesystems which supports the concept of an UUID and allows to change it

kiwi.filesystem.base.FileSystemBase.sync_data

Copy data tree into filesystem

Parameters

`exclude` () – list of exclude dirs/files

Returns

The mount created for syncing data. It should be used to un-mount the filesystem again.

kiwi.filesystem.base.FileSystemBase.umount

Umounts the filesystem in case it is mounted, does nothing otherwise

kiwi.filesystem.base.FileSystemBase.umount_volumes

Consistency layer with regards to VolumeManager classes

Invokes umount

14.12.3 kiwi.filesystem.btrfs Module

kiwi.filesystem.btrfs.FileSystemBtrfs

Bases: FileSystemBase (#kiwi.filesystem.base.FileSystemBase)

Implements creation of btrfs filesystem

kiwi.filesystem.btrfs.FileSystemBtrfs.create_on_device

Create btrfs filesystem on block device

Parameters

- label () – label name
- size () – size value, can also be counted from the end via -X The value is interpreted in units of: unit
- unit () – unit name. Default unit is set to: defaults.UNIT.kb
- uuid () – UUID name

kiwi.filesystem.btrfs.FileSystemBtrfs.set_uuid

Create new random filesystem UUID

14.12.4 kiwi.filesystem.ext2 Module

kiwi.filesystem.ext2.FileSystemExt2

Bases: FileSystemBase (#kiwi.filesystem.base.FileSystemBase)

Implements creation of ext2 filesystem

kiwi.filesystem.ext2.FileSystemExt2.create_on_device

Create ext2 filesystem on block device

Parameters

- label () – label name
- size () – size value, can also be counted from the end via -X The value is interpreted in units of: unit

- `unit` () – unit name. Default unit is set to: defaults.UNIT.kb
- `uuid` () – UUID name

kiwi.filesystem.ext2.FileSystemExt2.set_uuid

Create new random filesystem UUID

14.12.5 `kiwi.filesystem.ext3` Module

`kiwi.filesystem.ext3.FileSystemExt3`

Bases: `FileSystemBase` (#kiwi.filesystem.base.FileSystemBase)

Implements creation of ext3 filesystem

kiwi.filesystem.ext3.FileSystemExt3.create_on_device

Create ext3 filesystem on block device

Parameters

- `label` () – label name
- `size` () – size value, can also be counted from the end via -X The value is interpreted in units of: unit
- `unit` () – unit name. Default unit is set to: defaults.UNIT.kb
- `uuid` () – UUID name

kiwi.filesystem.ext3.FileSystemExt3.set_uuid

Create new random filesystem UUID

14.12.6 `kiwi.filesystem.ext4` Module

`kiwi.filesystem.ext4.FileSystemExt4`

Bases: `FileSystemBase` (#kiwi.filesystem.base.FileSystemBase)

Implements creation of ext4 filesystem

kiwi.filesystem.ext4.FileSystemExt4.create_on_device

Create ext4 filesystem on block device

Parameters

- `label` () – label name
- `size` () – size value, can also be counted from the end via -X The value is interpreted in units of: unit
- `unit` () – unit name. Default unit is set to: defaults.UNIT.kb
- `uuid` () – UUID name

kiwi.filesystem.ext4.FileSystemExt4.set_uuid

Create new random filesystem UUID

14.12.7 `kiwi.filesystem.fat16` Module

`kiwi.filesystem.fat16.FileSystemFat16`

Bases: `FileSystemBase` (#kiwi.filesystem.base.FileSystemBase)

Implements creation of fat16 filesystem

kiwi.filesystem.fat16.FileSystemFat16.create_on_device

Create fat16 filesystem on block device

Parameters

- `label` () – label name
- `size` () – size value, can also be counted from the end via -X The value is interpreted in units of: unit
- `unit` () – unit name. Default unit is set to: defaults.UNIT.kb
- `uuid` () – Volume Label, there is no real UUID on fat

kiwi.filesystem.fat16.FileSystemFat16.set_uuid

Create new random filesystem UUID

14.12.8 `kiwi.filesystem.fat32` Module

`kiwi.filesystem.fat32.FileSystemFat32`

Bases: `FileSystemBase` (#kiwi.filesystem.base.FileSystemBase)

Implements creation of fat32 filesystem

kiwi.filesystem.fat32.FileSystemFat32.create_on_device

Create fat32 filesystem on block device

Parameters

- `label` () – label name
- `size` () – size value, can also be counted from the end via -X The value is interpreted in units of: unit
- `unit` () – unit name. Default unit is set to: defaults.UNIT.kb
- `uuid` () – Volume Label, there is no real UUID on fat

kiwi.filesystem.fat32.FileSystemFat32.set_uuid

Create new random filesystem UUID

14.12.9 `kiwi.filesystem.isofs` Module

`kiwi.filesystem.isofs.FileSystemIsoFs`

Bases: `FileSystemBase` (#kiwi.filesystem.base.FileSystemBase)

Implements creation of iso filesystem

kiwi.filesystem.isofs.FileSystemIsoFs.create_on_file

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` () – result file path name
- `label` () – unused
- `exclude` () – unused

14.12.10 `kiwi.filesystem.setup` Module

`kiwi.filesystem.setup.FileSystemSetup`

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` () – Instance of XMLState
- `root_dir` () – root directory path

kiwi.filesystem.setup.FileSystemSetup.get_size_mbytes

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` () – name

Returns

mbytes

Return type

int

14.12.11 `kiwi.filesystem.squashfs` Module

`kiwi.filesystem.squashfs.FileSystemSquashFs`

Bases: `FileSystemBase` (#kiwi.filesystem.base.FileSystemBase)

Implements creation of squashfs filesystem

kiwi.filesystem.squashfs.FileSystemSquashFs.create_on_file

Create squashfs filesystem from data tree

There is no label which could be set for squashfs thus this parameter is not used

Parameters

- `filename` () – result file path name
- `label` () – unused
- `exclude` () – list of exclude dirs/files

14.12.12 kiwi.filesystem.xfs Module

kiwi.filesystem.xfs.FileSystemXfs

Bases: FileSystemBase (#kiwi.filesystem.base.FileSystemBase)

Implements creation of xfs filesystem

kiwi.filesystem.xfs.FileSystemXfs.create_on_device

Create xfs filesystem on block device

Parameters

- label () – label name
- size () – size value, can also be counted from the end via -X The value is interpreted in units of: unit
- unit () – unit name. Default unit is set to: defaults.UNIT.kb
- uuid () – UUID name

kiwi.filesystem.xfs.FileSystemXfs.set_uuid

Create new random filesystem UUID

14.12.13 Module Contents

kiwi.filesystem.FileSystem

Bases: object

FileSystem factory

Parameters

- name () – filesystem name
- device_provider () – Instance of DeviceProvider
- root_dir () – root directory path name
- custom_args () – dict of custom filesystem arguments

kiwi.filesystem.FileSystem.new

14.13 kiwi.iso_tools Package

14.13.1 Submodules

14.13.2 kiwi.iso_tools.base Module

kiwi.iso_tools.base.IsoToolsBase

Bases: object

Base Class for Parameter API for iso creation tools

kiwi.iso_tools.base.IsoToolsBase.add_efi_loader_parameters

Add ISO creation parameters to embed the EFI loader
Implementation in specialized tool class

kiwi.iso_tools.base.IsoToolsBase.create_iso

Create iso file
Implementation in specialized tool class

Parameters

- filename () – unused
- hidden_files () – unused

kiwi.iso_tools.base.IsoToolsBase.get_tool_name

Return caller name for iso creation tool
Implementation in specialized tool class

Returns

tool name

Return type

str

kiwi.iso_tools.base.IsoToolsBase.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

kiwi.iso_tools.base.IsoToolsBase.init_iso_creation_parameters

Create a set of standard parameters for the main loader
Implementation in specialized tool class

Parameters

`custom_args` () – unused

kiwi.iso_tools.base.IsoToolsBase.list_iso

List contents of an ISO image

Parameters

`isofile` () – unused

kiwi.iso_tools.base.IsoToolsBase.setup_media_loader_directory

14.13.3 `kiwi.iso_tools.xorriso` Module

`kiwi.iso_tools.xorriso.IsoToolsXorrIso`

Bases: `IsoToolsBase` (#kiwi.iso_tools.base.IsoToolsBase)

xorriso wrapper class

Implementation of Parameter API for iso creation tools using the libburnia project. Addressed here is the tool xorriso

kiwi.iso_tools.xorriso.IsoToolsXorrIso.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

kiwi.iso_tools.xorriso.IsoToolsXorrIso.create_iso

Creates the iso file with the given filename using xorriso

Parameters

- `filename` () – output filename
- `hidden_files` () – list of hidden files

kiwi.iso_tools.xorriso.IsoToolsXorrIso.get_tool_name

Lookup xorriso in search path

Raises

`kiwi.exceptions.KiwiIsoToolError` – if xorriso tool is not found

Returns

xorriso tool path

Return type

str

kiwi.iso_tools.xorriso.IsoToolsXorrIso.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

kiwi.iso_tools.xorriso.IsoToolsXorrIso.init_iso_creation_parameters

Create a set of standard parameters

Parameters

`custom_args` () – custom ISO meta data

14.13.4 `kiwi.iso_tools.iso` Module

`kiwi.iso_tools.iso.Iso`

Bases: `object`

Implements helper methods around the creation of ISO filesystems

kiwi.iso_tools.iso.Iso.set_media_tag

Include checksum tag in the ISO so it can be verified with the mediacheck program.

Parameters

`isofile` () – path to the ISO file

14.13.5 Module Contents

`kiwi.iso_tools.IsoTools`

Bases: `object`

IsoTools factory

kiwi.iso_tools.IsoTools.new

14.14 kiwi.package_manager Package

14.14.1 Submodules

14.14.2 `kiwi.package_manager.base` Module

`kiwi.package_manager.base.PackageManagerBase`

Bases: `object`

Implements base class for Package Management

Parameters

- `repository` () – instance of `Repository`
- `root_dir` () – root directory path name
- `package_requests` () – list of packages to install or delete
- `collection_requests` () – list of collections to install
- `product_requests` () – list of products to install

kiwi.package_manager.base.PackageManagerBase.clean_leftovers

Cleans package manager related data not needed in the resulting image such as custom macros

Implementation in specialized package manager class

kiwi.package_manager.base.PackageManagerBase.cleanup_requests

Cleanup request queues

kiwi.package_manager.base.PackageManagerBase.database_consistent

kiwi.package_manager.base.PackageManagerBase.dump_reload_package_database

kiwi.package_manager.base.PackageManagerBase.get_error_details

Provide further error details

In case the package manager call failed this method will return package manager specific error information if there is any

Returns

further error data as str or empty str

Return type

str

kiwi.package_manager.base.PackageManagerBase.has_failed

Evaluate given result return code

Any returncode != 0 is considered an error unless overwritten in specialized package manager class

Parameters

`returncode` () – return code number

Returns

True|False

Return type

boolean

kiwi.package_manager.base.PackageManagerBase.match_package_deleted

Match expression to indicate a package has been deleted

Implementation in specialized package manager class

Parameters

- `package_name` () – unused
- `package_manager_output` () – unused

Returns

True|False

Return type

bool

kiwi.package_manager.base.PackageManagerBase.match_package_installed

Match expression to indicate a package has been installed

Implementation in specialized package manager class

Parameters

- `package_name` () – unused
- `package_manager_output` () – unused

Returns

True|False

Return type

bool

kiwi.package_manager.base.PackageManagerBase.post_init

Post initialization method

Implementation in specialized package manager class

Parameters

`custom_args` () – unused

kiwi.package_manager.base.PackageManagerBase.post_process_delete_requests

Process extra code required after deleting packages

Implementation in specialized package manager class

kiwi.package_manager.base.PackageManagerBase.post_process_install_requests_bootstrap

Process extra code required after bootstrapping

Implementation in specialized package manager class

kiwi.package_manager.base.PackageManagerBase.process_delete_requests

Process package delete requests (chroot)

Implementation in specialized package manager class

Parameters

`force` () – unused

kiwi.package_manager.base.PackageManagerBase.process_install_requests

Process package install requests for image phase (chroot)

Implementation in specialized package manager class

kiwi.package_manager.base.PackageManagerBase.process_install_requests_bootstrap

Process package install requests for bootstrap phase (no chroot)

Implementation in specialized package manager class

kiwi.package_manager.base.PackageManagerBase.process_only_required

Setup package processing only for required packages

Implementation in specialized package manager class

kiwi.package_manager.base.PackageManagerBase.process_plus_recommended

Setup package processing to also include recommended dependencies

Implementation in specialized package manager class

kiwi.package_manager.base.PackageManagerBase.request_collection

Queue a package collection

Implementation in specialized package manager class

Parameters

`name` () – unused

kiwi.package_manager.base.PackageManagerBase.request_package

Queue a package request

Implementation in specialized package manager class

Parameters

`name` () – unused

kiwi.package_manager.base.PackageManagerBase.request_package_exclusion

Queue a package exclusion(skip) request

Implementation in specialized package manager class

Parameters

`name` () – unused

kiwi.package_manager.base.PackageManagerBase.request_package_lock

kiwi.package_manager.base.PackageManagerBase.request_product

Queue a product request

Implementation in specialized package manager class

Parameters

`name` () – unused

kiwi.package_manager.base.PackageManagerBase.setup_repository_modules

Setup repository modules and streams

Implementation in specialized package manager class

Parameters

`collection_modules` () – unused

kiwi.package_manager.base.PackageManagerBase.update

Process package update requests (chroot)

Implementation in specialized package manager class

14.14.3 `kiwi.package_manager.dnf4` Module

`kiwi.package_manager.dnf4.PackageManagerDnf4`

Bases: `PackageManagerBase` (#kiwi.package_manager.base.PackageManagerBase)

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

Parameters

- `dnf_args` () – dnf arguments from repository runtime configuration
- `command_env` () – dnf command environment from repository runtime configuration

kiwi.package_manager.dnf4.PackageManagerDnf4.clean_leftovers

Cleans package manager related data not needed in the resulting image such as custom macros

kiwi.package_manager.dnf4.PackageManagerDnf4.match_package_deleted

Match expression to indicate a package has been deleted

Parameters

- `package_name` () – package_name
- `package_manager_output` () – dnf status line

Returns

True|False

Return type

bool

kiwi.package_manager.dnf4.PackageManagerDnf4.match_package_installed

Match expression to indicate a package has been installed

This match for the package to be installed in the output of the dnf command is not 100% accurate. There might be false positives due to sub package names starting with the same base package name

Parameters

- `package_name` () – package_name
- `package_manager_output` () – dnf status line

Returns

True|False

Return type

bool

kiwi.package_manager.dnf4.PackageManagerDnf4.post_init

Post initialization method

Parameters

`custom_args` () – custom dnf arguments

kiwi.package_manager.dnf4.PackageManagerDnf4.post_process_install_requests_bootstrap

Move the rpm database to the place as it is expected by the rpm package installed during bootstrap phase

Parameters

- `root_bind` () – unused
- `delta_root` () – unused

kiwi.package_manager.dnf4.PackageManagerDnf4.process_delete_requests

Process package delete requests (chroot)

Parameters

`force` () – force deletion: true|false

Raises

`kiwi.exceptions.KiwiRequestError` – if none of the packages to delete is installed.

Returns

process results in command type

Return type

namedtuple

kiwi.package_manager.dnf4.PackageManagerDnf4.process_install_requests

Process package install requests for image phase (chroot)

Returns

process results in command type

Return type

namedtuple

kiwi.package_manager.dnf4.PackageManagerDnf4.process_install_requests_bootstrap

Process package install requests for bootstrap phase (no chroot)

Parameters

- `root_bind` () – unused
- `bootstrap_package` () – unused

Returns

process results in command type

Return type

namedtuple

kiwi.package_manager.dnf4.PackageManagerDnf4.process_only_required

Setup package processing only for required packages

kiwi.package_manager.dnf4.PackageManagerDnf4.process_plus_recommended

Setup package processing to also include recommended dependencies.

kiwi.package_manager.dnf4.PackageManagerDnf4.request_collection

Queue a collection request

Parameters

`name` () – dnf group ID name

kiwi.package_manager.dnf4.PackageManagerDnf4.request_package

Queue a package request

Parameters

`name` () – package name

kiwi.package_manager.dnf4.PackageManagerDnf4.request_package_exclusion

Queue a package exclusion(skip) request

Parameters

`name` () – package name

kiwi.package_manager.dnf4.PackageManagerDnf4.request_product

Queue a product request

There is no product definition in the fedora repo data

Parameters

`name` () – unused

kiwi.package_manager.dnf4.PackageManagerDnf4.setup_repository_modules

Setup repository modules and streams

Parameters

`collection_modules` () –

Expect dict of the form:

```
{
    'enable': [
        "module:stream", "module"
    ],
    'disable': [
        "module"
    ]
}
```

kiwi.package_manager.dnf4.PackageManagerDnf4.update

Process package update requests (chroot)

Returns

process results in command type

Return type

namedtuple

14.14.4 `kiwi.package_manager.zypper` Module

`kiwi.package_manager.zypper.PackageManagerZypper`

Bases: `PackageManagerBase` (#kiwi.package_manager.base.PackageManagerBase)

Implements Installation/Deletion of packages/collections with zypper

Parameters

- `zypper_args` () – zypper arguments from repository runtime configuration
- `command_env` () – zypper command environment from repository runtime configuration

kiwi.package_manager.zypper.PackageManagerZypper.clean_leftovers

Cleans package manager related data not needed in the resulting image such as custom macros

kiwi.package_manager.zypper.PackageManagerZypper.has_failed

Evaluate given result return code

In zypper any return code == 0 or >= 100 is considered success. Any return code different from 0 and < 100 is treated as an error we care for. Return codes >= 100 indicates an issue like 'new kernel needs reboot of the system' or similar which we don't care in the scope of image building

Parameters

`returncode` () – return code number

Returns

True|False

Return type

boolean

kiwi.package_manager.zypper.PackageManagerZypper.match_package_deleted

Match expression to indicate a package has been deleted

Parameters

- `package_name` () – package_name
- `package_manager_output` () – zypper status line

Returns

True|False

Return type

bool

kiwi.package_manager.zypper.PackageManagerZypper.match_package_installed

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_name` () – package_name
- `package_manager_output` () – zypper status line

Returns

True|False

Return type

bool

kiwi.package_manager.zypper.PackageManagerZypper.post_init

Post initialization method

Store custom zypper arguments

Parameters

`custom_args` () – custom zypper arguments

kiwi.package_manager.zypper.PackageManagerZypper.post_process_install_requests_bootstrap

Move the rpm database to the place as it is expected by the rpm package installed during bootstrap phase

Parameters

- `root_bind` () – unused
- `delta_root` () – unused

kiwi.package_manager.zypper.PackageManagerZypper.process_delete_requests

Process package delete requests (chroot)

Parameters

`force` () – force deletion: true|false

Raises

`kiwi.exceptions.KiwiRequestError` – if none of the packages to delete is installed

Returns

process results in command type

Return type

namedtuple

kiwi.package_manager.zypper.PackageManagerZypper.process_install_requests

Process package install requests for image phase (chroot)

Returns

process results in command type

Return type

namedtuple

kiwi.package_manager.zypper.PackageManagerZypper.process_install_requests_bootstrap

Process package install requests for bootstrap phase (no chroot)

Parameters

- `root_bind` () – unused
- `bootstrap_package` () – unused

Returns

process results in command type

Return type

namedtuple

kiwi.package_manager.zypper.PackageManagerZypper.process_only_required

Setup package processing only for required packages

kiwi.package_manager.zypper.PackageManagerZypper.process_plus_recommended

Setup package processing to also include recommended dependencies.

kiwi.package_manager.zypper.PackageManagerZypper.request_collection

Queue a collection request

Parameters

`name` () – zypper pattern name

kiwi.package_manager.zypper.PackageManagerZypper.request_package

Queue a package request

Parameters

`name` () – package name

kiwi.package_manager.zypper.PackageManagerZypper.request_package_exclusion

Queue a package exclusion(skip) request

Parameters

`name` () – package name

kiwi.package_manager.zypper.PackageManagerZypper.request_product

Queue a product request

Parameters

`name` () – zypper product name

kiwi.package_manager.zypper.PackageManagerZypper.setup_repository_modules

Repository modules not supported for zypper. The method does nothing in this scope

Parameters

`collection_modules` () – unused

kiwi.package_manager.zypper.PackageManagerZypper.update

Process package update requests (chroot)

Returns

process results in command type

Return type

namedtuple

14.14.5 Module Contents

`kiwi.package_manager.PackageManager`

Bases: `object`

Package manager factory

Parameters

- `repository` () – instance of `Repository`
- `package_manager` () – package manager name
- `custom_args` () – custom package manager arguments list

Raises

`kiwi.exceptions.KiwiPackageManagerSetupError` – if the requested package manager type is not supported

Returns

package manager

Return type

PackageManagerBase subclass

kiwi.package_manager.PackageManager.new

14.15 kiwi.partitioner Package

14.15.1 Submodules

14.15.2 `kiwi.partitioner.base` Module

`kiwi.partitioner.base.PartitionerBase`

Bases: `object`

Base class for partitioners

kiwi.partitioner.base.PartitionerBase.create

Create partition

Implementation in specialized partitioner class

Parameters

- `name` () – unused
- `mbsize` () – unused

- `type_name` () – unused
- `flags` () – unused
- `partition_id` () – If provided, use this exact partition ID instead of auto-incrementing.

kiwi.partitioner.base.PartitionerBase.get_id

Current partition number

Zero indicates no partition has been created so far

Returns

partition number

Return type

int

kiwi.partitioner.base.PartitionerBase.post_init

Post initialization method

Implementation in specialized partitioner class

kiwi.partitioner.base.PartitionerBase.resize_table

Resize partition table

Parameters

`entries` () – unused

kiwi.partitioner.base.PartitionerBase.set_flag

Set partition flag

Implementation in specialized partitioner class

Parameters

- `partition_id` () – unused
- `flag_name` () – unused

kiwi.partitioner.base.PartitionerBase.set_hybrid_mbr

Turn partition table into hybrid table if supported

Implementation in specialized partitioner class

kiwi.partitioner.base.PartitionerBase.set_mbr

Turn partition table into MBR (msdos table)

Implementation in specialized partitioner class

kiwi.partitioner.base.PartitionerBase.set_start_sector

Set start sector of first partition as configured

Parameters

`start_sector` () – unused

Does nothing by default

kiwi.partitioner.base.PartitionerBase.set_uuid

Set partition UUID

Implementation in specialized partitioner class

Parameters

- `partition_id` () – unused
- `uuid` () – unused

14.15.3 `kiwi.partitioner.dasd` Module

`kiwi.partitioner.dasd.PartitionerDasd`

Bases: `PartitionerBase` (#kiwi.partitioner.base.PartitionerBase)

Implements DASD partition setup

kiwi.partitioner.dasd.PartitionerDasd.create

Create DASD partition

Parameters

- `name` () – partition name
- `mbsize` () – partition size
- `type_name` () – unused
- `flags` () – unused
- `partition_id` () – unused

kiwi.partitioner.dasd.PartitionerDasd.post_init

Post initialization method

Setup fdasd partition type/flag map

kiwi.partitioner.dasd.PartitionerDasd.resize_table

Resize partition table

Nothing to be done here for DASD devices

Parameters

`entries` () – unused

kiwi.partitioner.dasd.PartitionerDasd.set_uuid

Set partition UUID

Nothing to be done here for DASD devices

Parameters

- `partition_id` () – unused
- `uuid` () – unused

14.15.4 `kiwi.partitioner.gpt` Module

`kiwi.partitioner.gpt.PartitionerGpt`

Bases: `PartitionerBase` (#kiwi.partitioner.base.PartitionerBase)

Implements GPT partition setup

kiwi.partitioner.gpt.PartitionerGpt.create

Create GPT partition

Parameters

- `name` () – partition name
- `mbsize` () – partition size
- `type_name` () – partition type
- `flags` () – additional flags
- `partition_id` () – If provided, use this exact partition ID instead of auto-incrementing

kiwi.partitioner.gpt.PartitionerGpt.post_init

Post initialization method

Setup gdisk partition type/flag map

kiwi.partitioner.gpt.PartitionerGpt.resize_table

Resize partition table

Parameters

`entries` () – number of default entries

kiwi.partitioner.gpt.PartitionerGpt.set_flag

Set GPT partition flag

Parameters

- `partition_id` () – partition number
- `flag_name` () – name from flag map

kiwi.partitioner.gpt.PartitionerGpt.set_hybrid_mbr

Turn partition table into hybrid GPT/MBR table

kiwi.partitioner.gpt.PartitionerGpt.set_mbr

Turn partition table into MBR (msdos table)

kiwi.partitioner.gpt.PartitionerGpt.set_uuid

Set partition UUID (TypeCode)

Parameters

- `partition_id` () – partition number
- `uuid` () – UUID

14.15.5 `kiwi.partitioner.msdos` Module

`kiwi.partitioner.msdos.PartitionerMsDos`

Bases: `PartitionerBase` (#kiwi.partitioner.base.PartitionerBase)

Implement old style msdos partition setup

kiwi.partitioner.msdos.PartitionerMsDos.create

Create msdos partition

Parameters

- `name` () – partition name
- `mbsize` () – partition size

- `type_name` () – partition type
- `flags` () – additional flags
- `partition_id` () – If provided, use this exact partition ID instead of auto-incrementing.

kiwi.partitioner.msdos.PartitionerMsDos.post_init

Post initialization method

Setup sfdisk partition type/flag map

kiwi.partitioner.msdos.PartitionerMsDos.resize_table

Resize partition table

Nothing to be done here for msdos table

Parameters

`entries` () – unused

kiwi.partitioner.msdos.PartitionerMsDos.set_flag

Set msdos partition flag

Parameters

- `partition_id` () – partition number
- `flag_name` () – name from flag map

kiwi.partitioner.msdos.PartitionerMsDos.set_start_sector

Set start sector of first partition as configured. fdisk and friends are not able to work correctly if the start sector of the first partition is any different from 2048.

Parameters

`start_sector` () – sector size

kiwi.partitioner.msdos.PartitionerMsDos.set_uuid

Set partition UUID

Nothing to be done here for MSDOS devices

Parameters

- `partition_id` () – unused
- `uuid` () – unused

14.15.6 Module Contents

`kiwi.partitioner.Partitioner`

Bases: `object`

Partitioner factory

Parameters

- `table_type` () – Table type name
- `storage_provider` () – Instance of class based on DeviceProvider
- `start_sector` () – sector number
- `extended_layout` () – support extended layout for msdos table

kiwi.partitioner.Partitioner.new

14.16 kiwi.repository Package

14.16.1 Submodules

14.16.2 `kiwi.repository.base` Module

`kiwi.repository.base.RepositoryBase`

Bases: `object`

Implements base class for package manager repository handling

Attributes

Parameters

- `root_bind` () – instance of `RootBind`
- `root_dir` () – root directory path name
- `shared_location` () – shared directory between image root and build system root

kiwi.repository.base.RepositoryBase.add_repo

Add repository

Implementation in specialized repository class

Parameters

- `name` () – unused
- `uri` () – unused
- `repo_type` () – unused
- `prio` () – unused
- `dist` () – unused
- `components` () – unused
- `user` () – unused
- `secret` () – unused
- `credentials_file` () – unused
- `repo_gpgcheck` () – unused
- `pkg_gpgcheck` () – unused
- `sourcetype` () – unused
- `customization_script` () – unused
- `architectures` () – unused

kiwi.repository.base.RepositoryBase.cleanup

Cleanup method

Implementation in specialized repository class

kiwi.repository.base.RepositoryBase.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

kiwi.repository.base.RepositoryBase.delete_all_repos

Delete all repositories

Implementation in specialized repository class

kiwi.repository.base.RepositoryBase.delete_repo

Delete repository

Implementation in specialized repository class

Parameters

`name` () – unused

kiwi.repository.base.RepositoryBase.delete_repo_cache

Delete repository cache

Implementation in specialized repository class

Parameters

`name` () – unused

kiwi.repository.base.RepositoryBase.import_trusted_keys

Imports trusted keys into the image

Implementation in specialized repository class

Parameters

`signing_keys` () – list of the key files to import

kiwi.repository.base.RepositoryBase.post_init

Post initialization method

Implementation in specialized repository class

Parameters

`custom_args` () – unused

kiwi.repository.base.RepositoryBase.run_repo_customize

Run an optional customization script

Parameters

- `script_path` () – unused
- `repo_file` () – unused

kiwi.repository.base.RepositoryBase.runtime_config

Repository runtime configuration and environment

Implementation in specialized repository class

kiwi.repository.base.RepositoryBase.setup_package_database_configuration

Setup package database configuration

Implementation in specialized repository class

kiwi.repository.base.RepositoryBase.use_default_location

Call repository operations with default repository manager setup

Implementation in specialized repository class

14.16.3 `kiwi.repository.dnf4` Module

`kiwi.repository.dnf4.RepositoryDnf4`

Bases: `RepositoryBase` (#kiwi.repository.base.RepositoryBase)

Implements repository handling for dnf package manager

Parameters

- `shared_dnf_dir` () – shared directory between image root and build system root
- `runtime_dnf_config_file` () – dnf runtime config file name
- `command_env` () – customized os.environ for dnf
- `runtime_dnf_config` () – instance of `ConfigParser`

kiwi.repository.dnf4.RepositoryDnf4.add_repo

Add dnf repository

Parameters

- `name` () – repository base file name
- `uri` () – repository URI
- `repo_type` () – repostory type name
- `prio` () – dnf repostory priority
- `dist` () – unused

- `components` () – unused
- `user` () – unused
- `secret` () – unused
- `credentials_file` () – unused
- `repo_gpgcheck` () – enable repository signature validation
- `pkg_gpgcheck` () – enable package signature validation
- `sourcetype` () – source type, one of 'baseurl', 'metalink' or 'mirrorlist'
- `customization_script` () – custom script called after the repo file was created
- `architectures` () – unused

kiwi.repository.dnf4.RepositoryDnf4.cleanup

Delete intermediate dnf config file

kiwi.repository.dnf4.RepositoryDnf4.cleanup_unused_repos

Delete unused dnf repositories

Repository configurations which are not used for this build must be removed otherwise they are taken into account for the package installations

kiwi.repository.dnf4.RepositoryDnf4.delete_all_repos

Delete all dnf repositories

kiwi.repository.dnf4.RepositoryDnf4.delete_repo

Delete dnf repository

Parameters

`name` () – repository base file name

kiwi.repository.dnf4.RepositoryDnf4.delete_repo_cache

Delete dnf repository cache

The cache data for each repository is stored in a directory and additional files all starting with the repository name. The method glob deletes all files and directories matching the repository name followed by any characters to cleanup the cache information

Parameters

`name` () – repository name

kiwi.repository.dnf4.RepositoryDnf4.import_trusted_keys

Imports trusted keys into the image

Parameters

`signing_keys` () – list of the key files to import

kiwi.repository.dnf4.RepositoryDnf4.post_init

Post initialization method

Store custom dnf arguments and create runtime configuration and environment

Parameters

`custom_args` () – dnf arguments

kiwi.repository.dnf4.RepositoryDnf4.runtime_config

dnf runtime configuration and environment

Returns

dnf_args:list, command_env:dict

Return type

dict

kiwi.repository.dnf4.RepositoryDnf4.setup_package_database_configuration

Setup rpm macros for bootstrapping and image building

1. Create the rpm image macro which persists during the build
2. Create the rpm bootstrap macro to make sure for bootstrapping the rpm database location matches the host rpm database setup. This macro only persists during the bootstrap phase. If the image was already bootstrapped a compat link is created instead.

kiwi.repository.dnf4.RepositoryDnf4.use_default_location

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

14.16.4 `kiwi.repository.zypper` Module

`kiwi.repository.zypper.RepositoryZypper`

Bases: `RepositoryBase` (#kiwi.repository.base.RepositoryBase)

Implements repo handling for zypper package manager

Parameters

- `shared_zypper_dir` () – shared directory between image root and build system root
- `runtime_zypper_config_file` () – zypper runtime config file name
- `runtime_zypp_config_file` () – libzypp runtime config file name
- `zypper_args` () – zypper caller args plus additional custom args
- `command_env` () – customized os.environ for zypper
- `runtime_zypper_config` () – instance of `ConfigParser`

kiwi.repository.zypper.RepositoryZypper.add_repo

Add zypper repository

Parameters

- `name` () – repository name
- `uri` () – repository URI
- `repo_type` () – repostory type name
- `prio` () – zypper repostory priority
- `dist` () – unused
- `components` () – unused
- `user` () – credentials username
- `secret` () – credentials password
- `credentials_file` () – zypper credentials file
- `repo_gpgcheck` () – enable repository signature validation
- `pkg_gpgcheck` () – enable package signature validation
- `sourcetype` () – unused

- `customization_script` () – custom script called after the repo file was created
- `architectures` () – unused

kiwi.repository.zypper.RepositoryZypper.cleanup

Delete intermediate zypp and zypper config files

kiwi.repository.zypper.RepositoryZypper.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

kiwi.repository.zypper.RepositoryZypper.delete_all_repos

Delete all zypper repositories

kiwi.repository.zypper.RepositoryZypper.delete_repo

Delete zypper repository

Parameters

`name` () – repository name

kiwi.repository.zypper.RepositoryZypper.delete_repo_cache

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` () – repository name

kiwi.repository.zypper.RepositoryZypper.import_trusted_keys

Imports trusted keys into the image

Parameters

`signing_keys` () – list of the key files to import

kiwi.repository.zypper.RepositoryZypper.post_init

Post initialization method

Store custom zypper arguments and create runtime configuration and environment

Parameters

`custom_args` () – zypper arguments

kiwi.repository.zypper.RepositoryZypper.runtime_config

zypper runtime configuration and environment

kiwi.repository.zypper.RepositoryZypper.setup_package_database_configuration

Setup rpm macros for bootstrapping and image building

1. Create the rpm image macro which persists during the build
2. Create the rpm bootstrap macro to make sure for bootstrapping the rpm database location matches the host rpm database setup. This macro only persists during the bootstrap phase. If the image was already bootstrapped a compat link is created instead.
3. Create zypper compat link

kiwi.repository.zypper.RepositoryZypper.use_default_location

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

14.16.5 Module Contents

`kiwi.repository.Repository`

Bases: `object`

Repository factory

Parameters

- `root_bind` () – instance of RootBind
- `package_manager` () – package manager name
- `custom_args` () – list of custom package manager arguments to setup the repository

Raises

`kiwi.exceptions.KiwiRepositorySetupError` – if package_manager is not supported

kiwi.repository.Repository.new

14.17 kiwi.repository.template Package

14.17.1 Submodules

14.17.2 kiwi.repository.template.apt Module

kiwi.repository.template.apt.PackageManagerTemplateAptGet

Bases: object

apt-get configuration file template

kiwi.repository.template.apt.PackageManagerTemplateAptGet.get_host_template

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

Return type

Template

kiwi.repository.template.apt.PackageManagerTemplateAptGet.get_image_template

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

Return type

Template

14.17.3 Module Contents

14.18 kiwi.solver.repository Package

14.18.1 Submodules

14.18.2 kiwi.solver.repository.base Module

kiwi.solver.repository.base.SolverRepositoryBase

Bases: object

Base class interface for SAT solvable creation.

Parameters

- `uri` () – Instance of `Uri`
- `user` () – User name for uri authentication
- `secret` () – Secret token for uri authentication

kiwi.solver.repository.base.SolverRepositoryBase.create_repository_solvable

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` () – path name

Returns

file path to solvable

Return type

str

kiwi.solver.repository.base.SolverRepositoryBase.download_from_repository

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` () – source file in the repo
- `target` () – file path

Raises

`kiwi.exceptions.KiwiUriOpenError` – if the download fails

kiwi.solver.repository.base.SolverRepositoryBase.get_repo_type

kiwi.solver.repository.base.SolverRepositoryBase.is_uptodate

Check if repository metadata is up to date

Returns

True or False

Return type

bool

kiwi.solver.repository.base.SolverRepositoryBase.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

`kiwi.solver.repository.rpm_md.SolverRepositoryRpmMd`

Bases: `SolverRepositoryBase` (#kiwi.solver.repository.base.SolverRepositoryBase)

Class for SAT solvable creation for rpm-md type repositories.

kiwi.solver.repository.rpm_md.SolverRepositoryRpmMd.timestamp

Get timestamp from the first primary metadata

Returns

time value as text

Return type

str

`kiwi.solver.repository.rpm_dir.SolverRepositoryRpmDir`

Bases: `SolverRepositoryBase` (#kiwi.solver.repository.base.SolverRepositoryBase)

Class for SAT solvable creation for rpm_dir type repositories.

`kiwi.solver.repository.suse.SolverRepositorySUSE`

Bases: `SolverRepositoryBase` (#kiwi.solver.repository.base.SolverRepositoryBase)

Class for SAT solvable creation for SUSE type repositories.

14.18.3 Module Contents

`kiwi.solver.repository.SolverRepository`

Bases: `object`

Repository factory for creation of SAT solvables

- **param object uri**

  Instance of `Uri`

kiwi.solver.repository.SolverRepository.new

14.19 kiwi.solver Package

14.19.1 Submodules

14.19.2 `kiwi.solver.sat` Module

`kiwi.solver.sat.Sat`

Bases: `object`

Sat Solver class to run package solver operations

The class uses SUSE's libsolv sat plugin

kiwi.solver.sat.Sat.add_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` () – Instance of `SolverRepository`

kiwi.solver.sat.Sat.set_dist_type

kiwi.solver.sat.Sat.solve

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 of strings
- `skip_missing` () – skip job if not found
- `ignore_recommended` () – do not include recommended packages

Raises

`kiwi.exceptions.KiwiSatSolverJobProblems` – if solver reports solving problems

Returns

Transaction result information

Return type

dict

14.19.3 Module Contents

14.20 kiwi.storage Package

14.20.1 Submodules

14.20.2 `kiwi.storage.device_provider` Module

`kiwi.storage.device_provider.DeviceProvider`

Bases: `object`

Base class for any class providing storage devices

kiwi.storage.device_provider.DeviceProvider.get_byte_size

Size of device in bytes

Parameters

`device` () – node name

Returns

byte value from blockdev

Return type

int

kiwi.storage.device_provider.DeviceProvider.get_device

Representation of device nodes

Could provide one ore more devices representing the storage Implementation in specialized device provider class

kiwi.storage.device_provider.DeviceProvider.get_uuid

UUID of device

Parameters

`device` () – node name

Returns

UUID from blkid

Return type

str

kiwi.storage.device_provider.DeviceProvider.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

14.20.3 `kiwi.storage.disk` Module

`kiwi.storage.disk.Disk`

Bases: `DeviceProvider` (#kiwi.storage.device_provider.DeviceProvider)

Implements storage disk and partition table setup

kiwi.storage.disk.Disk.activate_boot_partition

Activate boot partition

Note: not all Partitioner instances supports this

kiwi.storage.disk.Disk.create_boot_partition

Create boot partition

Populates kiwi_BootPart(id) and optional kiwi_BootPartClone(id)

Parameters

- `mbsize` () – partition size string
- `clone` () – create [clone] cop(y/ies) of the boot partition
- `partition_id` () – If provided, use this exact partition ID instead of auto-incrementing. When cloned, the clone ID is calculated from the given partition_id

kiwi.storage.disk.Disk.create_custom_partitions

Create partitions from custom data set

```
table_entries = {
    map_name: ptable_entry_type
}
```

Parameters

`table` () – partition table spec

kiwi.storage.disk.Disk.create_efi_csm_partition

Create EFI bios grub partition

Populates kiwi_BiosGrub(id)

Parameters

- `mbsize` () – partition size string
- `partition_id` () – If provided, use this exact partition ID instead of auto-incrementing.

kiwi.storage.disk.Disk.create_efi_partition

Create EFI partition

Populates kiwi_EfiPart(id)

Parameters

- `mbsize` () – partition size string
- `partition_id` () – If provided, use this exact partition ID instead of auto-incrementing.

kiwi.storage.disk.Disk.create_hybrid_mbr

Turn partition table into a hybrid GPT/MBR table

Note: only GPT tables supports this

kiwi.storage.disk.Disk.create_mbr

Turn partition table into MBR (msdos table)

Note: only GPT tables supports this

kiwi.storage.disk.Disk.create_prep_partition

Create prep partition

Populates kiwi_PrepPart(id)

Parameters

- `mbsize` () – partition size string
- `partition_id` () – If provided, use this exact partition ID instead of auto-incrementing.

kiwi.storage.disk.Disk.create_root_lvm_partition

Create root partition for use with LVM

Populates kiwi_RootPart(id)

Parameters

- `mbsize` () – partition size string
- `clone` () – create [clone] cop(y/ies) of the lvm roo partition
- `partition_id` () – If provided, use this exact partition ID instead of auto-incrementing. When cloned, the clone ID is calculated from the given partition_id

kiwi.storage.disk.Disk.create_root_partition

Create root partition

Populates kiwi_RootPart(id) and kiwi_BootPart(id) if no extra boot partition is requested

Parameters

- `mbsize` () – partition size string
- `clone` () – create [clone] cop(y/ies) of the root partition
- `partition_id` () – If provided, use this exact partition ID instead of auto-incrementing. When cloned, the clone ID is calculated from the given partition_id

kiwi.storage.disk.Disk.create_root_raid_partition

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` () – partition size string
- `clone` () – create [clone] cop(y/ies) of the raid root partition
- `partition_id` () – If provided, use this exact partition ID instead of auto-incrementing. When cloned, the clone ID is calculated from the given partition_id

kiwi.storage.disk.Disk.create_root_readonly_partition

Create root readonly partition for use with overlayfs

Populates kiwi_ReadOnlyPart(id), the partition is meant to contain a squashfs readonly filesystem. The partition size should be the size of the squashfs filesystem in order to avoid wasting disk space

Parameters

- `mbsize` () – partition size string
- `clone` () – create [clone] cop(y/ies) of the ro root partition
- `partition_id` () – If provided, use this exact partition ID instead of auto-incrementing. When cloned, the clone ID is calculated from the given partition_id

kiwi.storage.disk.Disk.create_spare_partition

Create spare partition for custom use

Populates kiwi_SparePart(id)

Parameters

- `mbsize` () – partition size string
- `partition_id` () – If provided, use this exact partition ID instead of auto-incrementing.

kiwi.storage.disk.Disk.create_swap_partition

Create swap partition

Populates kiwi_SwapPart(id)

Parameters

- `mbsize` () – partition size string
- `partition_id` () – If provided, use this exact partition ID instead of auto-incrementing.

kiwi.storage.disk.Disk.gUID

Unified partition UUIDs according to systemd

kiwi.storage.disk.Disk.get_device

Names of partition devices

Note that the mapping requires an explicit map() call

Returns

instances of MappedDevice

Return type

dict

kiwi.storage.disk.Disk.get_discoverable_partition_ids

Ask systemd for a list of standardized GUIDs for the current architecture and return them in a dictionary. If there is no such information available an empty dictionary is returned

Returns

key:value dict from systemd-id128

Return type

dict

kiwi.storage.disk.Disk.get_public_partition_id_map

Populated partition name to number map

kiwi.storage.disk.Disk.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

kiwi.storage.disk.Disk.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

kiwi.storage.disk.Disk.protected_map_ids

list of protected map ids. If used in a custom partitions setup this will lead to a raise conditition in order to avoid conflicts with the existing partition layout and its customizaton capabilities

kiwi.storage.disk.Disk.set_start_sector

Set start sector

Note: only effective on DOS tables

kiwi.storage.disk.Disk.storage_provider

the underlaying device provider

kiwi.storage.disk.Disk.wipe

Zap (destroy) any GPT and MBR data structures if present For DASD disks create a new VTOC table

`kiwi.storage.disk.ptable_entry_type`

Bases: `NamedTuple`

kiwi.storage.disk.ptable_entry_type.clone

Alias for field number 1

kiwi.storage.disk.ptable_entry_type.filesystem

Alias for field number 6

kiwi.storage.disk.ptable_entry_type.label

Alias for field number 7

kiwi.storage.disk.ptable_entry_type.mbsize

Alias for field number 0

kiwi.storage.disk.ptable_entry_type.mountpoint

Alias for field number 5

kiwi.storage.disk.ptable_entry_type.partition_id

Alias for field number 4

kiwi.storage.disk.ptable_entry_type.partition_name

Alias for field number 2

kiwi.storage.disk.ptable_entry_type.partition_type

Alias for field number 3

14.20.4 `kiwi.storage.loop_device` Module

`kiwi.storage.loop_device.LoopDevice`

Bases: `DeviceProvider` (#kiwi.storage.device_provider.DeviceProvider)

Create and manage loop device file for block operations

Parameters

- `filename` () – loop file name to create
- `filesize_mbytes` () – size of the loop file
- `blocksize_bytes` () – blocksize used in loop driver

kiwi.storage.loop_device.LoopDevice.create

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` () – overwrite existing file to loop

kiwi.storage.loop_device.LoopDevice.get_device

Device node name

Returns

device node name

Return type

str

kiwi.storage.loop_device.LoopDevice.is_loop

Always True

Returns

True

Return type

bool

14.20.5 kiwi.storage.luks_device Module

kiwi.storage.luks_device.LuksDevice

Bases: DeviceProvider (#kiwi.storage.device_provider.DeviceProvider)

Implements luks setup on a storage device

Parameters

storage_provider () – Instance of class based on DeviceProvider

kiwi.storage.luks_device.LuksDevice.create_crypto_luks

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 () – credentials
- osname () – distribution name to match distribution specific options for cryptsetup
- options () – further cryptsetup options
- keyfile () – file path name file path name which contains an alternative key to unlock the luks device
- root_dir () – root dir path

kiwi.storage.luks_device.LuksDevice.create_crypttab

Create crypttab, setting the UUID of the storage device

Parameters

`filename` () – file path name

kiwi.storage.luks_device.LuksDevice.create_random_keyfile

Create keyfile with random data

Parameters

`filename` () – file path name

kiwi.storage.luks_device.LuksDevice.get_device

Instance of MappedDevice providing the luks device

Returns

mapped luks device

Return type

`kiwi.storage.mapped_device.MappedDevice`

kiwi.storage.luks_device.LuksDevice.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.luks_device.LuksDevice.storage_provider

the underlaying device provider

14.20.6 `kiwi.storage.mapped_device` Module

`kiwi.storage.mapped_device.MappedDevice`

Bases: `DeviceProvider` (#kiwi.storage.device_provider.DeviceProvider)

Hold a reference on a single device

Parameters

- `device_provider` () – Instance of class based on DeviceProvider
- `device` () – Device node name

kiwi.storage.mapped_device.MappedDevice.get_device

Mapped device node name

Returns

device node name

Return type

str

kiwi.storage.mapped_device.MappedDevice.is_loop

Check if storage provider is loop based

Return loop status from base storage provider

Returns

True or False

Return type

bool

14.20.7 `kiwi.storage.raid_device` Module

`kiwi.storage.raid_device.RaidDevice`

Bases: `DeviceProvider` (#kiwi.storage.device_provider.DeviceProvider)

Implement raid setup on a storage device

Parameters

`storage_provider` () – Instance of class based on DeviceProvider

kiwi.storage.raid_device.RaidDevice.create_degraded_raid

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` () – raid level name

kiwi.storage.raid_device.RaidDevice.create_raid_config

Create mdadm config file from mdadm request

Parameters

`filename` () – config file name

kiwi.storage.raid_device.RaidDevice.get_device

Instance of MappedDevice providing the raid device

Returns

mapped raid device

Return type

`kiwi.storage.mapped_device.MappedDevice`

kiwi.storage.raid_device.RaidDevice.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.RaidDevice.storage_provider

the underlaying device provider

14.20.8 `kiwi.storage.clone_device` Module

`kiwi.storage.clone_device.CloneDevice`

Bases: `DeviceProvider` (#kiwi.storage.device_provider.DeviceProvider)

Implements device cloning

kiwi.storage.clone_device.CloneDevice.clone

Clone source device to target device(s)

Parameters

`target_devices` () – List of target DeviceProvider instances

14.20.9 `kiwi.storage.setup` Module

`kiwi.storage.setup.DiskSetup`

Bases: `object`

Implements disk setup methods

Methods from this class provides information required before building a disk image

Parameters

- `xml_state` () – Instance of XMLState
- `root_dir` () – root directory path name

kiwi.storage.setup.DiskSetup.boot_partition_size

Size of the boot partition in mbytes

Returns

boot size mbytes

Return type

int

kiwi.storage.setup.DiskSetup.get_boot_label

Filesystem Label to use for the boot partition

Returns

label name

Return type

str

kiwi.storage.setup.DiskSetup.get_disksize_mbytes

Precalculate disk size requirements in mbytes

Parameters

- `root_clone` () – root partition gets cloned, N + 1 times the size is needed
- `boot_clone` () – boot partition gets cloned, N + 1 times the size is needed

Returns

disk size mbytes

Return type

int

kiwi.storage.setup.DiskSetup.get_efi_label

Filesystem Label to use for the EFI partition

Returns

label name

Return type

str

kiwi.storage.setup.DiskSetup.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

kiwi.storage.setup.DiskSetup.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

14.20.10 Module Contents

14.21 kiwi.storage.subformat Package

14.21.1 Submodules

14.21.2 `kiwi.storage.subformat.base` Module

`kiwi.storage.subformat.base.DiskFormatBase`

Bases: `object`

Base class to create disk formats from a raw disk image

Parameters

- `xml_state` () – Instance of XMLState
- `root_dir` () – root directory path name
- `arch` () – Defaults.get_platform_name
- `target_dir` () – target directory path name
- `custom_args` () – custom format options dictionary

kiwi.storage.subformat.base.DiskFormatBase.create_image_format

Create disk format

Implementation in specialized disk format class required

kiwi.storage.subformat.base.DiskFormatBase.get_qemu_option_list

Create list of qemu options from custom_args dict

Parameters

`custom_args` () – arguments

Returns

qemu option list

Return type

list

kiwi.storage.subformat.base.DiskFormatBase.get_target_file_path_for_format

Create target file path name for specified format

Parameters

`format_name` () – disk format name

Returns

file path name

Return type

str

kiwi.storage.subformat.base.DiskFormatBase.has_raw_disk

Check if the base raw disk image exists

Returns

True or False

Return type

bool

kiwi.storage.subformat.base.DiskFormatBase.post_init

Post initialization method

Implementation in specialized disk format class if required

Parameters

`custom_args` () – unused

kiwi.storage.subformat.base.DiskFormatBase.resize_raw_disk

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` () – size in bytes

Returns

True or False

Return type

bool

kiwi.storage.subformat.base.DiskFormatBase.store_to_result

Store result file of the format conversion into the provided result instance.

By default only the converted image file will be stored as compressed file. Subformats which creates additional metadata files or want to use other result flags needs to overwrite this method

Parameters

`result` () – Instance of Result

14.21.3 `kiwi.storage.subformat.gce` Module

`kiwi.storage.subformat.gce.DiskFormatGce`

Bases: `DiskFormatBase` (#kiwi.storage.subformat.base.DiskFormatBase)

Create GCE - Google Compute Engine image format

kiwi.storage.subformat.gce.DiskFormatGce.create_image_format

Create GCE disk format and manifest

kiwi.storage.subformat.gce.DiskFormatGce.get_target_file_path_for_format

Google requires the image name to follow their naming convetion. Therefore it's required to provide a suitable name by overriding the base class method

Parameters

`format_name` () – gce

Returns

file path name

Return type

str

kiwi.storage.subformat.gce.DiskFormatGce.post_init

GCE disk format post initialization method
Store disk tag from custom args

Parameters

`custom_args` () –
custom gce argument dictionary

```
{'--tag': 'billing_code'}
```

kiwi.storage.subformat.gce.DiskFormatGce.store_to_result

Store result file of the gce format conversion into the provided result instance. In this case compression is unwanted because the gce tarball is already created as a compressed archive

Parameters

`result` () – Instance of Result

14.21.4 `kiwi.storage.subformat.ova` Module

`kiwi.storage.subformat.ova.DiskFormatOva`

Bases: `DiskFormatBase` (#kiwi.storage.subformat.base.DiskFormatBase)

Create ova disk format, based on vmdk

kiwi.storage.subformat.ova.DiskFormatOva.create_image_format

Create ova disk format using ovftool from https://www.vmware.com/support/developer/ovf

kiwi.storage.subformat.ova.DiskFormatOva.post_init

vmdk disk format post initialization method

Store qemu options as list from custom args dict

Parameters

`custom_args` () – custom qemu arguments dictionary

kiwi.storage.subformat.ova.DiskFormatOva.store_to_result

Store the resulting ova file into the provided result instance.

Parameters

`result` () – Instance of Result

14.21.5 `kiwi.storage.subformat.qcow2` Module

`kiwi.storage.subformat.qcow2.DiskFormatQcow2`

Bases: `DiskFormatBase` (#kiwi.storage.subformat.base.DiskFormatBase)

Create qcow2 disk format

kiwi.storage.subformat.qcow2.DiskFormatQcow2.create_image_format

Create qcow2 disk format

kiwi.storage.subformat.qcow2.DiskFormatQcow2.post_init

qcow2 disk format post initialization method

Store qemu options as list from custom args dict

Parameters

`custom_args` () – custom qemu arguments dictionary

kiwi.storage.subformat.qcow2.DiskFormatQcow2.store_to_result

Store result file of the format conversion into the provided result instance.

In case of a qcow2 format we store the result uncompressed Since the format conversion only takes the real bytes into account such that the sparseness of the raw disk will not result in the output format and can be taken one by one

Parameters

`result` () – Instance of Result

14.21.6 kiwi.storage.subformat.vagrant_base Module

kiwi.storage.subformat.vagrant_base.DiskFormatVagrantBase

Bases: DiskFormatBase (#kiwi.storage.subformat.base.DiskFormatBase)

Base class for creating vagrant boxes.

The documentation of the vagrant box format can be found here: https://www.vagrant-up.com/docs/boxes/format.html In a nutshell, a vagrant box is a tar, tar.gz or zip archive of the following:

1. metadata.json: A json file that contains the name of the provider and arbitrary additional data (that vagrant doesn't care about).
2. Vagrantfile: A Vagrantfile which defines the boxes' MAC address. It can be also used to define other settings of the box, e.g. the method via which the /vagrant/ directory is shared. This file is either automatically generated by KIWI or we use a file that has been provided by the user (depends on the setting in vagrantconfig.embebbed_vagrantfile)
3. The actual virtual disk image: this is provider specific and vagrant simply forwards it to your virtual machine provider.

Required methods/variables that child classes must implement:

- vagrant_post_init() (#kiwi.storage.subformat.vagrant_base.DiskFormatVagrantBase.vagrant_post_init)
  post initializing method that has to specify the vagrant provider name in provider and the box name in image_format. Note: new providers also needs to be specified in the schema and the box name needs to be registered to *kiwi.defaults.Defaults.get_disk_format_types*
- create_box_img() (#kiwi.storage.subformat.vagrant_base.DiskFormatVagrantBase.create_box_img)

Optional methods:

- get_additional_metadata() (#kiwi.storage.subformat.vagrant_base.DiskFormatVagrantBase.get_additional_metadata)
- get_additional_vagrant_config_settings() (#kiwi.storage.subformat.vagrant_base.DiskFormatVagrantBase.get_additional_vagrant_config_settings)

kiwi.storage.subformat.vagrant_base.DiskFormatVagrantBase.create_box_img

Provider specific image creation step: this function creates the actual box image. It must be implemented by a child class.

kiwi.storage.subformat.vagrant_base.DiskFormatVagrantBase.create_image_format

Create a vagrant box for any provider. This includes:

- creation of box metadata.json
- creation of box Vagrantfile (either from scratch or by using the user provided Vagrantfile)
- creation of result format tarball from the files created above

kiwi.storage.subformat.vagrant_base.DiskFormatVagrantBase.get_additional_metadata

Provide `create_image_format()` (#kiwi.storage.subformat.vagrant_base.DiskFormatVagrantBase.create_image_format) with additional metadata that will be included in `metadata.json`.

The default implementation returns an empty dictionary.

Returns

A dictionary that is serializable to JSON

Return type

dict

kiwi.storage.subformat.vagrant_base.DiskFormatVagrantBase.get_additional_vagrant_config_settings

Supply additional configuration settings for vagrant to be included in the resulting box.

This function can be used by child classes to customize the behavior for different providers: the supplied configuration settings get forwarded to `VagrantConfigTemplate.get_template()` as the parameter `custom_settings` and included in the `Vagrantfile`.

The default implementation returns nothing.

Returns

additional vagrant settings

Return type

str

kiwi.storage.subformat.vagrant_base.DiskFormatVagrantBase.post_init

vagrant disk format post initialization method

store vagrantconfig information provided via custom_args

Parameters

`custom_args` () –

Contains instance of xml_parse::vagrantconfig

```
{'vagrantconfig': object}
```

kiwi.storage.subformat.vagrant_base.DiskFormatVagrantBase.store_to_result

Store result file of the vagrant format conversion into the provided result instance. In this case compression is unwanted because the box is already created as a compressed tarball

Parameters

`result` () – Instance of Result

kiwi.storage.subformat.vagrant_base.DiskFormatVagrantBase.vagrant_post_init

Vagrant provider specific post initialization method

Setup vagrant provider and box name. This information must be set by the specialized provider class implementation to make the this base class methods effective

14.21.7 `kiwi.storage.subformat.vagrant_libvirt` Module

`kiwi.storage.subformat.vagrant_libvirt.DiskFormatVagrantLibVirt`

Bases: `DiskFormatVagrantBase` (#kiwi.storage.subformat.vagrant_base.DiskFormatVagrantBase)

Create a vagrant box for the libvirt provider

kiwi.storage.subformat.vagrant_libvirt.DiskFormatVagrantLibVirt.create_box_img

Creates the qcow2 disk image box for libvirt vagrant provider

Parameters

`temp_image_dir` () – Path to the temporary directory used to build the box image

Returns

A list of files relevant for the libvirt box to be included in the vagrant box

Return type

list

kiwi.storage.subformat.vagrant_libvirt.DiskFormatVagrantLibVirt.get_additional_metadata

Provide box metadata needed to create the box in vagrant

Returns

Returns a dictionary containing the virtual image format and the size of the image.

Return type

dict

kiwi.storage.subformat.vagrant_libvirt.DiskFormatVagrantLibVirt.get_additional_vagrant_config_settings

Returns settings for the libvirt provider telling vagrant to use kvm.

Returns

ruby code to be evaluated as string

Return type

str

kiwi.storage.subformat.vagrant_libvirt.DiskFormatVagrantLibVirt.vagrant_post_init

Vagrant provider specific post initialization method

Setup vagrant provider and box name. This information must be set by the specialized provider class implementation to make the this base class methods effective

14.21.8 kiwi.storage.subformat.vagrant_virtualbox Module

`kiwi.storage.subformat.vagrant_virtualbox.DiskFormatVagrantVirtualBox`

Bases: `DiskFormatVagrantBase` (#kiwi.storage.subformat.vagrant_base.DiskFormatVagrantBase)

Create a vagrant box for the virtualbox provider

kiwi.storage.subformat.vagrant_virtualbox.DiskFormatVagrantVirtualBox.create_box_img

Create the vmdk image for the Virtualbox vagrant provider.

This function creates the vmdk disk image and the ovf file. The latter is created via the class `VirtualboxOvfTemplate`.

Parameters

`temp_image_dir` () – Path to the temporary directory used to build the box image

Returns

A list of files relevant for the virtualbox box to be included in the vagrant box

Return type

list

kiwi.storage.subformat.vagrant_virtualbox.DiskFormatVagrantVirtualBox.get_additional_vagrant_config_settings

Configure the default shared folder to use rsync when guest additions are not present inside the box.

Returns

ruby code to be evaluated as string

Return type

str

kiwi.storage.subformat.vagrant_virtualbox.DiskFormatVagrantVirtualBox.vagrant_post_init

Vagrant provider specific post initialization method

Setup vagrant provider and box name. This information must be set by the specialized provider class implementation to make the this base class methods effective

14.21.9 `kiwi.storage.subformat.vdi` Module

`kiwi.storage.subformat.vdi.DiskFormatVdi`

Bases: `DiskFormatBase` (#kiwi.storage.subformat.base.DiskFormatBase)

Create vdi disk format

kiwi.storage.subformat.vdi.DiskFormatVdi.create_image_format

Create vdi disk format

kiwi.storage.subformat.vdi.DiskFormatVdi.post_init

vdi disk format post initialization method

Store qemu options as list from custom args dict

Parameters

`custom_args` () – custom qemu arguments dictionary

14.21.10 kiwi.storage.subformat.vhd Module

`kiwi.storage.subformat.vhd.DiskFormatVhd`

Bases: `DiskFormatBase` (#kiwi.storage.subformat.base.DiskFormatBase)

Create vhd disk format

kiwi.storage.subformat.vhd.DiskFormatVhd.create_image_format

Create vhd disk format

kiwi.storage.subformat.vhd.DiskFormatVhd.post_init

vhd disk format post initialization method

Store qemu options as list from custom args dict

Parameters

`custom_args` () – custom qemu arguments dictionary

14.21.11 kiwi.storage.subformat.vhdfixed Module

`kiwi.storage.subformat.vhdfixed.DiskFormatVhdFixed`

Bases: `DiskFormatBase` (#kiwi.storage.subformat.base.DiskFormatBase)

Create vhd image format in fixed subformat

kiwi.storage.subformat.vhdfixed.DiskFormatVhdFixed.create_image_format

Create vhd fixed disk format

kiwi.storage.subformat.vhdfixed.DiskFormatVhdFixed.post_init

vhd disk format post initialization method

Store qemu options as list from custom args dict Extract disk tag from custom args

Parameters

`custom_args` () –

custom vhdfixed and qemu argument dictionary

```
{'--tag': 'billing_code', '--qemu-opt': 'value'}
```

kiwi.storage.subformat.vhdfixed.DiskFormatVhdFixed.store_to_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` () – Instance of Result

14.21.12 kiwi.storage.subformat.vhdx Module

`kiwi.storage.subformat.vhdx.DiskFormatVhdx`

Bases: `DiskFormatBase` (#kiwi.storage.subformat.base.DiskFormatBase)

Create vhdx image format in dynamic subformat

kiwi.storage.subformat.vhdx.DiskFormatVhdx.create_image_format

Create vhdx dynamic disk format

kiwi.storage.subformat.vhdx.DiskFormatVhdx.post_init

vhdx disk format post initialization method

Store qemu options as list from custom args dict

Parameters

`custom_args` () – custom qemu arguments dictionary

14.21.13 kiwi.storage.subformat.vmdk Module

`kiwi.storage.subformat.vmdk.DiskFormatVmdk`

Bases: `DiskFormatBase` (#kiwi.storage.subformat.base.DiskFormatBase)

Create vmdk disk format

kiwi.storage.subformat.vmdk.DiskFormatVmdk.create_image_format

Create vmdk disk format and machine settings file

kiwi.storage.subformat.vmdk.DiskFormatVmdk.post_init

vmdk disk format post initialization method

Store qemu options as list from custom args dict

Parameters

`custom_args` () – custom qemu arguments dictionary

kiwi.storage.subformat.vmdk.DiskFormatVmdk.store_to_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` () – Instance of Result

14.21.14 Module Contents

`kiwi.storage.subformat.DiskFormat`

Bases: `object`

DiskFormat factory

Parameters

- `name` () – Format name
- `xml_state` () – Instance of XMLState
- `root_dir` () – root directory path name
- `target_dir` () – target directory path name

kiwi.storage.subformat.DiskFormat.new

14.22 kiwi.storage.subformat.template Package

14.22.1 Submodules

14.22.2 kiwi.storage.subformat.template.vmware_settings Module

`kiwi.storage.subformat.template.vmware_settings.VmwareSettingsTemplate`

Bases: `object`

VMware machine settings template

kiwi.storage.subformat.template.vmware_settings.VmwareSettingsTemplate.get_template

VMware machine configuration template

Parameters

- `memory_setup` () – with main memory setup true|false
- `cpu_setup` () – with number of CPU's setup true|false
- `network_setup` () – with network emulation true|false
- `iso_setup` () – with CD/DVD drive emulation true|false
- `disk_controller` () – add disk controller setup to template
- `iso_controller` () – add CD/DVD controller setup to template
- `network_mac` () – add static MAC address setup to template
- `network_driver` () – add network driver setup to template
- `network_connection_type` () – add connection type to template

Return type

Template

14.22.3 `kiwi.storage.subformat.template.vagrant_config` Module

`kiwi.storage.subformat.template.vagrant_config.VagrantConfigTemplate`

Bases: `object`

Generate a Vagrantfile configuration template

This class creates a simple template for the Vagrantfile that is included inside a vagrant box. The included Vagrantfile carries additional information for vagrant: by default that is nothing, but depending on the provider additional information need to be present. These can be passed via the parameter `custom_settings` to the method `get_template()` (#kiwi.storage.subformat.template.vagrant_config.VagrantConfigTemplate.get_template).

Example usage:

The default without any additional settings will result in this Vagrantfile:

```
>>> vagrant_config = VagrantConfigTemplate()
>>> print(
...     vagrant_config.get_template()
... )
```

```
Vagrant.configure("2") do |config|
end
```

If your provider/box requires additional settings, provide them as follows:

```
>>> extra_settings = dedent('''
... config.vm.hostname = "no-dead-beef"
... config.vm.provider :special do |special|
...   special.secret_settings = "please_work"
... end
... ''').strip()
>>> print(
...     vagrant_config.get_template(extra_settings)
... )
Vagrant.configure("2") do |config|
  config.vm.hostname = "no-dead-beef"
  config.vm.provider :special do |special|
    special.secret_settings = "please_work"
  end
end
```

kiwi.storage.subformat.template.vagrant_config.VagrantConfigTemplate.get_template

Return a new template with `custom_settings` included and indented appropriately.

Parameters

`custom_settings` () – String of additional settings that get pasted into the Vagrantfile template. The string is put at the correct indentation level for you, but the internal indentation has to be provided by the caller.

Returns

A string with `custom_settings` inserted at the appropriate position. The template has one the variable `mac_address` that must be substituted.

Return type

str

14.22.4 kiwi.storage.subformat.template.virtualbox_ovf Module

`kiwi.storage.subformat.template.virtualbox_ovf.VirtualboxOvfTemplate`

Bases: `object`

Generate a OVF file template for a vagrant virtualbox box

This class provides a template for virtualbox' ovf configuration file that is embedded inside the vagrant box. The template itself was extracted from a vagrant box that was build via packer and from a script provided by Neal Gompa.

kiwi.storage.subformat.template.virtualbox_ovf.VirtualboxOvfTemplate.get_template

Return the actual ovf template. The following values must be substituted: - `vm_name`: the name of this VM - `disk_image_capacity`: Size of the virtual disk image in GB - `vm_description`: a description of this VM

14.22.5 Module contents

14.23 kiwi.system Package

14.23.1 Submodules

14.23.2 `kiwi.system.identifier` Module

`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` () – hex identifier string

kiwi.system.identifier.SystemIdentifier.calculate_id

Calculate random hex id

Using 4 tuples of rand in range from 1..0xfe

kiwi.system.identifier.SystemIdentifier.get_id

Current hex identifier

Returns

hex id

Return type

str

kiwi.system.identifier.SystemIdentifier.write

Write current hex identifier to file

Parameters

`filename` () – file path name

kiwi.system.identifier.SystemIdentifier.write_to_disk

Write current hex identifier to MBR at offset 0x1b8 on disk

Parameters

`device_provider` () – Instance based on DeviceProvider

14.23.3 `kiwi.system.kernel` Module

`kiwi.system.kernel.Kernel`

Bases: `object`

Implementes kernel lookup and extraction from given root tree

Parameters

- `root_dir` () – root directory path name
- `kernel_names` () – 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

kiwi.system.kernel.Kernel.copy_kernel

Copy kernel to specified target

If no file_name is given the target filename is set as kernel-<kernel.version>.kernel

Parameters

- `target_dir` () – target path name
- `filename` () – base filename in target

kiwi.system.kernel.Kernel.copy_xen_hypervisor

Copy xen hypervisor to specified target

If no file_name is given the target filename is set as hypervisor-<xen.name>

Parameters

- `target_dir` () – target path name
- `filename` () – base filename in target

kiwi.system.kernel.Kernel.get_kernel

Lookup kernel files and provide filename and version

Parameters

`raise_on_not_found` () – sets the method to raise an exception if the kernel is not found

Raises

`kiwi.exceptions.KiwiKernelLookupError` – if raise_on_not_found flag is active and kernel is not found

Returns

tuple with filename, kernelname and version

Return type

tuple|None

kiwi.system.kernel.Kernel.get_xen_hypervisor

Lookup xen hypervisor and provide filename and hypervisor name

Returns

tuple with filename and hypervisor name

Return type

tuple|None

`kiwi.system.kernel.kernel_type`

Bases: `tuple`

kiwi.system.kernel.kernel_type.filename

Alias for field number 1

kiwi.system.kernel.kernel_type.name

Alias for field number 0

kiwi.system.kernel.kernel_type.version

Alias for field number 2

`kiwi.system.kernel.xen_hypervisor_type`

Bases: `tuple`

kiwi.system.kernel.xen_hypervisor_type.filename

Alias for field number 0

kiwi.system.kernel.xen_hypervisor_type.name

Alias for field number 1

14.23.4 `kiwi.system.prepare` Module

`kiwi.system.prepare.SystemPrepare`

Bases: `object`

Implements preparation and installation of a new root system

Parameters

- `xml_state` () – instance of `XMLState`
- `root_dir` () – Path to new root directory
- `allow_existing` () – Allow using an existing root_dir

kiwi.system.prepare.SystemPrepare.clean_package_manager_leftovers

This methods cleans some package manager artifacts created at run time such as macros

kiwi.system.prepare.SystemPrepare.delete_packages

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` () – instance of a `PackageManager` subclass
- `packages` () – package list
- `force` () – force deletion true|false

Raises

`kiwi.exceptions.KiwiSystemDeletePackagesFailed` – if installation process fails

kiwi.system.prepare.SystemPrepare.install_bootstrap

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

Parameters

- `manager` () – instance of a `PackageManager` subclass
- `plus_packages` () – list of additional packages

Raises

`kiwi.exceptions.KiwiBootStrapPhaseFailed` – if the bootstrapping process fails either installing packages or including bootstrap archives

kiwi.system.prepare.SystemPrepare.install_packages

Install one or more packages using the package manager inside of the new root directory

Parameters

- `manager` () – instance of a `PackageManager` subclass
- `packages` () – package list

Raises

`kiwi.exceptions.KiwiSystemInstallPackagesFailed` – if installation process fails

kiwi.system.prepare.SystemPrepare.install_system

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 became installed via the bootstrap phase

Parameters

`manager` () – instance of a `PackageManager` subclass

Raises

`kiwi.exceptions.KiwiInstallPhaseFailed` – if the install process fails either installing packages or including any archive

kiwi.system.prepare.SystemPrepare.pinch_system

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` () – instance of `PackageManager` subclass
- `force` () – Forced deletion True|False

Raises

`kiwi.exceptions.KiwiPackagesDeletePhaseFailed` – if the deletion packages process fails

kiwi.system.prepare.SystemPrepare.setup_ca_certificates

Setup custom CA certificates in the chroot environment. This is done by copying the certificate list from get_certificates() to the distribution specific certificate store followed by calling the distribution specific update tool.

Raises

`kiwi.exceptions.KiwiBootStrapPhaseFailed` – The CA certificates are added immediately after the bootstrap package installation, yet before the system packages are installed, therefore a bootstrap failure is raised if we catch any exception here.

kiwi.system.prepare.SystemPrepare.setup_repositories

Set up repositories for software installation and return a package manager for performing software installation tasks

Parameters

- `clear_cache` () – Flag the clear cache before configure anything
- `signing_keys` () – Keys imported to the package manager
- `target_arch` () – Target architecture name

Returns

instance of `PackageManager` subclass

Return type

`kiwi.package_manager.PackageManager`

kiwi.system.prepare.SystemPrepare.update_system

Install package updates from the used repositories. the process uses the package manager from inside of the new root directory

Parameters

`manager` () – instance of a `PackageManager` subclass

Raises

`kiwi.exceptions.KiwiSystemUpdateFailed` – if packages update fails

kiwi.system.prepare.SystemPrepare.uri_list

A list of Uri references

14.23.5 `kiwi.system.profile` Module

`kiwi.system.profile.Profile`

Bases: `object`

Create bash readable .profile environment from the XML description

Parameters

`xml_state` () – instance of :class`XMLState`

kiwi.system.profile.Profile.add

Add key/value pair to profile dictionary

Parameters

- `key` () – profile key
- `value` () – profile value

kiwi.system.profile.Profile.create

Create bash quoted profile

Parameters

`filename` () – file path name

kiwi.system.profile.Profile.delete

kiwi.system.profile.Profile.get_settings

Return all profile elements that has a value

14.23.6 kiwi.system.result Module

kiwi.system.result.Result

Bases: object

Collect image building results

Parameters

- result_files () – dict of result files
- class_version () – Result (#kiwi.system.result.Result) class version
- xml_state () – instance of XMLState

kiwi.system.result.Result.add

Add result tuple to result_files list

Parameters

- key () – name
- filename () – file path name
- use_for_bundle () – use when bundling results true|false
- compress () – compress when bundling true|false
- shasum () – create shasum when bundling true|false

kiwi.system.result.Result.add_bundle_format

kiwi.system.result.Result.dump

Picke dump this instance to a file

Parameters

- filename () – file path name
- portable () – If set to true also create a .json formatted variant of the dump file which contains the elements of this instance that could be expressed in a portable json document. Default is set to: True

Raises

kiwi.exceptions.KiwiResultError – if pickle fails to dump Result (#kiwi.system.result.Result) instance

kiwi.system.result.Result.get_results

Current list of result tuples

kiwi.system.result.Result.load

Load pickle dumped filename into a Result instance

Parameters

`filename` () – file path name

Raises

`kiwi.exceptions.KiwiResultError` – if filename does not exist or pickle fails to load filename

kiwi.system.result.Result.print_results

Print results human readable

kiwi.system.result.Result.verify_image_size

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` () – The size limit for filename in bytes.
- `filename` () – File to verify.

Raises

`kiwi.exceptions.KiwiResultError` – if filename exceeds the size limit

`kiwi.system.result.result_file_type`

Bases: `tuple`

kiwi.system.result.result_file_type.compress

Alias for field number 2

kiwi.system.result.result_file_type.filename

Alias for field number 0

kiwi.system.result.result_file_type.shasum

Alias for field number 3

kiwi.system.result.result_file_type.use_for_bundle

Alias for field number 1

kiwi.system.result.result_name_tags

Bases: tuple

kiwi.system.result.result_name_tags.A

Alias for field number 2

kiwi.system.result.result_name_tags.I

Alias for field number 3

kiwi.system.result.result_name_tags.M

Alias for field number 5

kiwi.system.result.result_name_tags.N

Alias for field number 0

kiwi.system.result.result_name_tags.P

Alias for field number 1

kiwi.system.result.result_name_tags.T

Alias for field number 4

kiwi.system.result.result_name_tags.m

Alias for field number 6

kiwi.system.result.result_name_tags.p

Alias for field number 7

kiwi.system.result.result_name_tags.v

Alias for field number 8

14.23.7 kiwi.system.root_bind Module

kiwi.system.root_bind.RootBind

Bases: object

Implements binding/copying of host system paths into the new root directory

Parameters

- root_dir () – root directory path name
- cleanup_files () – list of files to cleanup, delete

- `mount_stack` () – list of mounted directories for cleanup
- `dir_stack` () – list of directories for cleanup
- `config_files` () – list of initial config files
- `bind_locations` () – list of kernel filesystems to bind mount
- `shared_location` () – shared directory between image root and build system root

kiwi.system.root_bind.RootBind.cleanup

Cleanup mounted locations, directories and intermediate config files

kiwi.system.root_bind.RootBind.mount_kernel_file_systems

Bind mount kernel filesystems

Raises

`kiwi.exceptions.KiwiMountKernelFileSystemsError` – if some kernel filesystem fails to mount

kiwi.system.root_bind.RootBind.mount_shared_directory

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` () – directory to share between image root and build system root

Raises

`kiwi.exceptions.KiwiMountSharedDirectoryError` – if mount fails

kiwi.system.root_bind.RootBind.setup_intermediate_config

Create intermediate config files

Some config files e.g etc/hosts needs to be temporarly 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

`kiwi.exceptions.KiwiSetupIntermediateConfigError` – if the management of intermediate configuration files fails

kiwi.system.root_bind.RootBind.umount_kernel_file_systems

Umount kernel filesystems

Raises

`kiwi.exceptions.KiwiMountKernelFileSystemsError` – if some kernel filesystem fails to mount

14.23.8 `kiwi.system.root_init` Module

`kiwi.system.root_init.RootInit`

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` () – root directory path name

kiwi.system.root_init.RootInit.create

Create new system root directory

The method creates a temporary directory and initializes it for the purpose of building a system image from it. This includes the following setup:

- create core system paths
- create static core device nodes

On success the contents of the temporary location are synced to the specified root_dir and the temporary location will be deleted. That way we never work on an incomplete initial setup

Raises

`kiwi.exceptions.KiwiRootInitCreationError` – if the init creation fails at some point

kiwi.system.root_init.RootInit.delete

Force delete root directory and its contents

14.23.9 kiwi.system.setup Module

kiwi.system.setup.SystemSetup

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

- xml_state () – instance of XMLState
- root_dir () – root directory path name

kiwi.system.setup.SystemSetup.call_config_host_overlay_script

Call config-host-overlay.sh script _NON_ chrooted

kiwi.system.setup.SystemSetup.call_config_overlay_script

Call config-overlay.sh script chrooted

kiwi.system.setup.SystemSetup.call_config_script

Call config.sh script chrooted

kiwi.system.setup.SystemSetup.call_disk_script

Call disk.sh script chrooted

kiwi.system.setup.SystemSetup.call_edit_boot_config_script

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 () – boot filesystem name
- boot_part_id () – boot partition number
- working_directory () – directory name

kiwi.system.setup.SystemSetup.call_edit_boot_install_script

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` () – file path name
- `boot_device_node` () – boot device node name
- `working_directory` () – directory name

kiwi.system.setup.SystemSetup.call_image_script

Call images.sh script chrooted

kiwi.system.setup.SystemSetup.call_post_bootstrap_script

Call post_bootstrap.sh script chrooted

kiwi.system.setup.SystemSetup.call_pre_disk_script

Call pre_disk_sync.sh script chrooted

kiwi.system.setup.SystemSetup.cleanup

Delete all traces of a kiwi description which are not required in the later image

kiwi.system.setup.SystemSetup.create_fstab

Create etc/fstab from given Fstab object

Custom fstab modifications are possible and handled in the following order:

1. Look for an optional fstab.append file which allows to append custom fstab entries to the final fstab. Once embedded the fstab.append file will be deleted
2. Look for an optional fstab.patch file which allows to patch the current contents of the fstab file with a given patch file. Once patched the fstab.patch file will be deleted
3. Look for an optional fstab.script file which is called chrooted for the purpose of updating the fstab file as appropriate. Note: There is no validation in place that checks if the script actually handles fstab or any other file in the image rootfs. Once called the fstab.script file will be deleted

Parameters

`fstab` () – instance of Fstab

kiwi.system.setup.SystemSetup.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

kiwi.system.setup.SystemSetup.create_recovery_archive

Create a compressed recovery archive from the root tree for use with kiwi's recvoery system. The method creates additional data into the image root filesystem which is deleted prior to the creation of a new recovery data set

kiwi.system.setup.SystemSetup.create_system_files

Create file list of packages to be used by flake-pilot

kiwi.system.setup.SystemSetup.export_flake_pilot_system_file_list

Export image package file list to the target_dir and filename

Parameters

- `target_dir` () – path name
- `file_name` () – file name

kiwi.system.setup.SystemSetup.export_modprobe_setup

Export etc/modprobe.d to given root_dir

Parameters

`target_root_dir` () – path name

kiwi.system.setup.SystemSetup.export_package_changes

Export image package changelog for comparision of actual changes of the installed packages

Parameters

`target_dir` () – path name

kiwi.system.setup.SystemSetup.export_package_list

Export image package list as metadata reference used by the open buildservice

Parameters

`target_dir` () – path name

kiwi.system.setup.SystemSetup.export_package_verification

Export package verification result as metadata reference used by the open buildservice

Parameters

`target_dir` () – path name

kiwi.system.setup.SystemSetup.import_cdroot_files

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` () – directory to unpack archive to

kiwi.system.setup.SystemSetup.import_description

Import XML descriptions, custom scripts, archives and script helper methods

kiwi.system.setup.SystemSetup.import_files

kiwi.system.setup.SystemSetup.import_image_identifier

Create etc/ImageID identifier file

kiwi.system.setup.SystemSetup.import_overlay_files

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.

In addition the method also supports profile specific overlay files which are searched in a directory of the same name as the profile name.

The overall order for including overlay files is as follows:

1. root/ dir or root.tar.gz
2. PROFILE_NAME/ dir(s) in the order of the selected profiles

Parameters

- `follow_links` () – follow symlinks true|false
- `preserve_owner_group` () – preserve permissions true|false

kiwi.system.setup.SystemSetup.import_repositories_marked_as_imageinclude

Those <repository> sections which are marked with the imageinclude attribute should be permanently added to the image repository configuration

kiwi.system.setup.SystemSetup.script_exists

Check if provided script base name exists in the image description

Parameters

`name` () – script base name

kiwi.system.setup.SystemSetup.set_selinux_file_contexts

Initialize the security context fields (extended attributes) on the files matching the security_context_file

Parameters

`security_context_file` () – path file name

kiwi.system.setup.SystemSetup.setup_groups

Add groups for configured users

kiwi.system.setup.SystemSetup.setup_keyboard_map

Setup console keyboard

kiwi.system.setup.SystemSetup.setup_locale

Setup UTF8 system wide locale

kiwi.system.setup.SystemSetup.setup_machine_id

Setup systemd machine id

There are various states of /etc/machine-id:

1. Does not exist: Triggers ConditionFirstBoot, but does not work if the filesystem is initially read-only (booted without “rw”).
2. Exists, is empty: Does not trigger ConditionFirstBoot, but works with read-only mounts.
3. Exists, contains the string “uninitialized”: Same as b), but triggers ConditionFirstBoot. Supported by systemd v247+ only.
4. Exists, contains a valid ID.

See the machine-id(5) man page for details.

In images, d) is not desirable, so truncate the file. This is the previous behaviour and what existing images expect. If the image has one of the other states, keep it as-is.

kiwi.system.setup.SystemSetup.setup_permissions

Check and Fix permissions using chkstat

Call chkstat in system mode which reads /etc/sysconfig/security to determine the configured security level and applies the appropriate permission definitions from the /etc/permissions* files. It’s possible to provide those files as overlay files in the image description to apply a certain permission setup when needed. Otherwise the default setup as provided on the package level applies.

It's required that the image root system has chkstat installed. If not present KIWI skips this step and continuous with a warning.

kiwi.system.setup.SystemSetup.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

kiwi.system.setup.SystemSetup.setup_registry_import

Fetch container(s) and activate systemd unit to load containers from local oci-archive file during boot

kiwi.system.setup.SystemSetup.setup_selinux_file_contexts

Set SELinux file security contexts if the default context file is found

kiwi.system.setup.SystemSetup.setup_timezone

Setup timezone symlink

kiwi.system.setup.SystemSetup.setup_users

Add/Modify configured users

14.23.10 `kiwi.system.shell` Module

`kiwi.system.shell.Shell`

Bases: `object`

Special character handling for shell evaluated code

kiwi.system.shell.Shell.format_to_variable_value

Format given variable value to return a string value representation that can be sourced by shell scripts. If the provided value is not representable as a string (list, dict, tuple etc) an exception is raised

Parameters

`value` () – a python variable

Raises

`kiwi.exceptions.KiwiShellVariableValueError` – if value is an iterable

Returns

string value representation

Return type

str

kiwi.system.shell.Shell.quote

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` () – message text

Returns

quoted text

Return type

str

kiwi.system.shell.Shell.quote_key_value_file

Quote given input file which has to be of the form key = value to be able to become sourced by the shell

Parameters

`filename` () – file path name

Returns

list of quoted text

Return type

List[str]

kiwi.system.shell.Shell.run_common_function

Run a function implemented in config/functions.sh

Parameters

- `name` () – function name
- `parameters` () – function arguments

14.23.11 kiwi.system.size Module

kiwi.system.size.SystemSize

Bases: object

Provide source tree size information

Parameters

source_dir () – source directory path name

kiwi.system.size.SystemSize.accumulate_files

Calculate sum of all files in the source tree

Returns

number of files

Return type

int

kiwi.system.size.SystemSize.accumulate_mbyte_file_sizes

Calculate data size of all data in the source tree

Parameters

exclude () – list of paths to exclude

Returns

mbytes

Return type

int

kiwi.system.size.SystemSize.customize

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 () – mbsize to update
- requested_filesystem () – filesystem name

Returns

mbytes

Return type

int

14.23.12 `kiwi.system.uri` Module

`kiwi.system.uri.Uri`

Bases: `object`

Normalize and manage URI types

kiwi.system.uri.Uri.alias

Create hex representation of uuid4

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 hex representation of uuid4

Return type

str

kiwi.system.uri.Uri.credentials_file_name

Filename to store repository credentials

Returns

credentials file name

Return type

str

kiwi.system.uri.Uri.get_fragment

Returns the fragment part of the URI.

Returns

fragment part of the URI if any, empty string otherwise

Return type

str

kiwi.system.uri.Uri.is_public

Check if URI is considered to be publicly reachable

Returns

True|False

Return type

bool

kiwi.system.uri.Uri.is_remote

Check if URI is a remote or local location

Returns

True|False

Return type

bool

kiwi.system.uri.Uri.print_sensitive

kiwi.system.uri.Uri.translate

Translate repository location according to their URI type
Depending on the URI type the provided location needs to be adapted e.g updated by the service URL in case of an open buildservice project name

Raises

`kiwi.exceptions.KiwiUriStyleUnknown` – if the uri scheme can't be detected, is unknown or it is inconsistent with the build environment

Parameters

`check_build_environment` () – 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

Returns

translated repository location

Return type

str

14.23.13 kiwi.system.users Module

kiwi.system.users.Users

Bases: object

Operations on users and groups in a root directory

Parameters

root_dir () – root directory path name

kiwi.system.users.Users.group_add

Add group with options

Parameters

- group_name () – group name
- options () – groupadd options

kiwi.system.users.Users.group_exists

Check if group exists

Parameters

group_name () – group name

Returns

True|False

Return type

bool

kiwi.system.users.Users.setup_home_for_user

Setup user home directory

Parameters

- user_name () – user name
- group_name () – group name
- home_path () – path name

kiwi.system.users.Users.user_add

Add user with options

Parameters

- `user_name` () – user name
- `options` () – useradd options

kiwi.system.users.Users.user_exists

Check if user exists

Parameters

`user_name` () – user name

Returns

True|False

Return type

bool

kiwi.system.users.Users.user_modify

Modify user with options

Parameters

- `user_name` () – user name
- `options` () – usermod options

14.23.14 Module Contents

14.24 kiwi.tasks package

14.24.1 Submodules

14.24.2 `kiwi.tasks.base` Module

`kiwi.tasks.base.CliTask`

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

kiwi.tasks.base.CliTask.attr_token

Helper method for commandline options of the form –option attribute=value

Parameters

`option` () – attribute=value string

Returns

common option value representation

Return type

list

kiwi.tasks.base.CliTask.eleventuple_token

Helper method for commandline options of the form –option a,b,c,d,e,f,g,h,i,j,k
Make sure to provide a common result for option values which separates the information in a comma separated list of values

Parameters

`option` () – comma separated option string

Returns

common option value representation

Return type

list

kiwi.tasks.base.CliTask.load_xml_description

Load, upgrade, validate XML description

Parameters

- `description_directory` () – Path to the image description
- `kiwi_file` () – Basename of kiwi file which contains the main image configuration elements. If not specified kiwi searches for a file named config.xml or a file matching .kiwi

kiwi.tasks.base.CliTask.quadruple_token

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

Parameters

`option` () – comma separated option string

Returns

common option value representation

Return type

list

kiwi.tasks.base.CliTask.run_checks

This method runs the given runtime checks excluding the ones disabled in the runtime configuration file.

Parameters

`checks` () – A dictionary with the runtime method names as keys and their arguments list as the values.

14.24.3 `kiwi.tasks.result_bundle` Module

`kiwi.tasks.result_bundle.ResultBundleTask`

Bases: `CliTask` (#kiwi.tasks.base.CliTask)
Implements result bundler
Attributes

- `manual`
  Instance of Help

kiwi.tasks.result_bundle.ResultBundleTask.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

14.24.4 kiwi.tasks.result_list Module

kiwi.tasks.result_list.ResultListTask

Bases: CliTask (#kiwi.tasks.base.CliTask)

Implements result listing

Attributes

- manual

  Instance of Help

kiwi.tasks.result_list.ResultListTask.process

List result information from a previous system command

14.24.5 kiwi.tasks.system_build Module

kiwi.tasks.system_build.SystemBuildTask

Bases: CliTask (#kiwi.tasks.base.CliTask)

Implements building of system images

Attributes

- manual

  Instance of Help

kiwi.tasks.system_build.SystemBuildTask.process

Build a system image from the specified description. The build command combines the prepare and create commands

14.24.6 kiwi.tasks.system_create Module

kiwi.tasks.system_create.SystemCreateTask

Bases: CliTask (#kiwi.tasks.base.CliTask)

Implements creation of system images

Attributes

- `manual`

  Instance of Help

kiwi.tasks.system_create.SystemCreateTask.process

Create a system image from the specified root directory the root directory is the result of a system prepare command

14.24.7 `kiwi.tasks.system_prepare` Module

`kiwi.tasks.system_prepare.SystemPrepareTask`

Bases: `CliTask` (#kiwi.tasks.base.CliTask)

Implements preparation and installation of a new root system

Attributes

- `manual`

  Instance of Help

kiwi.tasks.system_prepare.SystemPrepareTask.process

Prepare and install a new system for chroot access

14.24.8 `kiwi.tasks.system_update` Module

`kiwi.tasks.system_update.SystemUpdateTask`

Bases: `CliTask` (#kiwi.tasks.base.CliTask)

Implements update and maintenance of root systems

Attributes

- `manual`

  Instance of Help

kiwi.tasks.system_update.SystemUpdateTask.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

14.24.9 Module Contents

14.25 kiwi.utils Package

14.25.1 Submodules

14.25.2 kiwi.utils.checksum Module

kiwi.utils.block.BlockID

Bases: object

Get information from a block device

Parameters

device () – block device node name name. The device can also be specified as UUID=<uuid>

kiwi.utils.block.BlockID.get_blkid

Retrieve information for specified metadata ID from block device

Parameters

id_type () – metadata ID, see man blkid for details

Returns

ID of the block device

Return type

str

kiwi.utils.block.BlockID.get_filesystem

Retrieve filesystem type from block device

Returns

filesystem type

Return type

str

kiwi.utils.block.BlockID.get_label

Retrieve filesystem label from block device

Returns

block device label

Return type

str

kiwi.utils.block.BlockID.get_partition_count

Retrieve number of partitions from block device

Returns

A number

Return type

int

kiwi.utils.block.BlockID.get_ptuuid

Retrieve partition uuid from block device

Returns

uuid of the partition table

Return type

str

kiwi.utils.block.BlockID.get_uuid

Retrieve filesystem uuid from block device

Returns

uuid for the filesystem of the block device

Return type

str

14.25.3 `kiwi.utils.block` Module

`kiwi.utils.checksum.Checksum`

Bases: `object`

Manage checksum creation for files

Parameters

- `source_filename` () – source file name to build checksum for
- `checksum_filename` () – target file with checksum information

kiwi.utils.checksum.Checksum.matches

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` () – checksum string to compare
- `filename` () – filename containing checksum

Returns

True or False

Return type

bool

kiwi.utils.checksum.Checksum.md5

kiwi.utils.checksum.Checksum.sha256

Create sha256 checksum

Parameters

`filename` () – filename for checksum

14.25.4 `kiwi.utils.compress` Module

`kiwi.utils.compress.Compress`

Bases: `object`

File compression / decompression

Parameters

- `keep_source` () – Request to keep the uncompressed source
- `source_filename` () – Source file name to compress

- `supported_zipper` () – List of supported compression tools
- `compressed_filename` () – Compressed file name path with compression suffix
- `uncompressed_filename` () – Uncompressed file name path

kiwi.utils.compress.Compress.get_format

Detect compression format

Returns

compression format name or None if it couldn't be inferred

Return type

Optional[str]

kiwi.utils.compress.Compress.gzip

Create gzip(max compression) compressed file

kiwi.utils.compress.Compress.uncompress

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` () – uncompress to a temporary file

kiwi.utils.compress.Compress.xz

Create XZ compressed file

Parameters

`options` () – custom xz compression options

14.25.5 `kiwi.utils.sync` Module

`kiwi.utils.sync.DataSync`

Bases: `object`

Sync data from a source directory to a target directory

kiwi.utils.sync.DataSync.sync_data

Sync data from source to target using the rsync protocol

Parameters

- `options` () – rsync options
- `exclude` () – file patterns to exclude
- `force_trailing_slash` () – add '/' to source_dir if not present

A speciality of the rsync tool is that it behaves differently if the given source_dir ends with a '/' or not. If it ends with a slash the data structure below will be synced to the target_dir. If it does not end with a slash the source_dir and its contents are synced to the target_dir. For example

```
source
  └── some_data

1. $ rsync -a source target

target
  └── source
        └── some_data

2. $ rsync -a source/ target

target
  └── some_data
```

The parameter force_trailing_slash can be used to make sure rsync behaves like shown in the second case. If set to true a '/' is appended to the given source_dir if not already present

kiwi.utils.sync.DataSync.target_supports_extended_attributes

Check if the target directory supports extended filesystem attributes

Returns

True or False

Return type

bool

14.25.6 `kiwi.utils.sysconfig` Module

`kiwi.utils.sysconfig.SysConfig`

Bases: `object`

Read and Write sysconfig style files

Parameters

`source_file` () – source file path

kiwi.utils.sysconfig.SysConfig.get

kiwi.utils.sysconfig.SysConfig.write

Write back source file with changed content but in same order

14.25.7 Module Contents

14.26 kiwi.volume_manager Package

14.26.1 Submodules

14.26.2 `kiwi.volume_manager.base` Module

`kiwi.volume_manager.base.VolumeManagerBase`

Bases: `kiwi.storage.device_provider.DeviceProvider`

Implements base class for volume management interface

Parameters

- `device_map` () – dictionary of low level DeviceProvider intances
- `root_dir` () – root directory path name
- `volumes` () – list of volumes from `XMLState::get_volumes()`
- `custom_args` () – custom volume manager arguments for all volume manager and filesystem specific tasks

Raises

`kiwi.exceptions.KiwiVolumeManagerSetupError` – if the given root_dir doesn't exist

kiwi.volume_manager.base.VolumeManagerBase.apply_attributes_on_volume

kiwi.volume_manager.base.VolumeManagerBase.create_verification_metadata
Write verification block on LVM devices is not supported

kiwi.volume_manager.base.VolumeManagerBase.create_verity_layer
veritysetup on LVM devices is not supported

kiwi.volume_manager.base.VolumeManagerBase.create_volume_paths_in_root_dir
Implements creation of volume paths in the given root directory

kiwi.volume_manager.base.VolumeManagerBase.create_volumes
Implements creation of volumes
Implementation in specialized volume manager class required

Parameters
`filesystem_name` () – unused

kiwi.volume_manager.base.VolumeManagerBase.custom_args
custom arguments passed to setup the volumes

kiwi.volume_manager.base.VolumeManagerBase.custom_filesystem_args
custom filesystem creation and mount arguments, subset of the custom_args information suitable to be passed to a FileSystem instance

kiwi.volume_manager.base.VolumeManagerBase.device
main storage device node name

kiwi.volume_manager.base.VolumeManagerBase.device_map
dictionary of mapped DeviceProviders

kiwi.volume_manager.base.VolumeManagerBase.device_provider_root
the underlaying device provider

kiwi.volume_manager.base.VolumeManagerBase.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

kiwi.volume_manager.base.VolumeManagerBase.get_device

Return current DeviceProvider dictionary

Returns

device_map

Return type

dict

kiwi.volume_manager.base.VolumeManagerBase.get_fstab

Implements setup of the fstab entries. The method should return a list of fstab compatible entries

Parameters

- `persistency_type` () – unused
- `filesystem_name` () – unused

kiwi.volume_manager.base.VolumeManagerBase.get_mountpoint

Provides mount point directory

Effective use of the directory is guaranteed only after sync_data

Returns

directory path name

Return type

string

kiwi.volume_manager.base.VolumeManagerBase.get_root_volume_name

Provides name of the root volume

This is by default set to '/'. Volume Managers that supports the concept of sub-volumes overrides this method

Returns

directory path name

Return type

string

kiwi.volume_manager.base.VolumeManagerBase.get_volume_mbsize

Implements size lookup for the given path and desired filesystem according to the specified size type

Parameters

- `volume` () – volume to check size for
- `all_volumes` () – list of all volume tuples
- `filesystem_name` () – filesystem name
- `resize_on_boot` – specify the time of the resize. If the resize happens at boot time the volume size is only the minimum size to just store the data. If the volume size is fixed and does not change at boot time the returned size is the requested size which can be greater than the minimum needed size.

Returns

mbsize

Return type

int

kiwi.volume_manager.base.VolumeManagerBase.get_volumes

Implements return of dictionary of volumes and their mount options

kiwi.volume_manager.base.VolumeManagerBase.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

kiwi.volume_manager.base.VolumeManagerBase.mount

Consistency layer with regards to FileSystem classes

Invokes mount_volumes

kiwi.volume_manager.base.VolumeManagerBase.mount_list

list of volume MountManager's

kiwi.volume_manager.base.VolumeManagerBase.mount_volumes

Implements mounting of all volumes below one master directory

Implementation in specialized volume manager class required

kiwi.volume_manager.base.VolumeManagerBase.mountpoint

all volumes are combined into one mountpoint. This is needed at sync_data time. How to mount the volumes is special to the volume management class

kiwi.volume_manager.base.VolumeManagerBase.post_init

Post initialization method

Implementation in specialized volume manager class if required

Parameters

`custom_args` () – unused

kiwi.volume_manager.base.VolumeManagerBase.root_dir

root directory path name

kiwi.volume_manager.base.VolumeManagerBase.set_property_readonly_root

Implements setup of read-only root property

kiwi.volume_manager.base.VolumeManagerBase.setup

Implements setup required prior to the creation of volumes

Implementation in specialized volume manager class required

Parameters

`name` () – unused

kiwi.volume_manager.base.VolumeManagerBase.setup_mountpoint

Implements creation of a master directory holding the mounts of all volumes

kiwi.volume_manager.base.VolumeManagerBase.sync_data

Implements sync of root directory to mounted volumes

Parameters

`exclude` () – file patterns to exclude

Returns

If a mount was created, then a context manager implementing the unmount is returned.

kiwi.volume_manager.base.VolumeManagerBase.umount

Consistency layer with regards to FileSystem classes

Invokes umount_volumes

kiwi.volume_manager.base.VolumeManagerBase.umount_volumes

Implements umounting of all volumes

Implementation in specialized volume manager class required

kiwi.volume_manager.base.VolumeManagerBase.volume_group

volume group name

kiwi.volume_manager.base.VolumeManagerBase.volume_map

map volume name to device node

kiwi.volume_manager.base.VolumeManagerBase.volumes

list of volumes from `XMLState::get_volumes()`

14.26.3 `kiwi.volume_manager.btrfs` Module

`kiwi.volume_manager.btrfs.VolumeManagerBtrfs`

Bases: `VolumeManagerBase` (#kiwi.volume_manager.base.VolumeManagerBase)

Implements btrfs sub-volume management

Parameters

- `subvol_mount_list` () – list of mounted btrfs subvolumes
- `toplevel_mount` () – `MountManager` for root mountpoint

kiwi.volume_manager.btrfs.VolumeManagerBtrfs.create_volumes

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` () – unused

kiwi.volume_manager.btrfs.VolumeManagerBtrfs.get_fstab

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

Parameters

- `persistency_type` () – by-label | by-uuid
- `filesystem_name` () – unused

Returns

list of fstab entries

Return type

list

kiwi.volume_manager.btrfs.VolumeManagerBtrfs.get_mountpoint

Provides btrfs root mount point directory

Effective use of the directory is guaranteed only after sync_data

Returns

directory path name

Return type

string

kiwi.volume_manager.btrfs.VolumeManagerBtrfs.get_root_volume_name

Provides name of the root volume

Returns

directory path name

Return type

string

kiwi.volume_manager.btrfs.VolumeManagerBtrfs.get_volumes

Return dict of volumes

Returns

volumes dictionary

Return type

dict

kiwi.volume_manager.btrfs.VolumeManagerBtrfs.mount_volumes

Mount btrfs subvolumes

kiwi.volume_manager.btrfs.VolumeManagerBtrfs.post_init

Post initialization method

Store custom btrfs initialization arguments

Parameters

`custom_args` () – custom btrfs volume manager arguments

kiwi.volume_manager.btrfs.VolumeManagerBtrfs.set_property_readonly_root

Sets the root volume to be a readonly filesystem

kiwi.volume_manager.btrfs.VolumeManagerBtrfs.setup

Setup btrfs volume management

In case of btrfs an optional toplevel subvolume is created and marked as default volume. If snapshots are activated via the custom_args the setup method also creates 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` () – unused

kiwi.volume_manager.btrfs.VolumeManagerBtrfs.sync_data

Sync data into btrfs filesystem

If snapshots are activated the root filesystem is synced into the first snapshot

Parameters

`exclude` () – files to exclude from sync

kiwi.volume_manager.btrfs.VolumeManagerBtrfs.umount_volumes

Umount btrfs subvolumes

14.26.4 `kiwi.volume_manager.lvm` Module

`kiwi.volume_manager.lvm.VolumeManagerLVM`

Bases: `VolumeManagerBase` (#kiwi.volume_manager.base.VolumeManagerBase)

Implements LVM volume management

kiwi.volume_manager.lvm.VolumeManagerLVM.create_volumes

Create configured lvm volumes and filesystems

All volumes receive the same filesystem

Parameters

`filesystem_name` () – volumes filesystem name

kiwi.volume_manager.lvm.VolumeManagerLVM.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

kiwi.volume_manager.lvm.VolumeManagerLVM.get_fstab

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

Parameters

- `persistency_type` () – unused
- `filesystem_name` () – volumes filesystem name

Returns

fstab entries

Return type

list

kiwi.volume_manager.lvm.VolumeManagerLVM.get_volumes

Return dict of volumes

Returns

volumes dictionary

Return type

dict

kiwi.volume_manager.lvm.VolumeManagerLVM.mount_volumes

Mount lvm volumes

kiwi.volume_manager.lvm.VolumeManagerLVM.post_init

Post initialization method

Store custom lvm initialization arguments

Parameters

`custom_args` () – custom lvm volume manager arguments

kiwi.volume_manager.lvm.VolumeManagerLVM.setup

Setup lvm volume management

In case of LVM a new volume group is created on a PV initialized storage device

Parameters

`name` () – volume group name

kiwi.volume_manager.lvm.VolumeManagerLVM.umount_volumes

Umount lvm volumes

14.26.5 Module Contents

`kiwi.volume_manager.VolumeManager`

Bases: `object`

VolumeManager factory

Parameters

- `name` () – volume management name
- `device_map` () – dictionary of low level DeviceProvider intances
- `root_dir` () – root directory path name
- `volumes` () – list of volumes from `XMLState::get_volumes()`
- `custom_args` () – dictionary of custom volume manager arguments

kiwi.volume_manager.VolumeManager.new