Simple Disk Image

Dec 01, 2022

Contents:

CHAPTER 1

Examples

1.1 FAT and ext partition on GPT

```
import logging
import os

import simplediskimage

logging.basicConfig(level=logging.DEBUG)

def main():
    image = simplediskimage.DiskImage("foo.img", partition_table='gpt')
    part_fat = image.new_partition("fat12", partition_flags=["BOOT"],
                                   partition_label="EFI System Partition")
    part_ext = image.new_partition("ext4", filesystem_label="hello")

    # Allocate some extra data on top of what's taken up by the data on the
    # ext partition
    part_ext.set_extra_bytes(16 * simplediskimage.SI.Mi)

    # Create two directories in the root of each partition
    part_fat.mkdir("fat1", "fat2")
    part_ext.mkdir("ext1", "ext2")

    # Copy the testdata dir into each partition in some interesting ways
    datadir = os.path.abspath(os.path.join(os.path.dirname(__file__), "testdata"))
    for part in (part_fat, part_ext):
        part.copy(os.path.join(datadir, "x"), os.path.join(datadir, "y"))
        part.mkdir("internet")
        part.copy(os.path.join(datadir, "internet/z"), destination="internet")
        part.copy(os.path.join(datadir, "internet/recursive_copy"), destination=
↪"internet")

    image.commit()
```

(continues on next page)

(continued from previous page)

```
    print("sudo kpartx -av foo.img")
    print("...")
    print("sudo kpartx -dv foo.img")

if __name__ == '__main__':
    main()
```

1.2 Single bootable FAT partition, using sfdisk

```
import logging

import simplediskimage

from common import generate_bb_testdata

logging.basicConfig(level=logging.DEBUG)

def main():
    # Generate test data
    generate_bb_testdata()

    # Create image
    image = simplediskimage.DiskImage("bar.img", partition_table='msdos',
                                      partitioner=simplediskimage.Sfdisk)
    part_fat = image.new_partition("fat16", partition_flags=["BOOT"])

    # Copy the files to the root, could also be written:
    # part_fat.copy("file1", "file2", destination="/"), or without destination
    part_fat.copy("generated/u-boot.img")
    part_fat.copy("generated/MLO")

    # Make sure that the partition is always 48 MiB
    part_fat.set_fixed_size_bytes(48 * simplediskimage.SI.Mi)

    image.commit()
    print("sudo kpartx -av bar.img")
    print("...")
    print("sudo kpartx -dv bar.img")

if __name__ == '__main__':
    main()
```

1.3 Using the NullPartitioner to create a raw image

```
import logging
import os

import simplediskimage

logging.basicConfig(level=logging.DEBUG)

```

(continues on next page)

(continued from previous page)

```
def main():
    image = simplediskimage.DiskImage("null-partitioner.ext4",
                                      partition_table='null',
                                      partitioner=simplediskimage.NullPartitioner)
    part = image.new_partition("ext4")

    # Allocate some extra data on top of what's taken up by the data on the
    # ext partition
    part.set_extra_bytes(2 * simplediskimage.SI.Mi)

    # Create two directories in the root of the partition
    part.mkdir("ext1", "ext2")

    # Copy some data from the testdata dir into the image
    datadir = os.path.abspath(os.path.join(os.path.dirname(__file__), "testdata"))
    part.copy(os.path.join(datadir, "x"), os.path.join(datadir, "y"))

    image.commit()
    print("sudo mount null-partitioner.ext4 /mnt")
    print("...")
    print("sudo umount /mnt")

if __name__ == '__main__':
    main()
```

1.4 Using a raw image as the filesystem for a partition

```
import logging
import os

import simplediskimage

from common import generate_bb_testdata

logging.basicConfig(level=logging.DEBUG)

def main():
    ext4_source = "./null-partitioner.ext4"
    # Make sure the raw ext4 image has been created by running the
    # null-partitioner example
    if not os.path.exists(ext4_source):
        print("Run the null-partitioner.py example first")

    # Generate test data
    generate_bb_testdata()

    # Create image
    image = simplediskimage.DiskImage("raw-filesystem-on-p2.img",
                                      partition_table='msdos',
                                      partitioner=simplediskimage.Sfdisk)
    part_fat = image.new_partition("fat16", partition_flags=["BOOT"])
    part_ext = image.new_partition("ext4", raw_filesystem_image=True)

    # Copy the files to the root, could also be written:
```

(continues on next page)

(continued from previous page)

```
    # part_fat.copy("file1", "file2", destination="/"), or without destination
    part_fat.copy("generated/u-boot.img")
    part_fat.copy("generated/MLO")

    # Make sure that the partition is always 48 MiB
    part_fat.set_fixed_size_bytes(48 * simplediskimage.SI.Mi)

    # Copy the ext image into the raw partition
    part_ext.copy(ext4_source)

    # The partition can be expanded beyond the size of the image, but beware
    # the warnings in the documentation before doing something like this!
    #part_ext.set_extra_bytes(16 * simplediskimage.SI.Mi)

    image.commit()
    print("sudo kpartx -av raw-filesystem-on-p2.img")
    print("...")
    print("sudo kpartx -dv raw-filesystem-on-p2.img")

if __name__ == '__main__':
    main()
```

1.5 Initialize a file system using a rootfs archive

```
# This example might look a bit convoluted as we create an archive here just to
# extract it in the companion python script, but we are simulating a situation
# where a separate build system delivers a tar (might as well be a cpio
# archive), which the python script will unpack and use as the basis of a
# partition. There are multiple variants to this, for example:
#
# - The rootfs is delivered as a directory. In this case, the image creation
#   script might be run in the same fakeroot session, or the -s and -i
#   arguments to fakeroot may be used.
# - The rootfs is created using some python scripts under a fakeroot session.
#   If so, just add some calls to simplediskimage when the rootfs has been
#   created to turn it into an image.

TEMPDIR=./.rootfs-in-p2.tmp
EXAMPLE_USER=nobody
if ! id nobody &> /dev/null; then
  EXAMPLE_USER=$USER
fi

# Create rootfs.tar, this is usually done in a build system
rm -rf $TEMPDIR
mkdir $TEMPDIR
fakeroot <<EOF
set -e
cd $TEMPDIR
mkdir rootfs
mkdir -p rootfs/{root,dev,home/nobody}
echo data > rootfs/home/nobody/data
chown -R nobody:$(id -gn nobody) rootfs/home/nobody
mknod rootfs/dev/null c 1 3
```

(continues on next page)

(continued from previous page)

```
dd if=/dev/urandom bs=1M count=16 of=rootfs/large_file
ln rootfs/large_file rootfs/large_file_link
tar -cf rootfs.tar -C rootfs .
EOF
rm -rf $TEMPDIR/rootfs

set -x
ls $TEMPDIR

# Invoke the example under fakeroot
fakeroot ./_rootfs-in-p2.py $TEMPDIR

rm -rf $TEMPDIR
```

```
import logging
import sys
import os
import tarfile

import simplediskimage

from common import generate_bb_testdata

logging.basicConfig(level=logging.DEBUG)

def main():
    # Check our surroundings, we should be:
    # - Getting the tempdir containing rootfs.tar as the sole argument
    # - Be executed inside a fakeroot environment
    if len(sys.argv) != 2 or "FAKEROOTKEY" not in os.environ:
        print("Don't run me directly, use rootfs-in-p2.sh")
        sys.exit(1)

    # Generate some testdata for the boot partition
    generate_bb_testdata()

    # Create image
    image = simplediskimage.DiskImage("rootfs-in-p2.img",
                                      partition_table='msdos',
                                      partitioner=simplediskimage.Sfdisk)
    part_fat = image.new_partition("fat16", partition_flags=["BOOT"])
    part_ext = image.new_partition("ext4", filesystem_label="root")

    # Copy files to the boot partition and set a fixed size
    part_fat.copy("generated/u-boot.img")
    part_fat.copy("generated/MLO")
    part_fat.set_fixed_size_bytes(48 * simplediskimage.SI.Mi)

    # Unpack the rootfs.tar file into a temporary directory
    temp_dir = sys.argv[1]
    rootfs_tar = os.path.join(temp_dir, "rootfs.tar")
    rootfs_dir = os.path.join(temp_dir, "p2-rootfs-dir")
    with tarfile.open(rootfs_tar, 'r:') as tf:
        def is_within_directory(directory, target):

            abs_directory = os.path.abspath(directory)
```

(continues on next page)

(continued from previous page)

```
            abs_target = os.path.abspath(target)

            prefix = os.path.commonprefix([abs_directory, abs_target])

            return prefix == abs_directory

        def safe_extract(tar, path=".", members=None, *, numeric_owner=False):

            for member in tar.getmembers():
                member_path = os.path.join(path, member.name)
                if not is_within_directory(path, member_path):
                    raise Exception("Attempted Path Traversal in Tar File")

            tar.extractall(path, members, numeric_owner=numeric_owner)


        safe_extract(tf, rootfs_dir)

    # Use the rootfs directory as the initial data directory for the second
    # partition (the rootfs)
    part_ext.set_initial_data_root(rootfs_dir)

    image.commit()
    print("sudo kpartx -av rootfs-in-p2.img")
    print("...")
    print("sudo kpartx -dv rootfs-in-p2.img")

if __name__ == '__main__':
    main()
```

CHAPTER 2

Main module

Module used to create simpler disk images, typically to boot embedded systems.

For more information see: https://github.com/zqad/simplediskimage/

class simplediskimage.**DiskImage**(*path*, *partition_table='gpt'*, *temp_fmt='{path}-{extra}.tmp'*, *partitioner=<class 'simplediskimage.partitioners.PyParted'>*, *clean_temp_files='always'*)

Bases: object

Helper class to generate disk images.

Parameters

- **path** – Path to the destination image file.
- **partition_table** – Partition table (label) format, *gpt* or *msdos*, (or 'null' for the Null-Partitioner).
- **temp_fmt** – Format/path of the temp files containing format strings for *path* and *extra*. Make sure the temp files are on the same filesystem as the destination path.
- **partitioner** – Partitioner class, for example PyParted or Sfdisk.
- **clean_temp_files** – Whether or not to retain the temp files, accepts either *"always"*, *"not on error"* or *"never"*. Default is *"always"*. Note that an unconditional clean is usually run before image creation, and that the image is moved in place meaning that its temp file will disappear on successful runs.

check()

Check this disk image for errors that will hinder us from doing a *commit()* later.

Will call *.check()* for each partition too.

commit()

Commit this disk image and create the image.

get_size_bytes()

Calculate and return the size of the disk image.

new_partition(*filesystem*, *partition_label=None*, *partition_flags=None*, *filesystem_label=None*, *raw_filesystem_image=False*)

Create a new partition on this disk image.

Parameters

- **filesystem** – Filesystem, e.g. ext3 or fat32.
- **partition_label** – Partition label, only supported by GPT.
- **partition_flags** – Partition flags, e.g. BOOT.
- **filesystem_label** – Filesystem label to be passed to mkfs.
- **raw_filesystem_image** – Flag that this partition will be populated using a raw filesystem image

class simplediskimage.**Partition**(*disk_image*, *path*, *filesystem*, *blocksize*, *metadata=None*)

Bases: object

Create partition instance, do not call directly, use *Diskimage.new_partition()*.

Parameters

- **disk_image** – Disk image instance.
- **path** – Path to the partition temp file.
- **filesystem** – Filesystem for this partition.
- **blocksize** – Block (sector) size.
- **metadata** – Metadata.

check()

Run a check of this partition, also called by DiskImage.

clean()

Clean up all temp files of this partition.

commit()

Commit this partition to it's temp file, do not call directly.

copy(**source_paths*, *destination='/'*)

Copy one or more files or directories recursively to the destination directory.

Parameters

- **source_paths** – The files to copy.
- **destination** – The destination to which to copy, default */*.

get_content_size_bytes()

Get the size of all content copied into this image so far.

get_total_size_bytes()

Get the total size of this image, using the fixed size if set, or the content + extra bytes if not.

mkdir(**dirs*)

Create one or many directories.

Parameters **dirs** – The directories to create.

set_extra_bytes(*num*)

Set the extra bytes to be added to the size on top of the content size.

Warning: When writing raw filesystem images to a partition, setting the partition size to something other than the size specified by the file system headers *will* confuse some partition parsing implementations. Notably, this has been observed with U-boot and FAT.

Parameters **num** – The number of bytes, see the SI class for conversion.

set_fixed_size_bytes(*num*)
Set a fixed size of this partition. For raw filesystem images, see the warnings under *set_extra_bytes()*.

Parameters **num** – The number of bytes, see the SI class for conversion.

set_initial_data_root(*source_path*)
Set the initial data root directory, to be used when initializing the file system. All contents of this directory will be included in the file system. This differs from *copy()* in a few ways:

- The path will be used as the file system root, rather than being copied as a file/directory under the root
- The users and unix rights will be preserved, unlike *copy()* which always writes files owned by uid 0/gid 0.
- Hard links are handled correctly, and not copied twice

Note that this feature is only supported for the ext family of file systems.

Parameters **source_paths** – The directory to use as the filesystem root.

class simplediskimage.**RawPartition**(*disk_image*, *temp_path*, *filesystem*, *metadata*)
Bases: *simplediskimage.Partition*

Simplified Partition class, used for partitions without filesystems. Only supports one file being copied (the raw image). Do not call directly, use *Diskimage.new_partition()*.

Parameters

- **disk_image** – Disk image instance.
- **temp_path** – Temporary partition part, only used if the partition is instructed to grow beyond the image size.
- **filesystem** – Filesystem for this partition.
- **metadata** – Metadata.

check()
Run a check of this partition, also called by DiskImage.

clean()
Usually a no-op, unless we ended up creating the temp file

commit()
Usually a no-op, unless extra_bytes was set, or fixed_size_bytes does not equal the size of the image

copy(**source_paths*, *destination='/'*)
Copy one or more files or directories recursively to the destination directory.

Parameters

- **source_paths** – The files to copy (only one file supported).
- **destination** – The destination to which to copy, must be left out or /.

mkdir(**dirs*)
Not supported

set_initial_data_root(*source_path*)
Not supported

CHAPTER 3

Submodules

These modules are generally only interesting if you debug or want to extend this library.

3.1 simplediskimage.cfr module

Wrappers and implementations of *copy_file_range*

simplediskimage.cfr.**get_copy_file_range**()
Get best suited copy_file_range implementation

Returns *copy_file_range* function

simplediskimage.cfr.**naive_copy_file_range**(*src_fd*, *dst_fd*, *count*, *offset_src=None*, *offset_dst=None*)
Naïve copy_file_range implementation, with some non-compatibilities

This function does *not* behave exactly like the libc version, as it does not care about the position in the file; it will gladly change it no matter the values of offset_*.

Parameters

- **src_fd** (*int*) – Source/in file descriptor
- **dst_fd** (*int*) – Destination/out file descriptor
- **offset_src** (*int or None*) – Offset to seek to in source fd, or None to run from the current position
- **offset_dst** (*int or None*) – Offset to seek to in destination fd, or None to run from the current position

Returns The amount copied, or a negative value on error

3.2 simplediskimage.common module

Common parts shared between the simple disk image modules

exception simplediskimage.common.**CheckFailed**
Bases: *simplediskimage.common.DiskImageException*

A check has failed

exception simplediskimage.common.**DiskImageException**
Bases: Exception

A generic DiskImage error

exception simplediskimage.common.**InvalidArguments**
Bases: *simplediskimage.common.DiskImageException*

Invalid arguments was passed

class simplediskimage.common.**SI**
Bases: object

Helper class with some constants for calculating sizes in bytes.

G = 1000000000

Gi = 1073741824

M = 1000000

Mi = 1048576

T = 1000000000000

Ti = 1099511627776

k = 1000

ki = 1024

exception simplediskimage.common.**UnknownError**
Bases: *simplediskimage.common.DiskImageException*

Unknown error, probably related to an underlying library

3.3 simplediskimage.partitioners module

Abstraction of different partitioning tools used by simple disk image

class simplediskimage.partitioners.**NullPartitioner**(*image_path*, *table_type*)
Bases: *simplediskimage.partitioners.Partitioner*

Null partitioner abstraction, only allows for one partition

commit()
Commit the partition table to the image

new_partition(*offset_blocks*, *size_blocks*, *filesystem*, *label=None*, *flags=()*)
Create new partition

Parameters

- **offset_blocks** – Offset for the new partition in blocks (sectors)

- **size_blocks** – Size of the new partition in blocks (sectors)
- **filesystem** – Filesystem of the new partition
- **label** – Partition label of the new partition, only for GPT
- **flags** – Flags for the new partition

class simplediskimage.partitioners.**Partitioner**(*image_path*, *table_type*)
Bases: object

Partitioner abstraction class

Parameters

- **image_path** – Path to the image to be created
- **table_type** – Partition table type (label type), 'gpt' or 'msdos' (or 'null' when using the NullPartitioner)

commit()
Commit the partition table to the image

new_partition(*offset_blocks*, *size_blocks*, *filesystem*, *label=None*, *flags=()*)
Create new partition

Parameters

- **offset_blocks** – Offset for the new partition in blocks (sectors)
- **size_blocks** – Size of the new partition in blocks (sectors)
- **filesystem** – Filesystem of the new partition
- **label** – Partition label of the new partition, only for GPT
- **flags** – Flags for the new partition

exception simplediskimage.partitioners.**PartitionerException**
Bases: *simplediskimage.common.DiskImageException*

Generic partitioner error

class simplediskimage.partitioners.**PyParted**(*image_path*, *table_type*)
Bases: *simplediskimage.partitioners.Partitioner*

PyParted partitioner abstraction

commit()
Commit the partition table to the image

new_partition(*offset_blocks*, *size_blocks*, *filesystem*, *label=None*, *flags=()*)
Create new partition

Parameters

- **offset_blocks** – Offset for the new partition in blocks (sectors)
- **size_blocks** – Size of the new partition in blocks (sectors)
- **filesystem** – Filesystem of the new partition
- **label** – Partition label of the new partition, only for GPT
- **flags** – Flags for the new partition

exception simplediskimage.partitioners.**PyPartedException**
Bases: *simplediskimage.partitioners.PartitionerException*

PyParted partitioner error

class simplediskimage.partitioners.**Sfdisk**(*image_path*, *table_type*)
Bases: *simplediskimage.partitioners.Partitioner*

Sfdisk partitioner abstraction

commit()
Commit the partition table to the image

new_partition(*offset_blocks*, *size_blocks*, *filesystem*, *label=None*, *flags=()*)
Create new partition

Parameters

- **offset_blocks** – Offset for the new partition in blocks (sectors)
- **size_blocks** – Size of the new partition in blocks (sectors)
- **filesystem** – Filesystem of the new partition
- **label** – Partition label of the new partition, only for GPT
- **flags** – Flags for the new partition

exception simplediskimage.partitioners.**SfdiskException**
Bases: *simplediskimage.partitioners.PartitionerException*

Sfdisk partitioner error

3.4 simplediskimage.tools module

Tool helpers; used to run commands for the simple disk image package

exception simplediskimage.tools.**DoubleQuoteInExtFile**
Bases: *simplediskimage.common.DiskImageException*

debugfs does not cope well with double quotes in file names, which was detected.

class simplediskimage.tools.**MkfsExt**(*command*)
Bases: *simplediskimage.tools.Tool*

Tool wrapper for mkfs.ext*

mkfs(*device*, *label=None*, *initial_data_root=None*)
Create filesystem

Parameters

- **device** – Device, typically a file in our use case
- **label** – Filesystem label

class simplediskimage.tools.**MkfsExt2**
Bases: *simplediskimage.tools.MkfsExt*

Tool wrapper for mkfs.ext2

class simplediskimage.tools.**MkfsExt3**
Bases: *simplediskimage.tools.MkfsExt*

Tool wrapper for mkfs.ext3

class simplediskimage.tools.**MkfsExt4**
Bases: *simplediskimage.tools.MkfsExt*

Tool wrapper for mkfs.ext4

class simplediskimage.tools.**MkfsFAT**(*fat_size*)
Bases: *simplediskimage.tools.Tool*

Tool wrapper for mkfs.fat

mkfs(*device*, *label=None*, *initial_data_root=None*)
Create filesystem

Parameters

- **device** – Device, typically a file in our use case
- **label** – Filesystem label

class simplediskimage.tools.**MkfsFAT12**
Bases: *simplediskimage.tools.MkfsFAT*

Tool wrapper for mkfs.fat -F 12

class simplediskimage.tools.**MkfsFAT16**
Bases: *simplediskimage.tools.MkfsFAT*

Tool wrapper for mkfs.fat -F 16

class simplediskimage.tools.**MkfsFAT32**
Bases: *simplediskimage.tools.MkfsFAT*

Tool wrapper for mkfs.fat -F 32

class simplediskimage.tools.**PopulateExt**
Bases: *simplediskimage.tools.Tool*

Tool wrapper for debugfs

run(*device*, *actions*)
Perform the requested actions using the tool.

Parameters

- **device** – Device to perfom the actions on
- **actions** – Actions to perform

class simplediskimage.tools.**PopulateFAT**
Bases: object

tool wrapper for mtools

check()
Check that both wrapped tools are available

run(*device*, *actions*)
Perform the requested actions using the tool.

Parameters

- **device** – Device to perfom the actions on
- **actions** – Actions to perform

class simplediskimage.tools.**Sfdisk**
Bases: *simplediskimage.tools.Tool*

Tool wrapper for sfdisk

class simplediskimage.tools.**Tool**(*command*)
Bases: object

Wrapper class for a runnable tool (command).

Parameters **command** – Command to be run, e.g. *ls*.

call(**args*, ***kwargs*)
Call the tool with the given arguments, and debug-log the output

Parameters

- **args** – Command-line arguments
- **kwargs** – Keyword arguments to pass to subprocess.check_output

check()
Check if the tool is available, i.e. in $path and executable.

exception simplediskimage.tools.**ToolNotFound**
Bases: *simplediskimage.common.DiskImageException*

The tool requested was not found (check $PATH and install all dependencies).

simplediskimage.tools.**get_tool**(*filesystem*, *action*)
Get a tool to perform a certain action on a certain filesystem type.

Parameters

- **filesystem** – Filesystem, e.g. “fat16”
- **action** – Action, e.g. “mkfs” or “populate”

CHAPTER 4

Introduction

When preparing installer images or root filesystems images for embedded devices, the classic way is to create a file, *losetup* it, partition it, run *mkfs*, mount it and copy data into it, unmount it, remember to remove the loopback device, etc. Mostly requiring root privileges and the *CAP_SYSADM* capability.

There are however tools to do most operations in user space, as long as you are willing to do some copying, as the tools do not handle offsets into files that well. It is however and error-prone operation, and out of those that do it by hand: How many never remembers how many LBAs into the disk that the GPT streches, and even when looking it up forgets to account for the extra LBA left for MBR and DOS label compatibility, or the extra GPT at the end of the disk? My estimate is most, since I tend to.

For the most times, you just need to set up a simple disk image with one or two partitions, typically just with FAT and/or ext. This library aims to simplify that task by removing some of the complex choices.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

P

R

S

T

U