EasyBuild: building software with ease

Writing easyconfigs and easyblocks: the basics

Kenneth Hoste
HPC-UGent, Ghent University, Belgium
kenneth.hoste@ugent.be

`http://users.ugent.be/~kehoste/EasyBuild-basics-CSCS_20150908.pdf`

EasyBuild workshop @ CSCS, Lugano, Switzerland – Sept 8th 2015

What is an easyconfig (file)?

- build specification for EasyBuild
- mostly key-value assignments to define *easyconfig parameters*
- plain text file, Python syntax (strings, lists, dictionaries, etc.)
- specified parameters (usually) override default value
- easyconfig files typically follow a strict naming scheme
  - `<name>-<version>[-<toolchain>][-<versionsuffix>].eb`
  - toolchain label (name, version) is omitted for '`dummy`' toolchain
  - version suffix is omitted when empty
  - filename only important w.r.t. dependency resolution (`--robot`)

Example

```
name = 'GCC'
version = '4.8.3'
...
```

Available easyconfig parameters

- build specification is defined by easyconfig parameters
- ∼70 different generic easyconfig parameters are supported
- see `eb --avail-easyconfig-params` or `eb -a` for full list
- parameters specific to a particular easyblock are indicated
- include parameters for a specific easyblock via `--easyblock`/`-e`

Example

```
$ eb -a -e Binary | grep install_cmd
install_cmd(*): Install command to be used. (default: None)
```

Mandatory easyconfig parameters

- **name**, **version**: specify what software (version) to build
- **homepage**, **description**: metadata (used for `module help`)
- **toolchain**: specifies compiler toolchain to use (name, version)
- some others are planned to be required in the future
  - *docurls*, *software_license*, *software_license_urls*

Example

```
name = 'foo'
version = '1.2.3'

homepage = 'http://foo.org'
description = "foo is a tool for doing foo"

toolchain = {'name': 'intel', 'version': '2014a'}
```

Common easyconfig parameters

source files, patches

- **sources**: list of source files (filenames only)
- **source_urls**: list of URLs where sources can be downloaded
- **patches**: list of patch files to be applied (`.patch` extension)
- sources are downloaded (best effort), unless already available
- patches must be EasyBuild-compatible
  - unified diff format (`diff -ru`)
  - patched locations relative to unpacked sources

Example

```
name = 'GROMACS'
version = '4.6.1'
...
source_urls = ['ftp://ftp.gromacs.org/pub/gromacs/']
sources = ['gromacs-4.6.1.tar.gz']
patches = ['gromacs-4.6.1_Makefile-fix.patch']
```

Common easyconfig parameters

dependencies

- **dependencies**: build/runtime dependencies
- **builddependencies**: build-only dependencies (not in module)
- **hiddendependencies**: dependencies via hidden modules
- **osdependencies**: system dependencies (package names)
- modules must exist for all (non-system) dependencies
- (non-system) dependencies can be resolved via `--robot`
- format: `(<name>, <version>[, <versionsuffix>[, <toolchain>]])`

Example

```
name = 'GTI'
...
toolchain = {'name': 'goolf', 'version': '1.5.14'}
dependencies = [('PnMPI', '1.2.0')]
builddependencies = [('CMake', '2.8.12', '', ('GCC', '4.8.2')]
```

Common easyconfig parameters

configure/build/install command options

- **configopts**: options for configure command
- **preconfigopts**: options used as prefix for configure command

Analogous:

- **buildopts**, **prebuildopts**: options for build command
- **installopts**, **preinstallopts**: options for install command

Example

```
easyblock = 'ConfigureMake'
...
preconfigopts = "./autogen.sh && "
buildopts = 'CC="$CC" CFLAGS="$CFLAGS"'
installopts = 'PREFIX=%(installdir)s'
```

Common easyconfig parameters

sanity check

sanity_check_paths: files/directories that should get installed

- used to check whether installation (partly) failed unnoticed
- paths are *relative* to installation directory
- specified in Python dictionary syntax
- mandatory: *only* `files` and `dirs` keys
- values: lists of file/directory paths (one must be non-empty)
- default: non-empty `bin` *and* `lib` or `lib64` directories

Example

```
sanity_check_paths = {
  'files': ['bin/otfconfig', 'include/open-trace-format/otf.h'],
  'dirs': [('lib', 'lib64')],
}
```

Common easyconfig parameters

easyblock specification

easyblock: specify which easyblock must be used

- overrides easyblock derived from software name (default)
- usually a generic easyblock, but there are exceptions
  - `EB_OpenFOAM` for OpenFOAM and OpenFOAM-Extend
  - `EB_Score_minus_P` for Score-P, Cube, OTF2, Scalasca, ...
- automagic fallback to `ConfigureMake` *disabled* in EBv2.0!

Example

```
easyblock = 'CMakeMake'

name = 'GTI'
version = '1.2.0'
...
```

Common easyconfig parameters

module class

moduleclass: 'category' in which the software package fits

- only known module classes can be specified
- define extra module classes via `--moduleclasses`
- see default list via `--show-default-moduleclasses`
- symlink for module class is created for module (by default)

Example

```
name = 'GCC'
...
moduleclass = 'compiler'
```

Tweaking existing easyconfig files

- modify easyconfig(s) straight from command line via `--try-*`
- `--try-toolchain` to try building with a different toolchain
- `--try-software-version` to try building a different version
- `--try-amend` to try tweaking a different parameter
  - currently only for parameters with string- or list-typed values
- see `eb --help` for all `--try-*` options
- cooperates as expected with `--robot`

Example

GCC version update:

```
eb GCC-4.9.0.eb --try-software-version=4.9.1
```

install WRF + dozen dependencies with a different toolchain (!):

```
eb WRF-3.5.1-ictce-5.3.0-dmpar.eb --try-toolchain=intel,2014b -r
```

String templates and constants

Dynamic values for easyconfig parameters

- string templates are completed by easyconfig parameters
  - typically `name` and/or `version`
- help to avoid hardcoding values in multiple locations
- required for making `--try-software-version` behave as expected
- list of available templates via `--avail-easyconfig-templates`
- list of available constants via `--avail-easyconfig-constants`

Example

```
name = 'GCC'
version = '4.8.3'
...
source_urls = [
  # http://ftpmirror.gnu.org/gcc/gcc-4.8.3
  'http://ftpmirror.gnu.org/%(namelower)s/%(namelower)s-%(version)s',
]
sources = [SOURCELOWER_TAR_GZ]  # gcc-4.8.3.tar.gz
...
```

Use available generic easyblocks

- use available *generic* easyblocks where applicable
- avoids need for creating (and maintaining) new easyblocks
- (custom) easyconfig parameters allow tweaking their behavior
- overview via: `eb --list-easyblocks | grep -v EB_`
- detailed documentation on generic easyblocks is being worked on

Example

```
easyblock = 'CMakeMake'
name = 'GTI'
...
dependencies = [('PnMPI', '1.2.0')]
configopts  = '-DCMAKE_BUILD_TYPE=Release '
configopts += '-DPnMPI_INSTALL_PREFIX=${EBROOTPNMPI}'
buildopts = 'CXXFLAGS="$CXXFLAGS -fpermissive"'
...
```

What is an easyblock?

- Python module that implements a software build procedure
- 'plugin' for the EasyBuild framework
- can be *generic* (using standard tools) or *software-specific*
- additional easyblocks can be added via `--include-easyblocks`
- overview of available easyblocks via `eb --list-easyblocks`
- names of software-specific easyblocks always start with '`EB_`'
- EasyBuild v2.3.0 includes:
  - 152 software-specific easyblocks
  - 25 generic easyblocks

  to support 692 software packages!

Easyblocks vs easyconfigs

- thin line between 'fat' easyconfigs and an easyblock
- easyblocks are "do once and forget"
- central solution for build peculiarities
- can significantly simplify easyconfigs
- implemented in Python on top of framework API: very flexible
- reasons to consider an easyblock alongside a simple easyconfig:
  - 'critical' values for easyconfig parameters
  - configure/build/install options that are toolchain-dependent
  - custom (configure) options for included dependencies
  - hackish usage of parameters for existing (generic) easyblocks

Implementing easyblocks

quick overview

- each easyblock (eventually) derives from `EasyBlock` base class
- defines/extends/replaces one or more 'step' methods
- the configure/build/install steps *must* be defined

Example

```
from easybuild.framework.easyblock import EasyBlock
from easybuild.tools.filetools import run_cmd

class EB_Foo(EasyBlock):
  def configure_step(self):
    run_cmd("PREFIX=%s ./configure.sh" % self.installdir)
  def build_step(self):
    run_cmd("build.sh %s" % self.cfg['buildopts'])
  def install_step(self):
    run_cmd("install.sh")
```

Derive from existing easyblocks

avoid duplicate code

- easyblocks can be defined hierarchically through inheritance
- (generic) easyblock can serve as a basis for others
- step methods of 'parent' can be inherited/extended/redefined
- maximizes code reuse across easyblocks

Example

```
class EB_Score_minus_P(ConfigureMake):
  def configure_step(self):
    """
    Custom configure step for Score-P:
    set configure options and run configure script.
    """
    comp_opt = "--with-nocross-compiler-suite=intel"
    self.cfg.update("configopts", compt_opt)
    super(EB_Score_minus_P, self).configure_step() # parent
```

Minimal easyconfigs

easyblock takes care of the hard work

- if an easyblock is available, easyconfigs should be kept minimal
- easyblock should take care of configure/build/install options:
  - that depend on toolchain being used
  - for listed dependencies
  - that are common across builds
- sanity check paths/commands should be defined via easyblock
- easyblock can define extra custom easyconfig parameters
- easyconfig file can:
  - specify additional configure/build/instal options
  - override sanity check paths defined by easyblock

Detailed documentation

https://github.com/hpcugent/easybuild/wiki/Writing-easyblocks

- different aspects of writing easyblocks to be documented
- including examples and references to existing easyblocks
- **work in progress**

Fully worked out example easyblock/easyconfig for WRF:

https://github.com/hpcugent/easybuild/wiki/Tutorial:-building-WRF-after-adding-support-for-it

Contributing back

- contribute back your working easyconfigs/easyblocks!
- share your expertise with the community, avoid duplicate work
- especially if:
  - software package is not supported yet
  - changes are required for a new version/toolchains
  - it is frequently used software package (compilers, MPI, etc.)
- requires a limited amount of knowledge of Git/GitHub
- contributions are reviewed & thoroughly tested before being included
- see EasyBuild wiki for detailed walkthrough & guidelines:

`https://github.com/hpcugent/easybuild/wiki/Contributing-back`

`https://github.com/hpcugent/easybuild/wiki/Policy-for-easyconfig-pull-requests`