OE-lite Handbook Documentation

Release 0.1

Kim Højgaard-Hansen

December 29, 2016

Contents

1 Host Setup **1**
- 1.1 Requirements . . . 1
- 1.2 Install Bakery . . . 1
- 1.3 Install Manifest Dependencies . . . 3
- 1.4 Goodbye dash . . . 3

2 Project Setup **5**
- 2.1 From Scratch . . . 5
- 2.2 From Template . . . 7
- 2.3 Repository Setup . . . 7

3 Cloning **9**

4 Building **11**

5 Source Mirrors **13**
- 5.1 Prerequisites . . . 13
- 5.2 The mirror class . . . 13
- 5.3 Creating and maintaining a source mirror . . . 13
- 5.4 Synchronizing mirror directory to remote server . . . 14
- 5.5 Build from source mirror . . . 14

6 Release Management **15**
- 6.1 Metadata Releases . . . 15
- 6.2 Release Cherry-Picking . . . 17
- 6.3 BSP Versioning . . . 18
- 6.4 Branching and Tagging . . . 18

7 Working with Upstream **19**
- 7.1 Mailing Lists . . . 19
- 7.2 Commit Message Guidelines . . . 20
- 7.3 Submitting Changes . . . 21

8 Recipes **23**
- 8.1 Naming conventions . . . 23
- 8.2 Language . . . 23
- 8.3 include and require . . . 24
- 8.4 Syntax . . . 24

9 Examples of recipes **25**
9.1 Trigonometric utilities . . . 25
9.2 Dissection of an existing recipe . . . 31

10 Tasks **33**
10.1 Environment . . . 33
10.2 Metadata . . . 34
10.3 Task types . . . 34

11 OE-lite Terminology **39**

12 Syntax **41**
12.1 Formal grammar . . . 41
12.2 Semantics . . . 43

13 Attribution-ShareAlike 3.0 Unported **47**
13.1 License . . . 47
13.2 1. Definitions . . . 47
13.3 2. Fair Dealing Rights . . . 48
13.4 3. License Grant . . . 49
13.5 4. Restrictions . . . 49
13.6 5. Representations, Warranties and Disclaimer . . . 50
13.7 6. Limitation on Liability . . . 51
13.8 7. Termination . . . 51
13.9 8. Miscellaneous . . . 51

14 Preface **53**
14.1 License . . . 53
14.2 Indices and tables . . . 53

CHAPTER 1

Host Setup

This chapter describes how to setup a host machine (your development PC, server or whatever) for working with OE-lite development.

1.1 Requirements

The only officially supported host OS is Linux, but at least one developer is also using Mac OS X with some luck.

1.2 Install Bakery

To do any kind of OE-lite development, you need to have the OE-lite Bakery tool installed. OE-lite Bakery depends on Python (2.6 or 2.7) and git.

There are several ways of installing OE-lite Bakery on your host machine.

1. Install from source.
2. Run from source.
3. Install from Ubuntu PPA (Ubuntu based distributions only).
4. Install on Exherbo Linux.

If installing or running from source, you need to install additional required host software tools manually. If installing from a host OS package, the package should pull in the required software automatically.

1.2.1 Install Additional Tools

In order to run OE-lite Bakery, you need a few additional software packages which you might not have installed. This is currently limited to Git and PLY.

Git

OE-lite Bakery uses the git command when fetching OE-lite manifests.

The easiest way is most likely to simply install the git command provided by your host OS.

Debian GNU/Linux, Ubuntu Linux, . . .

```
sudo apt-get install -y git
```

PLY (Python Lex-Yacc)

OE-lite Bakery uses PLY for parsing configuration files.

The easiest way is most likely to simply install the PLY version provided by your host OS.

Debian GNU/Linux, Ubuntu Linux, . . .

```
sudo apt-get install -y python-ply
```

1.2.2 Install from source

To install OE-lite Bakery, you need to have Python setuptools installed.

To install Python setuptools on Debian GNU/Linux, Ubuntu Linux and so on, use

```
sudo apt-get install -y python-setuptools
```

After that, you should be able to install it with the following command:

```
wget -qO- http://oe-lite.org/download/bakery/oe-lite-bakery-4.0.2.tar.gz \
        | tar -xz \
        && sudo oe-lite-bakery-4.0.2/setup.py install
```

1.2.3 Run from source

OE-lite Bakery also supports running directly from source distribution.

Download and extract the latest release from http://oe-lite.org/download/bakery/

```
wget -qO- http://oe-lite.org/download/bakery/oe-lite-bakery-4.0.2.tar.gz \
        | tar -xz
```

or clone the bakery repository with

```
git clone git://oe-lite.org/oe-lite/bakery.git
```

You can use the oebakery/oe.py script directly, but you should probably symlink it to “oe” somewhere in your $PATH or setup a shell alias so you can just type “oe” when using bakery.

Something like

```
ln -s ../src/bakery/oebakery/oe.py $HOME/bin/oe
```

(assuming you have the bakery source distribution in $HOME/src/bakery and have $HOME/bin in your $PATH)

1.2.4 Install from Ubuntu PPA

This method is only for use on Ubuntu Linux or distributions compatible with Ubuntu Linux (like Mint).

To install bakery from the PPA, you can use the following commands:

```
sudo apt-get install -y python-software-properties
sudo add-apt-repository ppa:esben-haabendal/oe-lite
sudo apt-get update
sudo apt-get install -y oe-lite
```

1.2.5 Install on Exherbo Linux

Since Exherbo is a source based distribution, most dependencies are installed already. The rest is pulled in by the oe-bakery package.

```
sudo cave resolve oe-bakery
```

1.3 Install Manifest Dependencies

Depending on the *OE-lite manifest(s)* you will be working with, and what you will build with it, you will require some additional host tools. If you installed bakery from PPA, you most likely already have all you need, and you can skip this section.

If you installed bakery in another way, you might want to install some additional development tools.

Installing additional development tools in Fedora 16 (and possibly other RPM based distributions):

```
sudo yum install python-magic python-ply python-pycurl \
python-sqlite2 python-devel fakeroot libstdc++-static glibc-static \
gettext-devel ncurses-devel libtool texinfo flex bison coreutils \
sed git-core cvs subversion mercurial quilt gawk texinfo automake \
autoconf curl texi2html openjade groff make gcc-c++ gcc binutils bc \
unzip lzma gtk-doc docbook-utils xml2 xmlto help2man glib2-devel gperf
```

Install additional development tools in Debian GNU/Linux, Ubuntu Linux and so on, something like:

```
sudo apt-get install python python-support python-magic python-ply \
python-pycurl python-pysqlite2 python-pkg-resources python-dev \
coreutils sed git-core cvs subversion mercurial quilt gawk texinfo \
automake autoconf autopoint libtool curl texi2html diffstat \
openjade groff mtd-utils build-essential make gcc g++ binutils \
bison flex bc ncurses-dev unzip lzma gtk-doc-tools docbook-utils \
libxml2-utils xmlto help2man libglib2.0-dev lzop gperf python-svn
```

Install additional development tools in RHEL 6.2, something like:

```
sudo yum install python-magic python-ply python-pycurl python-devel \
fakeroot gettext-devel ncurses-devel libtool texinfo flex bison \
coreutils sed git-core cvs subversion mercurial quilt gawk texinfo \
automake autoconf curl openjade groff make gcc-c++ gcc binutils bc \
unzip gtk-doc docbook-utils xmlto glib2-devel intltool glibc-static \
gperf
```

1.4 Goodbye dash

On some systems (fx. Ubuntu Linux), `/bin/sh` is a symlink to dash, which not all software packages are fully compatible with. To work with OE-lite, you therefore have to make sure that `/bin/sh` is actually `/bin/bash`.

You can do this the brute force way

```
sudo ln -sf bash /bin/sh
```

Or on Ubuntu Linux, you can do this more nicely with

```
sudo dpkg-reconfigure dash
```

and answer “No” to the “Use dash as the default system shell (/bin/sh)?” question.

CHAPTER 2

Project Setup

This chapter describes how to setup a new OE-lite project, ie. the creation of a new *OE-lite manifest* and setup of an *OE-lite repository* for it.

2.1 From Scratch

To create a new *OE-lite manifest* from scratch, all you need to do is:

1. Create an empty directory.
2. Create a conf/bakery.conf file.
3. Run `oe init`.
4. Convert *layers* to be of internal layer type.

2.1.1 Bakery.conf from scratch

The bakery.conf follows the OE-lite metadata syntax, or rather a subset of it. The primary purpose is to assign a value to the variable called `OESTACK`, which defines the *OE-lite stack* .

An OE-lite stack is composed of a number of OE-lite layers, with each layer typically being a seperate git repository.

A small OE-lite stack could look like this:

```
# OE-lite/base
OESTACK += "meta/base"
OESTACK .= ";srcuri=git://oe-lite.org/oe-lite/base.git"
OESTACK .= ";branch=master"

# OE-lite/core
OESTACK += "meta/core"
OESTACK .= ";srcuri=git://oe-lite.org/oe-lite/core.git"
OESTACK .= ";branch=master"

OESTACK += "lib/fetching/fetching"
OESTACK .= ";srcuri=git://oe-lite.org/python-fetching.git"
OESTACK .= ";pythonpath=.."

OESTACK += "lib/GitPython"
OESTACK .= ";srcuri=git://oe-lite.org/gitpython/GitPython.git"
OESTACK .= ";pythonpath="
```

```
OESTACK += "lib/urlgrabber"
OESTACK .= ";srcuri=git://oe-lite.org/urlgrabber.git"
OESTACK .= ";pythonpath="
```

The example above uses two different append assignment operators: “+=” and ”.=”. The “+=” operator adds an extra space before appending the value whereas the ”.=” operator just appends the value. The two expressions:

HELLO += “world” HELLO .= ” world”

are the same.

The resulting `OESTACK` variable is thus a space separated list of layers. Each layer is specified by a path and a number of parameters, separated by ”;”.

Note

Add reference to the OE-lite Bakery Manual for full documentation on the bakery.conf syntax here, when it is actually written...

After the `oe init` command is done, the my-bsp directory should be populated with the following structure:

```
-- conf
|   -- bakery.conf
-- lib
|   -- fetching
|   -- GitPython
|   -- urlgrabber
-- meta
    -- base
    -- core
```

and all the layers should be cloned from their upstream origin.

Example (for the copy-and-paste hungry):

```
mkdir my-bsp
cd my-bsp
mkdir conf
emacs conf/bakery.conf
oe init
```

At this point, you should create the initial git commit of your brand new OE-lite manifest:

```
git add conf/bakery.conf
git commit -s -m "Initial commit"
```

You are now (almost) ready to build something. To try this, see chapter *Building* for how to build.

Of-course, you might want to add some more metadata layers, and probably add your own machine and/or distro configurations and even some custom recipes, fx. a recipe for building a custom rootfs image. But that is a different story...

2.1.2 External Layers

Let’s say you are creating an OE-lite manifest for your embedded Linux BSP project. You of-course need to use OE-lite/core, and the simplest solution is to just add it to the STACK by adding the following to bakery.conf:

```
OESTACK += "meta/core"
OESTACK .= ";srcuri=git://oe-lite.org/oe-lite/core.git"
```

With this, users of your manifest will get an OE-lite/core layer at meta/core, using a clone from the git://oe-lite.org/oe-lite/core.git repository.

While this is definitely a lean and simple approach, it does come with a few drawbacks.

1. You will not be able to create any commits, tags or branches to the OE-lite/core layer.
2. When cloning the OE-lite repository, you depend on both the server hosting the manifest repository and the oe-lite.org server.

See also appendix *OE-lite Terminology* for definition of internal layer.

2.1.3 Internal Layers

For each layer you have added to the OE-lite stack as an external layer, you should consider to convert it to be an internal layer to address the problems with external layers described above. See appendix *OE-lite Terminology* for definition of internal layer.

By converting all external layers to internal layers, and thus having a manifest consisting of only embedded and internal layers, you will have a number of advantages:

1. When creating a clone of the OE-lite repository, you will only have to fetch from your project OE-lite repository.
2. You will be able to create backup/redundant copies of your entire OE-lite repository using a single command.
3. You will be able to switch back and forth between different copies of your OE-lite repository without making any changes to the OE-lite manifest.
4. You will be able to make complete from local clones of your OE-lite repository, without depending on any remote repositories.

For each layer you want to convert from external layer to internal layer, you have to do the following:

1. Remove the `srcuri` parameter for the layer in conf/bakery.conf
2. Change the url entry of the layer submodule in .gitmodules to the path relative to the containing git super project. Fx. the relative path of meta/core contained in the manifest repository is ./meta/core, and the relative path of lib/GitPython/git/ext/async contained in the lib/GitPython submodule is ./git/ext/async .

When done, run `oe update` and commit the changes in conf/bakery.conf and .gitmodules files.

2.2 From Template

TBD...

2.3 Repository Setup

This section describes how to setup an OE-lite repository, suitable for hosting as a remote repository. Details on how to setup hosting is out of scope of this section.

To setup an OE-lite repository of an existing OE-lite manifest, all you need to do is to call:

```
oe clone --bare <url> <path>
```

Note

OE-lite Bakery version 4.1 or newer is required for this.

This will create a new (bare) OE-lite repository clone of <url> at the local directory <path>. The `<url>` argument can be any valid git URL (see link:See git[git clone documentation] for more on this). This even includes a local path to an OE-lite manifest repository, which is handy for setting up the first OE-lite repository right after creation of a new OE-lite manifest.

All internal layers will be cloned (recursively) together with the manifest repository. Any other git submodules (ie. git submodules with absolute url's or relative paths different from the path relative to the git super project) will not be cloned.

CHAPTER 3

Cloning

This chapter describes how to clone an existing OE-lite repository.

To create a local clone of an OE-lite repository for development and/or build purposes, use the following command:

```
oe clone <url> <path>
```

This will create a new OE-lite repository clone of <url> at the local directory <path>. The <url> argument can be any valid git url. See link:See git[git clone documentation] for more on this.

All git submodules and/or OE-lite layers specified will be (recursively) cloned also.

CHAPTER 4

Building

This chapter describes how to build something with OE-lite, fx. how to build a specific OE-lite recipe, a Linux kernel image, a JFFS2 root filesystem image, an SDK toolchain image, and so on.

Building is done with the OE-lite Bakery sub-command called "bake".

Before building you need to setup the build configuration in the file `conf/local.conf`.

A very minimal example configuration purely to test that building works:

```
DISTRO = "base"
MACHINE_CPU = "arm-926ejs"
PROVIDED = "all"
SDK_CPU = "i686"
SDK_OS = "linux-gnu"
RMWORK = "0"
```

The `DISTRO` variable selects the OE-lite distribution. Here we choose a simple distribution called base to be able to build something. Next we set the cpu we want to cross compile for using `MACHINE_CPU`. It is also possible to set `MACHINE` to target a specific board e.g. pandaboard or rpi (raspberry-pi).

Note

To set `MACHINE="rpi"` you will need the raspberry-pi manifest from git.oe-lite.org.

The `PROVIDED` variable is used to inform the bake command what dependencies can be assumed to be provided on the host system. See `conf/provided/all.conf` in the core metadata layer. The `SDK` variables are used to specify what architecture the OE-lite SDK should be build for. `RMWORK` currently need to be set to 0 since automatic removal of temporary build files is not implemented. Optionally you may want to set `PARALLEL_MAKE = "-j X"` where X is the number of CPUs available on your host system + 1, to speed up the build.

Now it is possible to choose something to build with the bake command. In OE-lite all *recipes* can be build. A recipe is a file with the `.oe` file extension, take a look at what recipes you have in your current manifest using:

```
find . -name '*.oe'
```

The primary goal of the building process in OE-lite is to produce deployable images, so for this example we will build an image. In the base metadata layer a rootfs image recipe is located in: `recipes/images` which we can try building:

```
oe bake base-rootfs
```

`oe` will resolve the list of dependencies, present you with a list of what needs to be built and ask for confirmation before continuing. The build process takes a while, but in the end you should see that base-rootfs was build and the elapsed build time. The deployable images are now located in `tmp/images`

CHAPTER 5

Source Mirrors

OE-lite provides means for setting up and maintaining mirrors of external sources used in builds.

This is useful for ensuring that OE-lite builds are not affected by 3rd party sources being removed from their original location on the Internet. It can also be used for redirecting all fetches from the Internet to a local server (or filesystem), making OE-lite work in closed networks or without any network connection at all.

5.1 Prerequisites

To setup an OE-lite source mirror, you must use OE-lite/core 3.2.0 or newer.

The filesystem path to the source mirror must be specified, fx. in your `local.conf` file. Fx.

```
MIRRORDIR = "/local/mirror"
```

5.2 The mirror class

An OE-lite source mirror is both created and maintained with the help of the OE-lite `mirror` task. The `mirror` task will add any sources required by the recipe which is not already present in the mirror, and will fail if a file conflict occurs.

There is also an `mirrorall` task, which recursively causes the `mirror` task to be run for all recipes that the recipe depends on (both build and run-time dependencies).

The `mirror` and `mirrorall` tasks are implemented in the `classes/mirror.oeclass` file in OE-lite/core.

The `MIRRORDIR` directory will be created if it does not exist.

5.3 Creating and maintaining a source mirror

As the source mirror is maintained with the help of the `mirror` and `mirrorall` tasks, you need to configure your `local.conf` for the `MACHINE` and `SDK` configuration options required for your mirror.

For each configuration, you must run the `mirrorall` task on the required top-level recipes (or world or universe, if you want really everything in your mirror).

Fx.

```
oe bake -t mirrorall kernel my-rootfs my-sdk
```

or

```
oe bake -t mirrorall universe
```

After having run this on all the required configurations, your source mirror in `MIRRORDIR` will be up-to-date with all sources required for the current OE-lite checkout and the configurations used.

5.4 Synchronizing mirror directory to remote server

The mirror will be created (and maintained) in a directory on the machine running OE-lite. To maintain a mirror on a remote server, you could use the rsync command (requires ssh login access to server, and rsync command on the server):

The following command will synchronize the /local/mirror directory to the /mirror directory on the server (myserver):

```
rsync -av /local/mirror/ myserver:/mirror/
```

5.5 Build from source mirror

To use the OE-lite source mirror, the `MIRRORS` and `PREMIRRORS` variables must be modified.

If you only want to fall-back to using the source mirror, and thus always try to fetch from the original source first, you should change the `MIRRORS` variable to something like

```
MIRRORS = """
http://.*/ http://myserver/mirror/${INGREDIENTS_SUBDIR}/
ftp://.*/  http://myserver/mirror/${INGREDIENTS_SUBDIR}/
git://.*/  http://myserver/mirror/${INGREDIENTS_SUBDIR}/git//
svn://.*/  http://myserver/mirror/${INGREDIENTS_SUBDIR}/svn//
"""
```

If you want to use the source mirror first, and only fall-back to trying the orignial source in case fetching from the source mirror fails, you should change the `PREMIRRORS` variable to something like

```
PREMIRRORS = """
http://.*/ http://myserver/mirror/${INGREDIENTS_SUBDIR}/
ftp://.*/  http://myserver/mirror/${INGREDIENTS_SUBDIR}/
git://.*/  http://myserver/mirror/${INGREDIENTS_SUBDIR}/git//
svn://.*/  http://myserver/mirror/${INGREDIENTS_SUBDIR}/svn//
"""
```

Both of the configurations above assumes that your "myserver" is running an HTTP server, and is hosting the `/mirror` directory on the http://myserver/mirror URL.

If you only want to allow fetching from the source mirror, and thus forbid fetching from any other remote server, you can use the `URL_WHITELIST` variable. In addition to the `MIRRORS` and/or `PREMIRRORS` variables, you could add something like

```
URL_WHITELIST = "http://myserver/mirror"
```

CHAPTER 6

Release Management

This chapter covers the various aspects related to making releases of OE-lite based projects.

Some parts of this chapter is meant as a guideline and not something that you need to follow.

6.1 Metadata Releases

This section describes how to make releases of OE-lite metadata projects (like OE-lite/core, OE-lite/base and so on).

6.1.1 Metadata Versioning

For OE-lite.org metadata projects, the releases must be versioned according to the scheme described in this section.

OE-lite.org metadata releases must follow the Semantic Versioning specification (see http://semver.org). Briefly described, this means that version numbers are formatted as X.Y.X, with X being major number, Y being minor number, and Z being patch number.

For releases that only contains backwards compatible bugfixes (a bugfix release) should be versioned with an increment to the patch number. A bugfix release based on X.Y.Z would thus be X.Y.Z+1.

For release that contains new, backwards compatible functionality (feature releases) should be versioned with an increment to the minor number. A feature release based on X.Y.Z would thus be X.Y+1.0.

For releases that contains any backwards incompatile changes (major releases) should be versioned with an increment to the major number. A major release based on X.Y.Z would thus be X+1.0.0.

For more details, see http://semver.org

6.1.2 Metadata Release Branching

OE-lite.org metadata releases should be done from a release branch named X.Y (for release version X.Y.Z).

When creating a new major release, a new release branch must be created. This new X.0 branch should branch off of either the previous latest release branch (ie. X-1.Y) or the master branch.

When creating a new feature release, a new release branch must be created. This new X.Y branch should branch off of the previous release branch (X.Y-1).

When creating a new bugfix release, the X.Y release branch should already exist. It should have been created when the X.Y.0 feature release (or major release if Y=0) was made.

Release branchs must be pushed to the official OE-lite.org upstream repository (ie. git://oe-lite.org/oe-lite/core.git for OE-lite/core). Release branches are considered permanent branches, and should not be deleted, as they must be available for doing bugfix releases from.

Important

Public release branches must not be rebased, or the commit history in any other way be rewritten.

6.1.3 Metadata Release Tagging

When a release is ready to go out of the door, it must be tagged.

OE-lite.org metadata project releases must contain a `VERSION` file containing the release version number in plain text. So before making the git tag, a new commit with this file should be created.

The following example shows how to create a release commit and tag:

```
echo "3.4.1" > VERSION
git add VERSION
git commit -m "Release 3.4.1"
git tag -a -m "Release 3.4.1" v3.4.1
```

After the release is done, the `VERSION` file should be removed, so that only the actual release version carries it.

```
git rm VERSION
git commit -m "Unrelease"
```

The release branch (including both the release and unrelease commit) and the release tag must of-course be pushed to the official OE-lite.org upstream repository (ie. git://oe-lite.org/oe-lite/core.git for OE-lite/core).

Important

Release tags must not be changed.

Note

The release and unrelease commits does not need a Signed-off-by line.

6.1.4 Metadata Release Tarballs

OE-lite.org metadata project releases must be available as tarball for download on http://oe-lite.org/download/

To create release tarballs, use something like the following:

```
git archive --prefix=core-3.4.1/ -o oe-lite-core-3.4.1.tar v3.4.1
cat oe-lite-core-3.4.1.tar | gzip > oe-lite-core-3.4.1.tar.gz
cat oe-lite-core-3.4.1.tar | xz > oe-lite-core-3.4.1.tar.xz
```

To put the tarballs on oe-lite.org, stuff them somewhere on the net and send an email to esben@haabendal.dk (with cc to dev@oe-lite.org) requesting copies to be placed on the oe-lite.org server.

6.1.5 Metadata Release Announcement

When the OE-lite.org metadata project release is ready (ie. tarballs are on oe-lite.org, and the release has been pushed to the official oe-lite.org repository, the release must be announced to the OE-lite.org community.

The release must be announced both on the dev@oe-lite.org mailing list and the http://oe-lite.org site.

Metadata Release Email Announcement

The release announcement email could look something like http://lists.oe-lite.org/pipermail/dev/2012-November/001222.html.

To generate the contributer contribution and the per-author shortlog text, you can use the http://oe-lite.org/download/scripts/release-mail.py script. It should be called like this:

```
release-mail.py v3.3.0 v3.4.0
```

With the first argument specifying the previous release, and the second argument specifying the release you are announcing.

Metadata Relase Redmine Announcement

To announce the release on http://oe-lite.org, you must create a Redmine news item, and it could look something like http://oe-lite.org/redmine/news/11.

6.1.6 Metadata Release Checklist

1. Is the release created from a release branch according to the description in section *Metadata Release Branching*?
2. Is the release properly tagged according to the description in section *Metadata Release Tagging*?
3. Has tar-balls been created and uploaded to oe-lite.org according to the description in section *Metadata Release Tarballs*?
4. Has a release announcement mail been sent to the dev@oe-lite.org mailinglist according to the description in *Metadata Release Announcement*?
5. Has the http://oe-lite.org Redmine been updated with a News item according to the description in *Metadata Release Announcement*?

6.2 Release Cherry-Picking

This section describes how to use the `oe cherry` command for assistance in cherry picking commits to release branches.

To use the cherry command, you need OE-lite Bakery 4.0.0 or newer, and OE-lite/core 3.3.0 or newer.

The idea with the cherry command is to help you keep track of which commits eligible for a specific release branch.

You can fx. use the cherry command to find out which commits on the master branch are eligible for being cherry picked to the 3.4 release branch with the following command:

```
oe cherry master 3.4
```

This will list all commits that are currently seen as eligible for the 3.4 release branch.

To remove commits from this list, you can run cherry in interactive mode:

```
oe cherry -i master 3.4
```

For each commit, you will be asked for the target version. The allowed values are:

1. A release branch, ie. X.Y. Commits that you see as eligible for release branch X.Y (and newer) should be marked with target version X.Y (fx. "3.4", for release branch 3.4).

2. A major release version, ie. X. Commits that you see as eligible for a (most likely yet-to-come) major release, should be marked with target version X (fx. "4" for major release 4)
3. The master branch. Commits that is not eligible for any releases, and thus should stay on the master branch should be marked with target version "master".

Any target versions you set will be stored in your local git repository, and will be used the next time you use the cherry command.

When you have trimmed down the list, you should cherry pick the commits to the release branch you are working with.

> **Note**
>
> Remember to use the "-x" argument with the `git cherry-pick` command, as it will help `oe cherry` in determining if a commit has already been cherry-picked.

6.3 BSP Versioning

For OE-lite.org BSP projects, the releases must be versioned according to the scheme described in this section.

An OE-lite.org BSP is specified by a version number, and an optional release name. Notice that the version number is mandatory and must by itself specify the release. The release name is optional and only meant as a possibility of adding a short description (or perhaps for adding a funny name...).

Currently, there is no rules or guidelines for the numbering scheme. Suggestions and discussion related to this are welcome at dev@oe-lite.org :-)

6.4 Branching and Tagging

TBD...

CHAPTER 7

Working with Upstream

This chapter describes how to work with the upstream OE-lite.org project.

7.1 Mailing Lists

Development and coordination of the OE-lite.org project is managed on the dev@oe-lite.org mailing list.

Use this for getting in contact with OE-lite.org developers and discussion or coordination of OE-lite development.

7.1.1 Subscribing

To subscribe, you can either use the web interface or the email interface.

1. Go to the list information page (http://lists.oe-lite.org/mailman/listinfo/dev).
2. Look for the section marked "Subscribing to dev" and fill in the boxes. You can fill in the following:
    - You must enter your email address.
    - You may choose to supply your real name.
    - You may choose a password. If you do not choose one, Mailman will generate one for you. WARNING: Do NOT use a valuable password, since this password may be mailed to you in plain text.
    - If the list supports more than one language, you may be able to choose your preferred language. NOTE: This setting does not affect posts to the list, only pre-prepared Mailman texts such as your member options page.
3. Press the subscribe button. A new page should appear telling you that your request has been sent.

1. Open a mail program which sends mail from the address you want to subscribe.
2. Send a mail to dev-join@oe-lite.org. The subject and body of the message will be ignored, so it doesn't matter what you put there.

You may receive an email message asking for confirmation that you really want to be subscribed to the list. This is to prevent anyone from subscribing you to lists without your permission. Follow the instructions given in the message to confirm your wish to be subscribed.

Once this is done, you will receive another message welcoming you to the list. This message contains some useful information including your list password and some quick links for changing your options, so you may want to save it for later reference.

7.1.2 Unsubscribing

If you want to leave the list, there are two common ways you can unsubscribe.

1. Go to the list information page (http://lists.oe-lite.org/mailman/listinfo/dev).
2. Look for the section marked "dev subscribers" (near the bottom of the page).
3. There should be a button marked "Unsubscribe or Edit Options." Enter your email address in the box beside this button and press the button.
4. You should be brought to a new page which has an "Unsubscribe" button. Press it to unsubscribe and follow the instructions given.

1. Open a mail program which sends mail from the address you want to unsubscribe.
2. Send a mail to dev-leave@oe-lite.org. The subject and body of this message will be ignored, so it doesn't matter what you put there.

7.2 Commit Message Guidelines

The following guidelines should be followed when writing log messages for commits to be included in OE-lite.org repositories:

- The first line should be a short description, prefixed what is being changed.
  - The prefix should be one of
    - Name of subdir containing recipes, fx. *linux*, for changes to recipes within that directory.
    - Name of recipe, fx. *linux-yocto*, for changes to that specific recipe.
    - Name of OE-lite class (.bbclass file) prefixed with *class*, fx. *class/cross*, for changes to that specific class.
    - Name of distro configuration file prefixed with *distro/*, fx. *distro/base*, for changes to that distro configuration.
    - Name of machine configuraton file prefixed with *machine/*, fx. *machine/beagleboard*, for changes to that machine configuration.
    - Name of subdir containing Python library code prefixed with *lib/*, fx. *lib/oelite*, for changes to Python files in that dir.
    - Name of configuration file (in conf/ directory) prefixed with *conf/*, fx. *conf/package*, for changes to that file (package.conf).
  - The description should be very short (a few words), summarising the change.
  - The prefix and description is separated by a space followed by a colon (ie. ': ').
- The first line should be followed by an empty line
- Additional lines, may follow describing more details of the change.
- The details if feasible, should be structured as a list, with each item marked with a start *,
- Signed-off-by (SOB) lines are currently not required in OE-lite.
- Lines must not be longer than 75 characters.

The first line of commit log messages are very important, as it in some cases will be the the only description of the change being presented, fx. in subject lines of mails being created with git-format-patch.

These guidelines should be followed, but may be combined with common sense for doing things different when it makes sense.

Example: Adding a recipe named dhcp3

```
dhcp3: New recipe, for ISC DHCP version 3

In the newer versions of dhcp (4.X), bind is included statically linked in
dhcp, making some dhcp files unneccessary big.  Therefore, an older version
of dhcp is often desired.
```

7.3 Submitting Changes

When making changes to OE-lite metadata layers originating from OE-lite.org, you should make an effort to get your changes merged to OE-lite.org. By doing this, you will reduce future maintenance effort, as you don't have to take care with maintaining that particular change/feature anymore.

7.3.1 Pull Requests

The development model used for the OE-lite.org project is a pull model. Each repository on OE-lite.org has a single developer with push access, ie. the maintainer. Only the maintainer is allowed to push changes to the respective repository.

To get changes into an OE-lite.org repository where you are not the maintainer, you have to make a pull request. A pull request is an email, or most often a series of emails, to dev@oe-lite.org containing the entire change set that is proposed together with a description of the change, additional information, and everything in a form so that it can easily be integrated with the git SCM tool.

The pull request can then be reviewed by other OE-lite.org developers, including the maintainer, and everyone has the possibility to make comments, and propose improvements to the change set. The maintainer might then decide to pull in the pull request as it is, or ask the submitter to rework the change set according and resubmit it when done, where the process restarts.

7.3.2 Preparing a Patch Set

To create a patch set for sending to dev@oe-lite.org, you can use the create-pull-request script in OE-lite/core (in the scripts directory).

Let's say you have a couple of commits in your local "my-branch" branch, which you have pushed to the "my-gitorious" remote, which is your OE-lite/base clone on gitorious.org. Your "my-branch" branch is relative to the "master" branch of the "upstream" remote (git://oe-lite.org/oe-lite/base.git). In this case, you can prepare the patch set with the following command:

```
../core/scripts/create-pull-request -u my-gitorious -b my-branch \
        -r upstream/master -i my-branch
```

As the script also will remind you, you will then have to edit a file with the cover e-mail with a proper description of your patch set.

7.3.3 Sending a Patch Set

First, you should make sure that git send-email is properly configured. You can fx. set your email address with something like this:

```
git config --global sendemail.from your.name@gmail.com
```

You know have a patch set in something like a pull-1234 directory of your meta/base subdirectory. To send that, you can use the send-pull-request script to send to dev@oe-lite.org:

```
../core/scripts/send-pull-request -a -p pull-1234 -t dev@oe-lite.org
```

For this to work, you need to have your host machine configured to be able to send e-mail, so that git send-email is able to send mails to the dev@oe-lite.org list. The details for how to do this depends very much on your host system setup, and is not covered in this handbook.

7.3.4 Single patches

In some cases creating a pull request will require a lot of work overhead.

When it is figured that a single patch will apply to the master branch of a OE-lite repository even after some time this is the faster way to submit changes to the project.

Those special cases that applies cleanly could be, e.g. new recipes, small changes to split tasks, package tasks and so on.

Let say you made a new recipe for the core repository, tested it and just committed it locally, simply do:

```
git format-patch -1 --subject-prefix=core
```

"-1" may be replaced with a specific commitid or "-2" if you want that last two commits in a patchfile.

```
git format-patch -2 mypatches/ --subject-prefix=core
```

The subject prefix is needed for now to make it visible what repository the patch applies to.

Before sending single patch files upstream make sure that you have git send-email configure as described above.

If you dont think the log message itself is saying enough to explain you change to the other members of the mailing list add "–cover-letter" to generate and editable cover letter where you can elaborate on the greater meaning with the patch (life, and everything).

```
edit mypatch/0000-* #( if coverletter has been chosen)
git send-email mypatch/*
```

or just one simple patch:

```
git send-email 0001-<commit log name>.patch
```

CHAPTER 8

Recipes

Recipes describe how to build stuff, both individual software packages, complete root file systems as well as associated utilities such as SDKs. This chapter gives a short introduction to the conventions used as well as a short example.

When starting a build (`oe bake`), OE-lite starts by parsing all recipe files in all registered layers, pruning recipes which are not compatible with the current target architecture.

8.1 Naming conventions

Recipe files end with the suffix `.oe`, and should reside in subdirectories of the `recipes` directory in one of the registered OE-lite layers.

The filename must follow the format `<name>_<version>.oe`, for example `tcpdump_4.6.2.oe`, where `<name>` is the name of the item described and `<version>` is the version of the software. The name part must not contain an underscore, since everything after the first underscore is taken to be the version. It is ok to omit the version part (including the underscore) when it doesn't apply (e.g. in the case of recipes describing an entire BSP root file system), in which case OE-lite will simply pretend it is version "0".

Inside the recipe, the `<name>` is available as the variable `${PN}`, while the `<version>` is available as `${PV}`.

8.2 Language

Recipes are written in a domain-specific language defined by OE-lite. This is not as scary as it sounds. Essentially, the job of a recipe is to set a bunch of variables. Each variable has a well-defined semantic meaning to OE-lite. There are hundreds of variables, but fortunately most retrieve their value more or less automatically, and there is a lot of infrastructure for helping with defining the rest.

A few examples of variables and their meaning:

SRC_URI Where to fetch the sources for the software.

DEPENDS Utilities and libraries necessary to build the recipe.

EXTRA_OECONF When using one of the autotools *classes*, this variable is appended to the `./configure` command line in the do_configure step.

8.3 include and require

How a piece of software gets built usually doesn't change that much from version to version, so it is quite common to put most of the logic in `.inc` files which then get included from the recipe files. A complete recipe file can be as small as:

Listing 8.1: meta/base/recipes/vim/vim_7.4.oe

```
require ${PN}.inc
```

The `require` directive instructs OE-lite to look for the given file (`unzip.inc` in the example above) and include it at that point. It is a fatal error if the file is not found. The `include` directive works similar to `require`, but if the file cannot be found parsing continues as if the `include` was not present.

8.4 Syntax

The syntax and semantics of defining and manipulating variables is similar to the one used in Makefiles. For example, the right-hand side of an ordinary assignment `FOO = "BAR"` is not expanded until `FOO` is expanded, whereas the `:=` operator causes immediate expansion of the RHS. Also, the operator += appends the RHS value to the LHS variable, but also prepends a space if the variable was non-empty.

Note, however, that OE-lite does not have the concept of variable »flavors«, and that all right-hand sides should be properly quoted strings.

See the appendix *Syntax* for a semi-formal survey of the various allowed syntactic elements.

CHAPTER 9

Examples of recipes

9.1 Trigonometric utilities

Instead of showing a small *working* recipe, we'll start small and show what fails at each step and explain how to fix it.

Suppose we wish to have utilities `sin`, `cos` and `tan` that will compute these trigonometric functions. To keep the number of files small, we just use a single source file and some preprocessor magic. So we create this directory tree:

```
recipes/trig/
            -- files
            |   -- trig.c
            -- trig_0.1.oe
```

The file `trig.c` is simply:

Listing 9.1: recipes/trig/files/trig.c

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#ifndef FUNC
#error FUNC must be defined on the command line
#endif
/* standard stringify trick */
#define ss(x) #x
#define s(x) ss(x)

int main(int argc, char *argv[])
{
        double x = 0.0, y;

        /* Error checking omitted. */
        if (argc > 1)
                x = strtod(argv[1], NULL);
        y = FUNC(x);
        printf("%s(%f) = %f\n", s(FUNC), x, y);
        return 0;
}
```

Our first attempt at describing how to build this to OE-lite is this:

Listing 9.2: recipes/trig/trig_0.1.oe

```
DESCRIPTION = "Example trigonometric utilities"
LICENSE = "GPL-2.0+"

RECIPE_TYPES = "machine native"

inherit c

SRC_URI = "file://trig.c"

do_compile() {
    for f in sin cos tan ; do
      $CC -o $f -DFUNC=$f -O2 trig.c
    done
}
```

The `DESCRIPTION` and `LICENSE` variables are self-explanatory - neither are mandatory, but both are highly recommended. When possible, it is recommended to use SPDX identifiers in the `LICENSE` fields.

The `RECIPE_TYPES` variable should be a space-separated list of the targets this recipe is applicable to. The default is `machine`, but since there's nothing machine-specific about this small utility, we also include the `native` target. That allows us to say `oe bake native:trig` to have OE-lite build the recipe for our host machine, in turn allowing us to test the programs without transferring to the target.

The `inherit c` is an example of the use of a *`class`*. Even the simplest recipes will usually inherit a few classes. The `c` class ensures that a suitable (cross-)compiler gets *`staged`* and that variables such as `CC` get appropriate values. This would be very tedious to set up manually, especially if one wants the same recipe to work for multiple target architectures.

Next, we need to tell OE-lite the source files needed. In our case, there is just one. Local files (as indicated by the `file://` prefix) are searched for in a number of subdirectories of the directory containing the recipe file: First, `${PN}-${PV}`, then `${PN}` and finally `files`. This scheme allows sharing (and non-sharing) files between different recipes and versions of the same recipe. In our case, that's not important, so we just put the file in the `files` subdirectory.

Finally, we need to tell OE-lite how to actually compile our programs. We do this by defining a shell function called `do_compile`. In a larger project, we would most likely have created a Makefile or used autotools, but here a simple shell loop is sufficient.

Let's try this:

```
$ oe bake trig -y
machine:trig_0.1:do_stage started
machine:trig_0.1:do_stage finished - 0.521 s
machine:trig_0.1:do_fstage started
machine:trig_0.1:do_fstage finished - 0.001 s
machine:trig_0.1:do_fetch started
machine:trig_0.1:do_fetch finished - 0.000 s
machine:trig_0.1:do_unpack started
machine:trig_0.1:do_unpack finished - 0.001 s
machine:trig_0.1:do_patch started
machine:trig_0.1:do_patch finished - 0.001 s
machine:trig_0.1:do_configure started
machine:trig_0.1:do_configure finished - 0.078 s
machine:trig_0.1:do_compile started
waiting for machine:trig_0.1:do_compile (started 0.020 seconds ago) to finish
ERROR: machine:trig_0.1:do_compile failed - 0.023 s
Build: 0.674 seconds
```

```
ERROR: machine:trig_0.1:do_compile failed  /mnt/xfs/devel/oe-lite/tmp/work/machine/arm-926ejs-linux-g
> LC_ALL=C /mnt/xfs/devel/oe-lite/tmp/work/machine/arm-926ejs-linux-gnueabi/trig-0.1/tmp/do_compile.2
+ cd /mnt/xfs/devel/oe-lite/tmp/work/machine/arm-926ejs-linux-gnueabi/trig-0.1/src/trig-0.1
+ do_compile
+ for f in sin cos tan
+ arm-926ejs-linux-gnueabi-gcc -o sin -DFUNC=sin -O2 trig.c
arm-926ejs-linux-gnueabi-gcc: error: trig.c: No such file or directory
arm-926ejs-linux-gnueabi-gcc: fatal error: no input files
compilation terminated.
Error: Command failed: 'LC_ALL=C /mnt/xfs/devel/oe-lite/tmp/work/machine/arm-926ejs-linux-gnueabi/tri

CRITICAL: bake failed: error: 1
```

It can sometimes be difficult to see what the problem actually is. Here the compiler complains that `trig.c` cannot be found - yet we clearly listed that as a source file. We can also see that the do_unpack task succeeded, so it *should* be there. The problem is, what does *there* mean? Let's inspect the workdir:

```
$ tree -F -L 3 tmp/work/machine/arm-926ejs-linux-gnueabi/trig-0.1/
tmp/work/machine/arm-926ejs-linux-gnueabi/trig-0.1/
-- fstage/
-- src/
|   -- patches/
|   |   -- quiltrc
|   -- trig-0.1/
|   -- trig.c
-- stage/
|   -- cross/
|   |   -- arm-926ejs-linux-gnueabi/
|   |   -- bin/
|   |   -- lib/
|   |   -- libexec/
|   |   -- OE-lite/
|   |   -- x86_64-build_unknown-linux-gnu/
|   -- machine/
|   |   -- lib/
|   |   -- OE-lite/
|   |   -- usr/
|   -- native/
|       -- include/
|       -- lib/
|       -- OE-lite/
-- tmp/
    -- do_compile.20161102082713.log
    -- do_compile.20161102082713.run*
    -- do_compile.log -> do_compile.20161102082713.log
    -- do_compile.run -> do_compile.20161102082713.run*
    -- do_stage.20161102082713.log
    -- do_stage.log -> do_stage.20161102082713.log
    -- do_unpack.20161102082713.log
    -- do_unpack.log -> do_unpack.20161102082713.log

21 directories, 10 files
```

Here we see the problem: do_compile was run in the `${WORKDIR}/src/trig-0.1/` directory (aka `${S}` – see also the section *Directories*), but `trig.c` has been put in `${WORKDIR}/src` (aka `${SRCDIR}`). The simplest fix is to make `S` and `SRCDIR` the same. So we change our recipe like this:

```
diff --git a/recipes/trig/trig_0.1.oe b/recipes/trig/trig_0.1.oe
index 73f8c02..a446526 100644
--- a/recipes/trig/trig_0.1.oe
+++ b/recipes/trig/trig_0.1.oe
@@ -7,6 +7,8 @@ inherit c

 SRC_URI = "file://trig.c"

+S="${SRCDIR}"
+
 do_compile() {
     for f in sin cos tan ; do
       $CC -o $f -DFUNC=$f -O2 -g trig.c
```

With this in place, let's try again.

```
oe bake trig -y
...
+ cd /mnt/xfs/devel/oe-lite/tmp/work/machine/arm-926ejs-linux-gnueabi/trig-0.1/src
+ do_compile
+ for f in sin cos tan
+ arm-926ejs-linux-gnueabi-gcc -o sin -DFUNC=sin -O2 trig.c
/tmp/ccrw5e1U.o: In function `main':
trig.c:(.text.startup+0x2c): undefined reference to `sin'
collect2: error: ld returned 1 exit status
Error: Command failed: 'LC_ALL=C /mnt/xfs/devel/oe-lite/tmp/work/machine/arm-926ejs-linux-gnueabi/tri

CRITICAL: bake failed: error: 1
```

Right, we didn't provide the `-lm` linker flag. OK, that's easy to fix.

```
diff --git a/recipes/trig/trig_0.1.oe b/recipes/trig/trig_0.1.oe
index a446526..932855c 100644
--- a/recipes/trig/trig_0.1.oe
+++ b/recipes/trig/trig_0.1.oe
@@ -11,6 +11,6 @@ S="${SRCDIR}"

 do_compile() {
     for f in sin cos tan ; do
-      $CC -o $f -DFUNC=$f -O2 -g trig.c
+      $CC -o $f -DFUNC=$f -O2 -g trig.c -lm
     done
 }
```

Once more. What can possibly go wrong now?

```
$ oe bake trig -y
...
ERROR: machine:trig_0.1:do_compile failed  /mnt/xfs/devel/oe-lite/tmp/work/machine/arm-926ejs-linux-g
> LC_ALL=C /mnt/xfs/devel/oe-lite/tmp/work/machine/arm-926ejs-linux-gnueabi/trig-0.1/tmp/do_compile.2
+ cd /mnt/xfs/devel/oe-lite/tmp/work/machine/arm-926ejs-linux-gnueabi/trig-0.1/src
+ do_compile
+ for f in sin cos tan
+ arm-926ejs-linux-gnueabi-gcc -o sin -DFUNC=sin -O2 trig.c -lm
/mnt/xfs/devel/oe-lite/tmp/work/machine/arm-926ejs-linux-gnueabi/trig-0.1/stage/cross/bin/../lib/gcc/
collect2: error: ld returned 1 exit status
Error: Command failed: 'LC_ALL=C /mnt/xfs/devel/oe-lite/tmp/work/machine/arm-926ejs-linux-gnueabi/tri

CRITICAL: bake failed: error: 1
```

That this fails is actually a good thing, because it shows that OE-lite works as expected! We've told the linker to link with libm. But we haven't told OE-lite that that library is needed, so it hasn't been *staged*. Telling OE-lite about build-time dependencies is precisely what the DEPENDS variable is for. By inheriting the c class, we've already told OE-lite that we depend on the standard C library (and its header files), but libm is a separate library. Now what we understand the problem, the fix is easy:

```
diff --git a/recipes/trig/trig_0.1.oe b/recipes/trig/trig_0.1.oe
index 932855c..e9e2522 100644
--- a/recipes/trig/trig_0.1.oe
+++ b/recipes/trig/trig_0.1.oe
@@ -9,6 +9,9 @@ SRC_URI = "file://trig.c"

 S="${SRCDIR}"

+DEPENDS += "libm"
+RDEPENDS_${PN} += "libm"
+
 do_compile() {
     for f in sin cos tan ; do
       $CC -o $f -DFUNC=$f -O2 -g trig.c -lm
```

The other variables we've added describes a *runtime dependency*. Without that, the utilities would build just fine, but if some image recipe then included the trig package, nothing has informed OE-lite that it must also include libm.so in the resulting image. So the binaries would be present but unrunnable. (Of course, in a realistic full BSP, *some* recipe is bound to ensure that libm gets included, but that's not necessarily the case for other libraries, so it's better to always explicitly describe the exact dependencies.)

9.1.1 Aside: dependency types

So why did we spell the runtime dependency RDEPENDS_${PN} and not just RDEPENDS? There are actually two kinds of build-time as well as two types of run-time dependencies, *recipe dependencies* and *package dependencies*. Recipe dependencies (given in the unsuffixed DEPENDS, RDEPENDS variables) describe what is required to build the recipe. Package dependencies, given in DEPENDS_<package name>, RDEPENDS_<package name, describe what is needed to use the contents of the package at build-time respectively run-time. Since our utilities end up in the package by the same name as the recipe, we tell OE-lite that anything that run-time depends on the trig *package* should also pull in libm.

An example where package build-time dependencies would come into play is if we have two libraries, libfoo and libbar and a utility frob, with libfoo depending on libbar and frob depending on libbar. In the frob recipe, we would then have something like:

```
DEPENDS += "libbar"
RDEPENDS_${PN} += "libbar"
```

The frob utility probably does a #include <bar.h> somewhere, but bar.h contains a #include <foo.h>. That libbar depends on libfoo is an implementation detail of libbar, which frob doesn't care about (and it may change with a different version of libbar), but in this case we obviously need to ensure that foo.h gets staged when building frob. The solution to this is to ensure that the package providing libbar has a build-time dependency on libfoo. So the libbar recipe might contain

```
DEPENDS += "libfoo"
DEPENDS_${PN} += "libfoo-dev"
RDEPENDS_${PN} += "libfoo"
```

which says that (1) libfoo is necessary to build libbar, (2) to build anything against libbar, you also need the libfoo-dev package, (3) if you run-time depend on libbar, you also run-time depend on libfoo.

The alert reader may wonder how a *run-time dependency* for *building* a recipe makes any sense. And in truth, most normal recipes do not have those – a bare `RDEPENDS` in a recipe is usually an error. However, there is one type of recipes which do have `RDEPENDS`: Those that inherit image.oeclass, and hence describe a complete file system image. While normal recipes have a do_stage task, which pulls in all packages mentioned in the recipe's `DEPENDS` variable as well as their package dependencies (recursively), image recipes have an do_rstage task which pulls in all the packages in the recipe's `RDEPENDS` variable as well as their package rdependencies (recursively). It is admittedly a stretch to call this run-time build dependencies, but as the preceding sentence hopefully demonstrates, this makes the handling of the two staging tasks nicely symmetric.

9.1.2 Back to the example

While we can now succesfully build the trig utilities, the recipe is not quite complete. Looking at the directory `${WORKDIR}/packages`, we see that all the packages are empty apart from some auto-generated metadata. The problem is that we haven't described how to install the utilities. Most »real« recipes get built using a Makefile (which may be generated by autotools or whatnot), in which case there is usually also an `install` target, and if we had inherited the `make` class, OE-lite would by default simply do `make install`. We, however, have to describe the install step manually, just as we defined the do_compile function. So here goes

```
diff --git a/recipes/trig/trig_0.1.oe b/recipes/trig/trig_0.1.oe
index e9e2522..07766b9 100644
--- a/recipes/trig/trig_0.1.oe
+++ b/recipes/trig/trig_0.1.oe
@@ -17,3 +17,8 @@ do_compile() {
        $CC -o $f -DFUNC=$f -O2 -g trig.c -lm
     done
 }
+
+do_install() {
+    install -m 0755 -d ${D}${bindir}
+    install -m 0755 -t ${D}${bindir} sin cos tan
+}
```

If we then run `oe bake trig -y` and look at the directory `${D}` (aka `${WORKDIR}/install`), we see that the three utilities are there. Moreover, the do_install task by default strips debug symbols and puts them in the `.debug` subdirectory:

```
$ tree -a -F tmp/work/machine/arm-926ejs-linux-gnueabi/trig-0.1/install/
tmp/work/machine/arm-926ejs-linux-gnueabi/trig-0.1/install/
-- usr/
    -- bin/
        -- cos*
        -- .debug/
        |   -- cos*
        |   -- sin*
        |   -- tan*
        -- sin*
        -- tan*

3 directories, 6 files
```

The next task is do_split, which takes the contents of the `${D}` directory and distributes the files in subdirectories of `${WORKDIR}/packages` according to the `FILES_*` variables. These have reasonable default values, so we get this structure:

```
$ tree -a -F tmp/work/machine/arm-926ejs-linux-gnueabi/trig-0.1/packages/
tmp/work/machine/arm-926ejs-linux-gnueabi/trig-0.1/packages/
-- trig/
```

```
|   -- usr/
|       -- bin/
|           -- cos*
|           -- sin*
|           -- tan*
-- trig-dbg/
|   -- usr/
|       -- bin/
|           -- .debug/
|               -- cos*
|               -- sin*
|               -- tan*
-- trig-dev/
-- trig-doc/
-- trig-locale/

10 directories, 6 files
```

This allows one to RDEPEND on `trig`, but if one also wants the debug symbols, one should also add a run-time dependency on `trig-dbg`. The final task is do_package, which adds an OE-lite directory containing a little metadata (using the LICENSE and DESCRIPTION variables), and then creates a tarball which is placed in a subdirectory of `tmp/packages`:

```
$ ls -F tmp/packages/machine/arm-926ejs-linux-gnueabi/trig*
tmp/packages/machine/arm-926ejs-linux-gnueabi/trig_0.1_f7b2f5ade7888f1426ecbe773d909f0f.tar
tmp/packages/machine/arm-926ejs-linux-gnueabi/trig_0.1.tar@
tmp/packages/machine/arm-926ejs-linux-gnueabi/trig-dbg_0.1_f7b2f5ade7888f1426ecbe773d909f0f.tar
tmp/packages/machine/arm-926ejs-linux-gnueabi/trig-dbg_0.1.tar@
tmp/packages/machine/arm-926ejs-linux-gnueabi/trig-dev_0.1_f7b2f5ade7888f1426ecbe773d909f0f.tar
tmp/packages/machine/arm-926ejs-linux-gnueabi/trig-dev_0.1.tar@
tmp/packages/machine/arm-926ejs-linux-gnueabi/trig-doc_0.1_f7b2f5ade7888f1426ecbe773d909f0f.tar
tmp/packages/machine/arm-926ejs-linux-gnueabi/trig-doc_0.1.tar@
tmp/packages/machine/arm-926ejs-linux-gnueabi/trig-locale_0.1_f7b2f5ade7888f1426ecbe773d909f0f.tar
tmp/packages/machine/arm-926ejs-linux-gnueabi/trig-locale_0.1.tar@
```

The long hex string is the metadata hash of the do_package task.

By now, we have a working recipe, and we can include the utilities on our target by simply saying

```
RDEPENDS += "trig"
```

in our root filesystem recipe. However, there are some things one might want to improve.

- In a space-constrained root filesystem, it might be nice to be able to depend on the utilities individually, so that one doesn't have to include `tan` if one only needs `cos`.
- One would not normally have the complete source code in the recipe directory, but instead have the SRC_URI point at a git repository or tar-ball containing it.

Instead of showing how to achieve this, we'll turn our attention to an example from »real life«.

9.2 Dissection of an existing recipe

CHAPTER 10

Tasks

OE-lite divides the job of building software into a number of (interdependent) tasks, each with a well-defined purpose. For example, one task is responsible for fetching the source code, another for unpacking it, a third for applying local patches, a fourth for doing the actual compilation, and so on and so forth.

Most recipes end up being split into about 13 tasks. The section *Task types* below briefly explains the purpose of the various tasks.

10.1 Environment

The filename of a recipe implicitly defines two variables, `PN` and `PV`, which are used in the definition of lots of other variables. `PN` is the name of the recipe, while `PV` is the version. For a recipe file called `openssh_7.1p2.oe`, these would be `openssh` and `7.1p2`, respectively. Moreover, `P` is a shorthand for `${PN}-${PV}`. These should never be changed from within a recipe.

10.1.1 Directories

The directory containing the *OE-lite manifest* is available as the variable `TOPDIR`. Two other standard variables are defined in terms of this, `INGREDIENTS` (`${TOPDIR}/ingredients`) and `TMPDIR` (`${TOPDIR}/tmp`).

Every recipe gets built in a dedicated subdirectory of `${TMPDIR}/work`, named according to the recipe's type, the target architecture and the recipe version. Examples are `${TMPDIR}/work/machine/arm-cortexa9neon-linux-gnueabi/openssh-7.1p2` and `${TMPDIR}/work/native/x86_64-build_unknown-linux-gnu/gmp-6.0.0a`. This is the contents of the `${WORKDIR}` variable.

There are a number of other standard variables defined in terms of `WORKDIR`, `PN` and `PV` which one should know about.

- SRCDIR `${WORKDIR}/src`
- S `${SRCDIR}/${P}`
- B `${S}`
- D `${WORKDIR}/install`
- T `${WORKDIR}/tmp`
- PKGD `${WORKDIR}/packages`

Their default values are shown above, but that may be overridden by classes or the recipe itself. We give a few examples of when this might be necessary in the task descriptions below.

10.1.2 Logging

The output from each task gets written to a log file in `${WORKDIR}/tmp`. The files are called `<taskname>.<datetime>.log`, e.g. `do_compile.20161013074154.log`, and there is a symbolic link `<taskname>.log` pointing to the most recent log file.

Empty log files get deleted automatically.

10.1.3 Scripts

Some tasks are implemented as bash functions. OE-lite runs these by writing a complete bash script to `${WORKDIR}/tmp` called `<taskname>.<datetime>.run` (again, with `<taskname>.run` being a symlink to the most recent) containing all the necessary environment settings, function definitions etc., then executes it, with stdout and stderr redirected to the `.log` file.

These scripts can also be run manually, which can be very useful as a debugging tool.

10.2 Metadata

A task is completely controlled by its associated *metadata*, which is essentially a set of key-value pairs. This metadata is copied from the metadata for the parent recipe, filtering away variables which are not relevant to the specific task.

10.2.1 Metadata hashing

In order to know whether a task needs to be redone and to facilitate use of *prebakes*, OE-lite assigns a hash value to every task. This hash value is computed from two sources: The hash values of all tasks which this task depends on, and the set of key-value pairs constituting the task's metadata. The former ensures that any change in the dependency chain (e.g. a change of compiler) causes a rebuild.

A variable can be exempt from affecting the computed hash value by setting the `[nohash]` flag. This should be done with great care, since it is only safe if it is known not to affect the binaries generated, and it is only very rarely set in classes or recipes.

10.3 Task types

10.3.1 Common tasks types

These tasks are performed for almost all recipes during a normal build. Note that for the configure, compile and install tasks, if a recipe does not define a corresponding `do_` function (and does not inherit a class defining it), it is implicitly assumed that the step is irrelevant to the recipe, so a dummy no-op function is used.

The listed task dependencies are those that must have completed succesfully before the task is started. OE-lite does a `chdir` to the given working directory before starting the task.

do_fstage

TBD.

do_fetch

This task downloads the necessary source code to the local *ingredients directory*. This is typically in the form of compressed tar-balls, but it can also perform cloning of git repositories.

Task dependencies: fstage

Working directory: `${INGREDIENTS}`

do_unpack

This extracts the source code from the local *ingredients directory* to `${WORKDIR}/src`. For a tarball, this consists of (uncompressing and) extracting the file, but it can also consist of checking out a specific commit from a git repository. It also copies local patches (files mentioned in `SRC_URI` ending with `.patch`) to `${WORKDIR}/patches`.

Task dependencies: fetch

Working directory: `${SRCDIR}`

do_patch

This applies the local patches, if any, to the source code.

Task dependencies: unpack

Working directory: `${PATCHDIR}`

do_stage

This populates the directory `${WORKDIR}/stage` with all the necessary build-time dependencies as described by the recipe's `DEPENDS` variable.

Task dependencies: do_stage depends on the existence of all the *packages* providing the items defined in the `DEPENDS` variable. If a necessary package does not already exist in the `tmp/packages` directory or can be found as a prebake, the recipe providing that package will automatically get built, in which case do_stage depends on the do_package task of the other recipe.

Working directory: `${STAGE_DIR}`

do_configure

This is responsible for configuring the software. In many cases this is the classic `./configure` step. When a recipe uses an appropriate *class*, OE-lite automatically constructs and passes the relevant command line parameters to the configure script.

Task dependencies: patch and stage

Working directory: `${B}`

do_compile

This task is where the software actually gets built. In many cases this is just calling `make`. The working directory is `${S}`.

Task dependencies: configure

Working directory: `${B}`

do_install

This installs the software under `${WORKDIR}/install`, often just by invoking `make install`. During

Task dependencies: compile

Working directory: `${B}`

do_split

This splits the files installed under `${WORKDIR}/install` into packages. Files belonging to the package `foo` gets copied to a directory tree under `${PKGD}/foo`. The splitting is governed by the `FILES_*` variables. These contain space-separated lists of glob patterns. For example, `FILES_${PN}-dev` contain (among other things) `/lib/lib*.so /usr/include`, so all

Task dependencies: install

Working directory: `${D}`

do_package

This adds some metadata (descripton, license, version etc.) to the packages created by do_split, and then wraps the directories up in a tarball.

Task dependencies: split

Working directory: `${PKGD}`

10.3.2 Other tasks

These are usually only run when requested explicitly on the command line, e.g.

```
oe bake openssl -t packageqa
```

packageqa

Perform a number of Quality Assurance checks, for example:

- For shared libraries, check that the so-name matches the `LIBRARY_VERSION` version.
- For binaries and shared libraries, check that all runtime-dependencies are actually listed in the `RDEPENDS` variable.

Task dependencies: package

Working directory: `${PKGD}`

clean

Remove the entire `${WORKDIR}` as well as the `${STAMPDIR}` – the former ensures that there are no leftovers from earlier attempts to build the recipe, while the latter prevents OE-lite from believing that certain tasks are already succesfully completed and thus eliding them. Hence a subsequent `oe bake foo` should do all tasks related to the foo recipe.

Task dependencies: none

Working directory: `${TOPDIR}`

CHAPTER 11

OE-lite Terminology

OE-lite manifest A git repository used as top-level for the project, containing as a minimum the definition of OE-lite stack used in the project (the conf/bakery.conf file). It typically also contains other project specific parts, such as project specific configuration files, and OE-lite recipes, scripts and documentation.

OE-lite stack An ordered list of OE-lite layers, and various properties assigned to these.

OE-lite layer A subdirectory of the OE-lite manifest, holding either OE-lite metadata or Python library code. An OE-lite layer is typically contained in its own git repository.

OE-lite layer, external layer An OE-lite layer hosted in a git repository not related to the projects OE-lite repository. When creating clones of the OE-lite repository, the layer will be cloned from the (external) git repository. Using OE-lite/core directly from git://oe-lite.org/oe-lite/core.git is an example of an external layer.

OE-lite layer, internal layer An OE-lite layer hosted in a git repository which is placed under the manifest repository using the same relative path as is used in the OE-lite stack, and is referenced in the OE-lite manifest using relative paths. An example of an internal layer is an OE-lite project with the manifest repository hosted at git://oe-lite.org/bsp/foobar.git has an OE-lite/core layer at meta/core hosted at git://oe-lite.org/bsp/foobar.git/meta/core, and referenced in the manifest using the url ./meta/core .

OE-lite layer, embedded layer An OE-lite layer contained directly in the OE-lite manifest repository, and is as such indivisible from the manifest. This should normally only be used for layers that has no re-use potential for other projects, now and in the future. The top of the OE-lite manifest is always treated as an implicit embedded layer. Other than this implicit top-level embedded layer, this layer type is not advisable.

OE-lite repository A bare clone of the OE-lite manifest git repository, and bare clones of any OE-lite layers using relative paths.

OE-lite class A file providing certain functionality to recipes, allowing one to avoid duplicating logic and simplify recipe files.

OE-lite recipe A recipe describes how to build a piece of software. The output of a recipe is one or more *packages*. For example, a recipe for a library would typically be split into a package containing the library itself, a -dev package containing the header files, a -dbg package containing the debug symbols and possibly a -doc package containing documentation.

OE-lite package A package is a tar-ball containing a subset of the files produced from a specific recipe. A package provides one or more *items*.

OE-lite task A task is one step in the building of a recipe.

OE-lite item An item is the fundamental unit which is used for resolving dependencies.

prebake A package which is used to satisfy dependencies without building it ourselves.

ingredients directory A directory, usually just named `ingredients` and located in the *OE-lite manifest* directory, acting as a local cache for fetched source tar-balls.

staging The process of populating the `${WORKDIR}/stage` directory with all utilities, libraries and other files necessary to build a given recipe.

CHAPTER 12

Syntax

This appendix contains a rough sketch of the formal syntax used to define OE-lite recipes.

12.1 Formal grammar

The BNF grammar below is extracted from the actual source code used to parse recipes. On the one hand, that makes it quite authoritative. On the other hand, it might have been more readable if it was a little less formal. Also, not all terminals (productions in uppercase) are defined below [1] – but at least some of the missing ones should be obvious, and we explain a few more (e.g. what constitutes a valid variable name) in the *semantics* section below.

```
syntax                ::=   statement
                            statement syntax
statement             ::=   NEWLINE
                            assignment NEWLINE
                            export_variable NEWLINE
                            include NEWLINE
                            require NEWLINE
                            inherit NEWLINE
                            func NEWLINE
                            fakeroot_func NEWLINE
                            python_func NEWLINE
                            def_func
                            addtask NEWLINE
                            addhook NEWLINE
                            prefer NEWLINE
                            COMMENT
variable              ::=   VARNAME
                            export_variable
export_variable       ::=   EXPORT VARNAME
varflag               ::=   VARNAME FLAG
varoverride           ::=   VARNAME OVERRIDE
string                ::=   empty_string
                            quoted_string
                            STRING
empty_string          ::=   QUOTE QUOTE
quoted_string         ::=   QUOTE string_value QUOTE
```

[1] Automatically extracting the regexps definining the various tokens and presenting them in a reasonable way is not easy.

```
string_value          ::=   STRING
                            STRING string_value
assignment            ::=   variable ASSIGN string
                            varflag ASSIGN string
                            varoverride ASSIGN string
                            variable EXPASSIGN string
                            varflag EXPASSIGN string
                            varoverride EXPASSIGN string
                            variable LAZYASSIGN string
                            variable WEAKASSIGN string
                            varflag WEAKASSIGN string
                            varoverride WEAKASSIGN string
                            variable APPEND string
                            varflag APPEND string
                            varoverride APPEND string
                            variable PREPEND string
                            varflag PREPEND string
                            varoverride PREPEND string
                            variable PREDOT string
                            varflag PREDOT string
                            varoverride PREDOT string
                            variable POSTDOT string
                            varflag POSTDOT string
                            varoverride POSTDOT string
include               ::=   INCLUDE INCLUDEFILE
require               ::=   REQUIRE INCLUDEFILE
inherit               ::=   INHERIT inherit_classes
inherit_classes       ::=   INHERITCLASS
                            INHERITCLASS inherit_classes
addtask               ::=   addtask_task
                            addtask_task addtask_dependencies
addtask_task          ::=   ADDTASK TASK
addtask_dependencies  ::=   addtask_dependency
                            addtask_dependency addtask_dependencies
addtask_dependency    ::=   addtask_after
                            addtask_before
addtask_after         ::=   AFTER tasks
addtask_before        ::=   BEFORE tasks
tasks                 ::=   TASK
                            TASK tasks
addhook               ::=   ADDHOOK HOOK TO HOOKNAME
                            ADDHOOK HOOK TO HOOKNAME HOOKSEQUENCE
                            ADDHOOK HOOK TO HOOKNAME addhook_dependencies
                            ADDHOOK HOOK TO HOOKNAME HOOKSEQUENCE addhook_dependencies
addhook_dependencies  ::=   addhook_dependency
                            addhook_dependency addhook_dependencies
addhook_dependency    ::=   addhook_after
                            addhook_before
addhook_after         ::=   AFTER hooks
addhook_before        ::=   BEFORE hooks
hooks                 ::=   HOOK
                            HOOK hooks
prefer                ::=   PREFER recipe maybe_layer maybe_version
                            PREFER packages maybe_recipe maybe_layer maybe_version
```

```
recipe                ::=   RECIPE RECIPENAME
maybe_recipe          ::=
                            recipe
layer                 ::=   LAYER LAYERNAME
maybe_layer           ::=
                            layer
version               ::=   VERSION VERSIONNAME
maybe_version         ::=
                            version
packages              ::=   PACKAGE package
package               ::=   PACKAGENAME
                            PACKAGENAME package
func                  ::=   VARNAME FUNCSTART func_body FUNCSTOP
func_body             ::=   FUNCLINE
                            FUNCLINE func_body
fakeroot_func         ::=   FAKEROOT func
python_func           ::=   python_func_start func_body FUNCSTOP
python_func_start     ::=   PYTHON VARNAME FUNCSTART
def_func              ::=   DEF VARNAME def_funcargs NEWLINE func_body
                            DEF VARNAME def_funcargs NEWLINE func_body FUNCSTOP
def_funcargs          ::=   ARGSTART STRING ARGSTOP
                            ARGSTART ARGSTOP
ADDHOOK               ::=   addhook
ADDTASK               ::=   addtask
AFTER                 ::=   after
APPEND                ::=   +=
ASSIGN                ::=   =
BEFORE                ::=   before
DEF                   ::=   def
EXPASSIGN             ::=   :=
EXPORT                ::=   export
FAKEROOT              ::=   fakeroot
INCLUDE               ::=   include
INHERIT               ::=   inherit
POSTDOT               ::=   =.
PREDOT                ::=   .=
PREFER                ::=   prefer
PREPEND               ::=   =+
PYTHON                ::=   python
QUOTE                 ::=   "
REQUIRE               ::=   require
TO                    ::=   to
WEAKASSIGN            ::=   ?=
```

12.2 Semantics

This section describes the semantics of the most important top-level productions in the above grammar.

12.2.1 Assignment

The most common statement in a recipe is some form of assignment. The LHS must be a valid variable name, which means that it must match the regular expression `[a-zA-Z_][a-zA-Z0-9_\-\${}\+\.]*`. In other words, it must start with a letter or underscore, and otherwise consist of alphanumeric characters, along with `-${}+.`.

The characters `${}` are not part of the actual variable name, but can be used to substitute the value of another variable. For example, if `PN` contains `openssh`, `RDEPENDS_${PN} = "something"` would assign the value `something` to `RDEPENDS_openssh`. In practice, `${PN}` is the only variable one will ever use in this context.

The RHS should normally consist of a quoted string. References to other variables can be done by wrapping them in `${}` (this differs from Makefile syntax where `$()` is used).

The semantics of the various operators is as follows:

`LHS = "RHS"`: Assign `RHS` to the variable `LHS`.

`LHS .= "RHS"`: Append `RHS` to the current value of `LHS` – if `LHS` was not defined, it is treated as if it was defined to the empty string.

`LHS =. "RHS"`: This works just like `.=` except that it prepends rather than appends.

`LHS += "RHS"`: If LHS is not currently defined or is the empty string, this works just as `LHS = "RHS"`. Otherwise, this appends a space and then `RHS` to the value of `LHS`.

`LHS =+ "RHS"`: This works just like += except that it prepends rather than appends.

`LHS := "RHS"`: Expand all variables appearing in `RHS` (recursively) and assign the result to `LHS`. It is an error if the RHS, or any of the text it expands to, refers to undefined variables.

`LHS ?= "RHS"`: If `LHS` is already defined (even as the empty string), this does nothing. Otherwise, it works just as `LHS = "RHS"`.

12.2.2 Flags

Apart from its value, a variable can also have a number of attributes, or flags. It is rarely necessary to set flags in recipes, but you may encounter the syntax in classes and configuration files.

In general, the syntax for flag settings is just as for variable settings:

```
varname[flag] = "value"
```

Some flags just serve as boolean flags (hence the name) and are hence normally only set using the `=`, `?=` and `:=` operators, while others are treated as a whitespace separated list of words.

nohash

This flag indicates that the variable it is attached to should not be part of the metadata hashing.

export

When a shell function is executed as part of a task, most of the task's metadata variables [2] are written to the shell script. Only those variables with the `export` flag set are further exported to the commands executed by the script.

Instead of setting this flag using the `varname[export] = "1"` syntax, an alternative is to use the `export varname` statement.

[2] Variables names which are not valid as shell variables, e.g. those containing −, are not exported.

unexport

A variable with this flag does not get exported to the shell environment when a shell function is run. It is thus not quite the opposite of the *export* flag.

emit

This flag is used to limit the tasks which a given variable gets copied to. If set, the variable is only emitted to the metadata instances for the tasks listed, e.g.

```
PACKAGES[emit] = "do_split do_package"
```

CHAPTER 13

Attribution-ShareAlike 3.0 Unported

Note

CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM ITS USE.

13.1 License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

13.2 1. Definitions

1. "Adaptation" means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.
2. "Collection" means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined below) for the purposes of this License.

3. "Creative Commons Compatible License" means a license that is listed at http://creativecommons.org/compatiblelicenses that has been approved by Creative Commons as being essentially equivalent to this License, including, at a minimum, because that license: (i) contains terms that have the same purpose, meaning and effect as the License Elements of this License; and, (ii) explicitly permits the relicensing of adaptations of works made available under that license under this License or a Creative Commons jurisdiction license with the same License Elements as this License.
4. "Distribute" means to make available to the public the original and copies of the Work or Adaptation, as appropriate, through sale or other transfer of ownership.
5. "License Elements" means the following high-level license attributes as selected by Licensor and indicated in the title of this License: Attribution, ShareAlike.
6. "Licensor" means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.
7. "Original Author" means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.
8. "Work" means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.
9. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.
10. "Publicly Perform" means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.
11. "Reproduce" means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

13.3 2. Fair Dealing Rights

Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

13.4 3. License Grant

Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

1. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;
2. to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";
3. to Distribute and Publicly Perform the Work including as incorporated in Collections; and,
4. to Distribute and Publicly Perform Adaptations.
5. For the avoidance of doubt:
   (a) Non-waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;
   (b) Waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor waives the exclusive right to collect such royalties for any exercise by You of the rights granted under this License; and,
   (c) Voluntary License Schemes. The Licensor waives the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved.

13.5 4. Restrictions

The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

1. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(c), as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section 4(c), as requested.
2. You may Distribute or Publicly Perform an Adaptation only under the terms of: (i) this License; (ii) a later version of this License with the same License Elements as this License; (iii) a Creative Commons jurisdiction license (either this or a later license version) that contains the same License Elements as this License (e.g.,

Attribution-ShareAlike 3.0 US)); (iv) a Creative Commons Compatible License. If you license the Adaptation under one of the licenses mentioned in (iv), you must comply with the terms of that license. If you license the Adaptation under the terms of any of the licenses mentioned in (i), (ii) or (iii) (the "Applicable License"), you must comply with the terms of the Applicable License generally and the following provisions: (I) You must include a copy of, or the URI for, the Applicable License with every copy of each Adaptation You Distribute or Publicly Perform; (II) You may not offer or impose any terms on the Adaptation that restrict the terms of the Applicable License or the ability of the recipient of the Adaptation to exercise the rights granted to that recipient under the terms of the Applicable License; (III) You must keep intact all notices that refer to the Applicable License and to the disclaimer of warranties with every copy of the Work as included in the Adaptation You Distribute or Publicly Perform; (IV) when You Distribute or Publicly Perform the Adaptation, You may not impose any effective technological measures on the Adaptation that restrict the ability of a recipient of the Adaptation from You to exercise the rights granted to that recipient under the terms of the Applicable License. This Section 4(b) applies to the Adaptation as incorporated in a Collection, but this does not require the Collection apart from the Adaptation itself to be made subject to the terms of the Applicable License.

3. If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and (iv) , consistent with Ssection 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

4. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

13.6 5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLU-

SION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

13.7 6. Limitation on Liability

EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

13.8 7. Termination

1. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.
2. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

13.9 8. Miscellaneous

1. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
2. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.
3. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
4. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
5. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.
6. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

Note

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, Creative Commons does not authorize the use by either party of the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time. For the avoidance of doubt, this trademark restriction does not form part of the License.

Creative Commons may be contacted at http://creativecommons.org/.

Authors Esben Haabendal, esben@haabendal.dk

Kim Højgaard-Hansen kimrhh@gmail.com

CHAPTER 14

Preface

The purpose of this document is to serve as a handbook for developers working on projects using OE-lite.

14.1 License

Copyright (C) 2013 Esben Haabendal.

Copyright (C) 2016 Kim Højgaard-Hansen.

Permission is granted to copy, distribute and/or modify this document under the terms of the link: Creative Commons Attribution-ShareAlike 3.0 Unported as published by Creative Commons.

14.2 Indices and tables

- genindex
- modindex
- search

I

ingredients directory, **39**

O

OE-lite class, **39**
OE-lite item, **39**
OE-lite layer, **39**
OE-lite layer, embedded layer, **39**
OE-lite layer, external layer, **39**
OE-lite layer, internal layer, **39**
OE-lite manifest, **39**
OE-lite package, **39**
OE-lite recipe, **39**
OE-lite repository, **39**
OE-lite stack, **39**
OE-lite task, **39**

P

prebake, **39**

S

staging, **40**