Cabal User Guide

Cabal is the standard package system for Haskell software. It helps people to configure, build and install Haskell software and to distribute it easily to other users and developers.

There is a command line tool called `cabal` for working with Cabal packages. It helps with installing existing packages and also helps people developing their own packages. It can be used to work with local packages or to install packages from online package archives, including automatically installing dependencies. By default it is configured to use Hackage which is Haskell’s central package archive that contains thousands of libraries and applications in the Cabal package format.

Contents

Introduction

Cabal is a package system for Haskell software. The point of a package system is to enable software developers and users to easily distribute, use and reuse software. A package system makes it easier for developers to get their software into the hands of users. Equally importantly, it makes it easier for software developers to be able to reuse software components written by other developers.

Packaging systems deal with packages and with Cabal we call them *Cabal packages*. The Cabal package is the unit of distribution. Every Cabal package has a name and a version number which are used to identify the package, e.g. `filepath-1.0`.

Cabal packages can depend on other Cabal packages. There are tools to enable automated package management. This means it is possible for developers and users to install a package plus all of the other Cabal packages that it depends on. It also means that it is practical to make very modular systems using lots of packages that reuse code written by many developers.

Cabal packages are source based and are typically (but not necessarily) portable to many platforms and Haskell implementations. The Cabal package format is designed to make it possible to translate into other formats, including binary packages for various systems.

When distributed, Cabal packages use the standard compressed tarball format, with the file extension `.tar.gz`, e.g. `filepath-1.0.tar.gz`.

Note that packages are not part of the Haskell language, rather they are a feature provided by the combination of Cabal and GHC (and several other Haskell implementations).

A tool for working with packages

There is a command line tool, called “`cabal`”, that users and developers can use to build and install Cabal packages. It can be used for both local packages and for packages available remotely over the network. It can automatically install Cabal packages plus any other Cabal packages they depend on.

Developers can use the tool with packages in local directories, e.g.

```
cd foo/
cabal install
```

While working on a package in a local directory, developers can run the individual steps to configure and build, and also generate documentation and run test suites and benchmarks.

It is also possible to install several local packages at once, e.g.

```
cabal install foo/ bar/
```

Developers and users can use the tool to install packages from remote Cabal package archives. By default, the `cabal` tool is configured to use the central Haskell package archive called Hackage but it is possible to use it with any other suitable archive.

```
cabal install xmonad
```

This will install the `xmonad` package plus all of its dependencies.

In addition to packages that have been published in an archive, developers can install packages from local or remote tarball files, for example

```
cabal install foo-1.0.tar.gz
cabal install http://example.com/foo-1.0.tar.gz
```

Cabal provides a number of ways for a user to customise how and where a package is installed. They can decide where a package will be installed, which Haskell implementation to use and whether to build optimised code or build with the ability to profile code. It is not expected that users will have to modify any of the information in the `.cabal` file.

For full details, see the section on building and installing packages.

Note that `cabal` is not the only tool for working with Cabal packages. Due to the standardised format and a library for reading `.cabal` files, there are several other special-purpose tools.

What's in a package

A Cabal package consists of:

- Haskell software, including libraries, executables and tests
- metadata about the package in a standard human and machine readable format (the "`.cabal`" file)
- a standard interface to build the package (the "`Setup.hs`" file)

The `.cabal` file contains information about the package, supplied by the package author. In particular it lists the other Cabal packages that the package depends on.

For full details on what goes in the `.cabal` and `Setup.hs` files, and for all the other features provided by the build system, see the section on developing packages.

Cabal featureset

Cabal and its associated tools and websites covers:

- a software build system
- software configuration
- packaging for distribution
- automated package management
    - natively using the `cabal` command line tool; or
    - by translation into native package formats such as RPM or deb
- web and local Cabal package archives
    - central Hackage website with 1000's of Cabal packages

Some parts of the system can be used without others. In particular the built-in build system for simple packages is optional: it is possible to use custom build systems.

Similar systems

The Cabal system is roughly comparable with the system of Python Eggs, Ruby Gems or Perl distributions. Each system has a notion of distributable packages, and has tools to manage the process of distributing and installing packages.

Hackage is an online archive of Cabal packages. It is roughly comparable to CPAN but with rather fewer packages (around 5,000 vs 28,000).

Cabal is often compared with autoconf and automake and there is some overlap in functionality. The most obvious similarity is that the command line interface for actually configuring and building packages follows the same steps and has many of the same configuration parameters.

```
./configure --prefix=...
make
make install
```

compared to

```
cabal configure --prefix=...
cabal build
cabal install
```

Cabal’s build system for simple packages is considerably less flexible than make/automake, but has builtin knowledge of how to build Haskell code and requires very little manual configuration. Cabal’s simple build system is also portable to Windows, without needing a Unix-like environment such as cygwin/mingwin.

Compared to autoconf, Cabal takes a somewhat different approach to package configuration. Cabal’s approach is designed for automated package management. Instead of having a configure script that tests for whether dependencies are available, Cabal packages specify their dependencies. There is some scope for optional and conditional dependencies. By having package authors specify dependencies it makes it possible for tools to install a package and all of its dependencies automatically. It also makes it possible to translate (in a mostly-automatically way) into another package format like RPM or deb which also have automatic dependency resolution.