NAME

mkimg - utility to make disk images

SYNOPSIS

mkimg [**-H** *heads*] [**-P** *blksz*] [**-S** *secsz*] [**-T** *tracksz*] [**-b** *bootcode*] [**-c** *min_capacity*] [**-C** *max_capacity*] [**--capacity** *capacity*] [**-f** *format*] [**-o** *outfile*] [**-a** *active*] [**-v**] [**-y**] [**-s** *scheme* [**-p** *partition* ...]]
mkimg --formats | **--schemes** | **--version**

DESCRIPTION

The **mkimg** utility creates a disk image from the raw partition contents specified with the *partition* argument(s) and using the partitioning scheme specified with the *scheme* argument. The disk image is written to *stdout* by default or the file specified with the *outfile* argument. The image file is a raw disk image by default, but the format of the image file can be specified with the *format* argument.

The disk image can be made bootable by specifying the scheme-specific boot block contents with the *bootcode* argument and, depending on the scheme, with a boot partition. The contents of such a boot partition is provided like any other partition and the **mkimg** utility does not treat it any differently from other partitions.

Some partitioning schemes need a disk geometry and for those the **mkimg** utility accepts the *tracksz* and *heads* arguments, specifying the number of sectors per track and the number of heads per cylinder (resp.)

Both the logical and physical sector size can be specified and for that the **mkimg** utility accepts the *secsz* and *blksz* arguments. The *secsz* argument is used to specify the logical sector size. This is the sector size reported by a disk when queried for its capacity. Modern disks use a larger sector size internally, referred to as block size by the **mkimg** utility and this can be specified by the *blksz* argument. The **mkimg** utility will use the (physical) block size to determine the start of partitions and to round the size of the disk image.

The **-c** option can be used to specify a minimal capacity for the disk image. Use this option without the **-s** and **-p** options to create an empty disk image with the given (virtual) size. An empty partition table can be written to the disk when specifying a partitioning scheme with the **-s** option, but without specifying any partitions. When the size required for all the partitions is larger than the given capacity, then the disk image will be larger than the capacity given.

The **-C** option specifies a maximum capacity for the disk image. If the combined sizes of the given partitions exceed the size given with **-C**, image creation fails.

The **--capacity** option is a shorthand to specify the minimum and maximum capacity at the same time.

The **-v** option increases the level of output that the **mkimg** utility prints.

The **-y** option is used for testing purposes only and is not to be used in production. When present, the **mkimg** utility will generate predictable values for Universally Unique Identifiers (UUIDs) and time stamps so that consecutive runs of the **mkimg** utility will create images that are identical.

The *active* option marks a partition as active, if the partitioning scheme supports it. Currently, only the *mbr* scheme supports this concept. By default, **mkimg** will only mark the first partition as active when boot code is specified. Use the *active* option to override the active partition. The number specified corresponds to the number after the ’s’ in the partition’s geom(8) name. No partitions are marked active when the value is 0.

A set of long options exist to query about the **mkimg** utility itself. Options in this set should be given by themselves because the **mkimg** utility exits immediately after providing the requested information. The version of the **mkimg** utility is printed when the **--version** option is given. The list of supported output formats is printed when the **--formats** option is given and the list of supported partitioning schemes is printed when the **--schemes** option is given. Both the format and scheme lists a space-separated lists for easy handling in scripts.

For a more descriptive list of supported partitioning schemes or supported output format, or for a detailed description of how to specify partitions, run the **mkimg** utility without any arguments. This will print a usage message with all the necessary details.

DISK FORMATS

The **mkimg** utility supports a number of output file formats. A short description of these is given below.

QCOW and QCOW2

QCOW stands for "QEMU Copy On Write". It’s a sparse file format akin to VHD and VMDK and QCOW represents the first version. QCOW2 represents version 2 of the file format. Version 2 is not backward compatible with version 1 and adds support for snapshots among other things. The QCOW file formats are natively supported by QEMU and Xen. To write QCOW, specify **-f** *qcow* on the command line. To write version 2 QCOW, specify **-f** *qcow2* on the command line. The preferred file extension is ".qcow" and ".qcow2" for QCOW and QCOW2 (resp.), but ".qcow" is sometimes used for version 2 files as well.

RAW file format

This file format is a sector by sector representation of an actual disk. There is no extra information that describes or relates to the format itself. The size of the file is the size of the (virtual) disk. This file format is suitable for being copyied onto a disk with utilities like **dd**. To write a raw disk file, either omit the **-f** option, or specify **-f** *raw* on the command line. The preferred file extension is one of ".img"

or ".raw", but there's no real convention for it.

Dynamic VHD and Fixed VHD

Microsoft's "Virtual Hard Disk" file formats. The dynamic format is a sparse format akin to QCOW and VMDK. The fixed format is effectively a raw format with a footer appended to the file and as such it's often indistinguishable from the raw format. The fixed file format has been added to support Microsoft's Azure platform and due to inconsistencies in interpretation of the footer is not compatible with utilities like **qemu** when it is specifically instructed to interpreted the file as a VHD file. By default **qemu** will treat the file as a raw disk file, which mostly works fine. To have **mkimg** create a dynamic VHD file, specify **-f** *vhd* on the command line. To create a fixed VHD file for use by Azure, specify **-f** *vhdf* on the command line. The preferred file extension is ".vhd".

Dynamic VHDX

Microsoft's "Virtual Hard Disk v2" file formats, the successor to VHD. VHDX is the required format for the 2nd generation Hyper-V VMs. To have **mkimg** create a dynamic VHDX file, specify **-f** *vhdx* on the command line. The preferred file extension is ".vhdx".

VMDK

VMware's "Virtual Machine Disk" file format. It's a sparse file format akin to QCOW and VHD and supported by many virtualization solutions. To create a VMDK file, specify **-f** *vmdk* on the command line. The preferred file extension is ".vmdk".

Not all virtualization solutions support all file formats, but often those virtualization environments have utilities to convert from one format to another. Note however that conversion may require that the virtual disk size is changed to match the constraints of the output format and this may invalidate the contents of the disk image. For example, the GUID Partition Table (GPT) scheme has a header in the last sector on the disk. When changing the disk size, the GPT must be changed so that the last header is moved accordingly. This is typically not part of the conversion process. If possible, use an output format specifically for the environment in which the file is intended to be used.

ENVIRONMENT

TMPDIR Directory to put temporary files in; default is */tmp*.

EXAMPLES

To create a bootable disk image that is partitioned using the GPT scheme and containing a root file system that was previously created using makefs(8) and also containing a swap partition, run the **mkimg** utility as follows:

```
% mkimg -s gpt -b /boot/pmbr -p freebsd-boot:=/boot/gptboot -p freebsd-ufs:=root-file-system.ufs -p freebsd-swap::1G -o gpt.img
```

The command line given above results in a raw image file. This is because no output format was given. To create a VMDK image for example, add the **-f** *vmdk* argument to the **mkimg** utility and name the output file accordingly.

A nested partitioning scheme is created by running the **mkimg** utility twice. The output of the first will be fed as the contents of a partition to the second. This can be done using a temporary file, like so:

```
% mkimg -s bsd -b /boot/boot -p freebsd-ufs:=root-file-system.ufs -p freebsd-swap::1G -o
/tmp/bsd.img
% mkimg -s mbr -b /boot/mbr -p freebsd:=/tmp/bsd.img -o mbr-bsd.img
```

Alternatively, the **mkimg** utility can be run in a cascaded fashion, whereby the output of the first is fed directly into the second. To do this, run the **mkimg** utility as follows:

```
% mkimg -s mbr -b /boot/mbr -p freebsd:-'mkimg -s bsd -b /boot/boot -p
freebsd-ufs:=root-file-system.ufs -p freebsd-swap::1G' -o mbr-bsd.img
```

To accommodate the need to have partitions named or numbered in a certain way, the **mkimg** utility allows for the specification of empty partitions. For example, to create an image that is compatible with partition layouts found in */etc/disktab*, the 'd' partition often needs to be skipped. This is accomplished by inserting an unused partition after the first 2 partition specifications. It is worth noting at this time that the BSD scheme will automatically skip the 'c' partition by virtue of it referring to the entire disk. To create an image that is compatible with the qp120at disk, use the **mkimg** utility as follows:

```
% mkimg -s bsd -b /boot/boot -p freebsd-ufs:=root-file-system.ufs -p freebsd-swap::20M -p- -p- -p-
-p- -p freebsd-ufs:=usr-file-system.ufs -o bsd.img
```

For partitioning schemes that feature partition labels, the **mkimg** utility supports assigning labels to the partitions specified. In the following example the file system partition is labeled as 'backup':

```
% mkimg -s gpt -p freebsd-ufs/backup:=file-system.ufs -o gpt.img
```

SEE ALSO

dd(1), gpart(8), makefs(8), mdconfig(8), newfs(8)

HISTORY

The **mkimg** utility first appeared in FreeBSD 10.1.

AUTHORS

The **mkimg** utility and manpage were written by Marcel Moolenaar <*marcel@FreeBSD.org*>.