A guided tour through the CHICKEN scheme republic

Christian Kellermann

2011-08-20 Sat, updated 2011-10-02 Sun

1 What to expect from this tour

This session will inform interested programmers about the infrastructure and motivations that drive the CHICKEN scheme republic and the people behind it. The major goal is to get you ready for programming in scheme today, so you can implement your great idea for a program in a nice programming language. After this talk you should be able to find your way on your own, know who you can ask for help and how to operate the tools that come with the CHICKEN scheme system.

2 Welcome to CHICKEN scheme

The CHICKEN scheme republic consists of

- A compiler that uses C as intermediate language and gcc to compile native code.
- An interpreter that provides a read, eval, print, loop (aka REPL) for rapid prototyping and explorative programming.
- A software package system that allows you to distribute and package software easily: The egg system.
- A wiki for gathering egg documentation, and general advice.
- A API / Identifier search system called chickadee.
- A test suite compiling all available eggs and CHICKEN core on call-cc.org called salmonella.

- A bugtracking system for all extensions and the core library.
- Mailing lists and an IRC channel for asking real people
- Code repositories for the core language and extensions

3 Installing CHICKEN scheme

Depending on your operating system you may choose your distribution's package system to install CHICKEN or build it yourself from source. Please do use the latest release version. If your distribution's package is too old you can nag the package maintainer to update and build your own in the meantime.

3.1 Building CHICKEN from a stable source

The CHICKEN core system is held in a git repository at http://code.call-cc.org The latest major release is 4.7.0. Please use this first. There are development snapshots up to 4.7.2 if you want to try or need new features not already in the latest release.

The code website offers tarballs for each.

The CHICKEN system requires GNU make as a build dependency as well as a recent C compiler. Compilation works on all major free OSes like GNU/Linux, *BSD, Haiku, as well as Mac OS X, Windows and Solaris.

So to install the 4.7.0 release on a linux box download the tarball:

```
$ wget http://code.call-cc.org/releases/4.7.0/chicken-4.7.0.tar.gz
$ tar xfz chicken-4.7.0.tar.gz
$ cd chicken-4.7.0
$ make PLATFORM=linux
# make PLATFORM=linux install
```

Other supported platforms are: bsd, cross-linux-mingw, cygwin, haiku, linux, macosx, mingw, mingw-msys, and solaris.

This will build and install CHICKEN in the default prefix ``/usr/local''. If you want to use a custom path for your installation provide a PREFIX parameter to both make calls.

```
$ make PLATFORM=linux PREFIX=$HOME/chickens/4.7.0
$ make PLATFORM=linux PREFIX=$HOME/chickens/4.7.0 install
```

Note that the PREFIX is used as part of search path for extensions by the loader and needs to be consistent for both parts.

3.2 Building a development snapshot or from git

Development snapshots differ from release tarballs in one significant way: They do not include bootstrapping code and require a pre-installed CHICKEN to build. So you need to install the latest release first and add the additional steps when building the snapshot:

```
$ make PLATFORM=linux chicken-boot
$ make PLATFORM=linux CHICKEN=./boot-chicken
$ make PLATFORM=linux install
```

Of course you can also build this in a separate PREFIX path, the same rules apply.

4 Using CHICKEN

After a successful installation you should have the following binaries in your path:

- chicken: The CHICKEN compiler, called by csc
- csc: The CHICKEN scheme compiler driver
- csi: The CHICKEN scheme interpreter
- chicken-install: For installing CHICKEN scheme eggs
- chicken-status: shows the currently installed eggs
- chicken-uninstall: gets rid of rotten eggs
- chicken-profile: Shows statistics of instrumented CHICKEN code
- chicken-bug: Generate bug report templates

4.1 Using the interpreter

Calling csi without arguments will start a REPL for you:

```
$ csi

CHICKEN
(c)2008-2011 The Chicken Team
```

```
(c)2000-2007 Felix L. Winkelmann
Version 4.7.0
openbsd-unix-gnu-x86 [ manyargs dload ptables ]
compiled 2011-06-25 on necronomicon.my.domain (OpenBSD)

#;1> "hello world!"
"hello world!"
#;2> (+ 1 2)
3
#;3> (string-split #1)
("hello" "world!")
```

As you can see you can refer to results from an earlier command by using its history number prefixed with a pound sign (#). To leave the interpreter type ``,q''. To see a list of available special commands type ,?.

4.2 Running interpreted scripts

Another common use for csi is to be used as a interpreter for your CHICKEN scripts. Here is a small script that reverses all input:

```
#!/usr/local/bin/csi -s
; This imports the nonstandard procedures read-all and string-reverse
(use utils srfi-13)
(display (string-reverse (read-all)))
(newline)
```

When calling the script, it does reverse all input:

```
$ chmod +x ./rev.scm
$ echo -n hello world|./rev.scm
dlrow olleh
$
```

Notice the -s parameters above. -s will exit the REPL for you when the script has been evaluated completely. For a complete list call ``csi -h''.

4.3 Compiling your programs

The real strength of CHICKEN is its compiler. Your code will run faster and some more advanced features like calling C functions of other libraries in your scheme code are available during compilation only.

For everyday usage you will use the CHICKEN scheme compiler driver mostly. Suppose we want to compile our reverse script from above:

```
$ csc rev.scm
$ echo -n hello world|./rev
dlrow olleh
$
```

csc will generate a binary that has the same name as your scheme file without the .scm extension. Note that this extension is not enforced by the compiler but has been used by most schemers in the past.

The resulting binary will be dynamically linked to the CHICKEN runtime library. In a more complex example you may need to add additional libraries for the linker yourself, for example external libraries that you use, like SDL, libexif, etc.

Building static binaries is still possible but officially not recommended anymore. For starters ignore static compilation.

If you want to distribute your binary as a standalone application without the need of CHICKEN installed there is the possibility of putting all the eggs in one basket, by the use of the -deploy option (more on this below).

4.4 More than just standard scheme: Extending R5RS with Units and Eggs

The simple example above is of course not at all that simple. There is a lot of work going on under the hood. Reading everything from a port, then reversing this string. If you already know scheme and the scheme standard you might have noticed that these procedures are not part of R5RS, the scheme standard CHICKEN is based on.

To meet the goal of being a practical scheme environment CHICKEN ships with extensions included in the core distribution. These extensions are called units for historical reasons and are covered by the CHICKEN Manual. They include these topics:

- Unit library: Basic Scheme definitions
- Unit eval: Evaluation
- Unit expand: Modules and macros handling
- Unit data-structures: Data structures

- Unit ports: I/O ports
- Unit files: File and pathname operations
- Unit extras: Useful utility definitions
- Unit irregex: Regular expressions
- Unit srfi-1: List Library
- Unit srfi-4: Homogeneous numeric vectors
- Unit srfi-13: String library
- Unit srfi-14: Character set library
- Unit srfi-18: multithreading
- Unit srfi-69: Hashtable Library
- Unit posix: Unix-like services
- Unit utils: Shell scripting and file operations
- Unit tcp: Basic TCP-sockets
- Unit lolevel: Low-level operations

As those are so commonly used in scheme programming they are part of the core distribution. Extensions not shipped with the core system are called ``eggs''. Those can be installed with the chicken-install program. If we need a sha1-sum in our application for example we just install the ``simple-sha1'' egg:

```
$ chicken-install simple-sha1
retrieving...
[loads of output]
$
```

This will need a working internet connection and download the extension and its scheme dependencies from call-cc.org or a mirror. After that the extension is compiled and placed in your PREFIX, ready to be used. To be able to use the procedures exported by an egg just place (use egg-name) in your program. If we now want to calculate a sha-1 has from all input read instead of reversing it, change or script like this:

```
#!/usr/local/bin/csi -ns
; This imports the nonstandard procedures read-all
(use utils)
; This makes the sha1 egg procedures available
(use simple-sha1)
(display (string->sha1sum (read-all)))
(newline)
```

So let's see the sha1 hashsum for the string ``hello world'':

```
$ csc sha1.scm
$ echo -n "hello world" | ./sha1
2aae6c35c94fcfb415dbe95f408b9ce91ee846ed
$
```

As you can see it is quite easy to reuse other people's code by using their published extensions. How do I know which procedures simple-sha1 exports? I have checked the documentation.

4.5 Finding your way: The documentation

4.5.1 Manual

CHICKEN scheme is documented in mainly two places, one offline and one online: The manual and the wiki. The manual is part of every release tarball and installs HTML versions of the wiki pages locally in $PREFIX/share/chicken/doc/manual/. Point your browser to the index.html file in this directory to get started.

The manual is also available online in the CHICKEN wiki. The wiki resides at http://wiki.call-cc.org with the manual being available at http://wiki.call-cc.org/manual

The manual covers the usage of the core binaries, the deviations and extensions to the R5RS standard and the documentation of the core units.

4.5.2 Wiki and egg index

The wiki also includes documentation for eggs. All eggs are supposed to host their api description there. Also there is a list of available eggs in the egg index at: http://wiki.call-cc.org/egg-index

Note that the wiki can be edited by anyone including yourself. Feel free to reword unclear passages or correct spelling errors as you encounter them. The wiki is backed by svn so do not be afraid of breaking anything.

Since it is svn you can also do a checkout of the whole wiki and edit it locally in your favourite text editor:

$ svn co https://code.call-cc.org/svn/wiki wiki (user: anonymous, empty password)

4.5.3 Chickadee

The wiki's search function is currently not optimised for searching for suitable procedures. For identifier searches use the chickadee. Chickadee is reachable at http://api.call-cc.org and supports identifier searching as well as regexp searching.

Both chickadee and the wiki search the manual as well as egg documentation.

4.5.4 Searching offline: chicken-doc

The database for chickadee can be reused offline by installing the chicken-doc extension. To be able to use chicken-doc install it with chicken-install and download the database:

```
$ chicken-install chicken-doc
$ cd `csi -p '(chicken-home)'`
$ curl http://3e8.org/pub/chicken-doc/chicken-doc-repo.tgz | sudo tar zx
```

After this you can query chicken-doc for procedures as well as whole module documentation. Calling chicken-doc without arguments will list all possible options as well as usage examples.

For our simple-sha1 example we can ask chicken-doc about anything that contains ``sha1'' in its name:

```
$ chicken-doc -m sha1
(sha1)                          sha1 egg
(sha1 sha1-binary-digest)       (sha1-binary-digest SOURCE)
(sha1 sha1-digest)              (sha1-digest SOURCE)
(sha1 sha1-primitive)           (sha1-primitive)
(simple-sha1 sha1sum)           (sha1sum filename)
(simple-sha1)                   simple-sha1 egg
(simple-sha1 string->sha1sum)   (string->sha1sum str)
```

So there are two eggs for calculating sha1 hashes: sha1 and simple-sha1. Let's have a look at the egg documentation for the simple-sha1 egg as we only want to convert a string to a hash:

```
$ chicken-doc simple-sha1
[simple-sha1's egg page]
```

we could have skipped this step and just asked for the description of string->sha1sum:

```
$ chicken-doc "string->sha1sum"
path: (simple-sha1 string->sha1sum)

-- procedure: (string->sha1sum str)

Generates the SHA1 hash for the given string `str`.
$
```

The latter tells us that string->sha1sum is part of the simple-sha1 egg (in the ``path:'' line) and it does what we need for our little script.

4.6 The home of bugs: CHICKEN'S bug tracker

CHICKEN scheme uses trac as its bug tracker. It can be reached at http://bugs.call-cc.org. Currently filing a new ticket requires getting a trac account, which we will do happily upon request. Please follow the instructions on http://wiki.call-cc.org/contribute to get one.

4.7 Getting help from humans

Searched through the manual, wiki, bugs, and chickadee without finding an answer to your question? Then it might be time to ask a human. Chickenistas can be found online on IRC in #chicken on freenode or if you want to reach more people and have an archived form of your question you can post to one of the mailing lists. These are:

- chicken-users: For all general questions about the usage of CHICKEN scheme. If in doubt post to this list.
- chicken-hackers: For all questions about the implementation of scheme in CHICKEN
- chicken-janitors: The bug tracking list which will post all changes to the bug tracking system. Subscribe here if you are interested in CHICKEN's bugs and the handling of those

These lists are quite responsive as is the IRC channel. Provided the usual netiquette you will get an answer to your problem almost instantly.

5 Programming in CHICKEN

Up to now you should be able to find your way around and write simple scripts. There are a couple of hints and tricks scattered throughout the CHICKEN village that try to make programming in CHICKEN scheme even more pleasant. The most requested things are collected here.

5.1 Choosing an editor

People in #chicken usually do not engage in the editor wars. Be it notepad, vim, nano or emacs you should choose the one that floats your boat. There are bindings for vim and emacs available that enable auto completion of identifiers, easy integration of the REPL and basic compilation support. As emacs is more popular among schemers the support here is a bit better but in the end it does not matter much. If you are coming from Common Lisp you might enjoy the SLIME egg that provides a SWANK backend for the famous Lisp mode.

Another common caveat for new users is the missing line editing support for csi. Here you can either install bindings for readline, linenoise or parley to get line editing support. The installation procedures are covered in the documentation for those eggs.

5.2 Building complex projects

Once your application grows you will want to split up your code into separate, independent entities. The CHICKEN way to do so is to write modules. Modules give you means to separate and manipulate namespaces and encourage encapsulation and controlled exposure of your code to the outside world.

Compiling more than one file on the command line is a tedious task. To automate this you can either write a Makefile as with other projects or use the systems egg. The systems egg works as a simple build system, that allows you to recompile your code from inside the REPL as well as on the command line: Load a system definition and tell the system egg to recompile it and you can continue testing your program. A typical definition looks like this:

```
(define-system xyz
    (file "z")
    (scheme-file "y" includes: '("z"))
    (compiled-scheme-file "x" depends: '("y")))
```

You can then use this definition in csi like:

```
#;1> (use system posix)
#;2> (load "xyz.system")
#;3> (load-system xyz)
```

Whenever you load the system changes to the defined files will be detected and recompilation initiated automatically. You can wrap this in a csi script for replacing a Makefile as well. This can be especially useful if you are using FFI code in your application which is only available in compiled code.

5.3 Writing your own extensions

If you have modules that may be used by others or you want to be able to deploy those with chicken-install you can wrap them into an egg. An egg can include 1 or more modules and executables. The current infrastructure (known as ``the new system'' amongst chicken-users) allows you to either include your egg in the svn egg tree (you will need to apply for an account, where ``application'' means just asking for it) or you can host it in a git, mercurial or fossil repository. If you choose the decentralised version you still need to get an entry into a centralised egg-location file for chicken-install being able to pick it up. Again, this is another matter of asking for it.

Whatever you choose please consider your egg complete only if you provide documentation for it in the wiki. Otherwise people are unable to find information about it neither through chickadee nor chicken-doc.

An egg consists of your source files, a so called .meta file, a .setup script that handles the compilation of your source files and optionally a test directory including a test suite that will get run by salmonella, CHICKEN's test infrastructure.

Let's suppose you want to write an mp3 decoder. A setup script for it may look like this:

```
; These two instructions will produce a dynamically linkable object file "mpeg3.so" respe
(compile -s -O2 -d1 mpeg3.scm -j mpeg3)
(compile -s mpeg.import.scm -O2 -d0)

(install-extension
; Name of your extension:
'mpeg3
; Files to install for your extension:
'("mpeg3.so" "mpeg3.import.so")
; Assoc list with properties for your extension:
'((version 1.2)))
```

For the egg index as well as declaring dependencies for your egg the .meta file is used:

```
(
; Your egg's license:
(license "BSD")

(category misc)

; A list of eggs mpeg3 depends on.  If none, you can omit this declaration
; altogether. `depends' is an alias to `needs'.
; Notice that you should NOT put CHICKEN units (e.g., srfi-1, srfi-13
; and many others) in `needs' or in `depends'.
(needs sandbox syntax-case)

; A list of eggs required for TESTING ONLY.  See the `Tests' section.
; Just like `needs' and `depends', `test-depends' should NOT contain
; CHICKEN units.
(test-depends test)

(author "Your Name Goes Here")
(synopsis "A basic description of the purpose of the egg."))
```

Now to release the egg create a tag that matches the name of the version clause above (``1.12'') and people are able to chicken-install your egg. All the details to this procedure is found at http://wiki.call-cc.org/eggs%20tutorial

5.4 Deploy your application as a standalone bundle

CHICKEN scheme supports the deployment of your application as a self-contained version without the need of CHICKEN installed on your target machine.

To use this use the -deploy option of csc. To compile a single executable it is sufficient to add -deploy to the call. If you are using eggs in your application you need to use chicken-install with the -deploy option. As for our sha1 example this means that the following will result in a directory called sha1 containing everything you need to run this on another machine (provided it is the same OS and architecture):

```
$ csc -deploy sha1.scm
$ chicken-install -prefix $PWD/sha1 -deploy simple-sha1
$ ls -l sha1
total 6328
-rwxr-xr-x  1 ckeen  ckeen  3124541 Aug 13 18:16 libchicken.so.6
-rwxr-xr-x  1 ckeen  ckeen    13074 Aug 13 18:16 sha1
-rwxr-xr-x  1 ckeen  ckeen    12164 Aug 13 18:18 simple-sha1.import.so
-rw-r--r--  1 ckeen  ckeen       92 Aug 13 18:18 simple-sha1.setup-info
-rwxr-xr-x  1 ckeen  ckeen    27647 Aug 13 18:18 simple-sha1.so
```

Please note that you need to install all extensions on your build system first. Installing directly from the egg repositories without this step will not work for deployment for reasons beyond the scope of this tour. Also this currently does not work on NetBSD; OpenBSD needs a trampoline script that sets LD_LIBRARY_PATH

5.5 Using the test infrastructure

Salmonella, located at http://tests.call-cc.org, will compile your egg every day against the development branch of CHICKEN, giving you hints for API changes in CHICKEN and build problems. You can subscribe to RSS feeds of your eggs to get notifications if your egg suddenly fails to build.

6 Wrapping up

Now we have reached the end of our tour through the CHICKEN landscape. There have been rough edges and lots of omissions for the sake of clarity and simplicity. I hope you got a good impression of the CHICKEN infrastructure and will try it out on your own during the workshop.

7 Thanks

I want to thank the fine people from `#chicken` on Freenode IRC for their corrections and suggestions. Many corrections have been done by John Gabriele.