Scripting with Lisp

Using Common Lisp as a Scripting Language

François-René Rideau <tunes@google.com>

International Lisp Conference, 2014-08-17

Ways that CL beats scripting languages

BOTH lexical variables AND dynamic variables

Recursive data structures, not mere string manipulation.

Native read-write invariance, not manual serialization.

Higher-order functions, not just top-level functions.

Pattern-matching, not just regular expressions.

Interactive debugging, not just error (back)traces.

Condition handling, not just global traps

An advanced Object System, not just multiple inheritance.

Native code compilation, not just script interpreters.

Ways scripting languages used to beat CL

"Scripting" (whatever that means)

Popularity (seems to follow from "scripting")

Plenty of libraries (follows from "popularity")

Familiarity (follows from "popularity")

What is Scripting?

Low-overhead, frictionless programming

Can reuse other people's code

Can write code reusable by others

Division of Labor

CL is NOW acceptable at scripting

Software Modularity

mk-defsystem (1991); ASDF 1 (2002), 2 (2010), 3 (2013), 3.1 (2014)

Software Distribution

asdf-install (2003); clbuild 1 (2007), 2 (2010); quicklisp (2010)

Run CL code from Unix

cl-launch 1 (2005), 2 (2006), 3 (2012), 4 (2014)

Run Unix code from CL

xcvb-driver (2009), uiop (2013); inferior-shell 1 (2012), 2 (2014)

Low overhead

One-line script invocation.

```
#!/usr/bin/cl
```

Write Once, Run Most-anywhere (WORM)

Any OS, Implementation combination

No sysadmining required

No editing scripts, configuration, etc.

Writing a Unix-style script in Lisp

```
#!/usr/bin/cl -sp lisp-stripper -E main
(defun main (argv)
  (if argv
      (map () 'print-loc-count argv)
      (print-loc-count *standard-input*)))
```

*lispwc *.lisp*

Invoking CL code from the shell

```
#!/bin/sh

form='`#5(1 ,@`(2 3))'

for impl in allegro ccl clisp sbcl ecl \
            lispworks abcl cmucl gcl scl xcl ; \
do
  cl -l $impl \
      "(format t \"$impl ~S~%\" $form)" \
  2>&1 | grep "^$impl " # LW, GCL are verbose
done
```

Invoking external commands from Lisp

```
#!/usr/bin/cl -sp inferior-shell

(loop with form = "`#5(1 ,@`(2 3))"

   for impl in '(allegro ccl clisp sbcl ecl
                 lispworks abcl cmucl gcl scl xcl)
   do
   (run `(pipe (cl -l ,impl (>& 2 1)
                    ("(format t \"" ,impl " ~S~%\" "
                       ,form ")"))
            (grep ("^" ,impl " "))))))
```

Better abstractions for scripting

```
(loop with form = "`#5(1 ,@`(2 3))"

   for impl in '(allegro ccl clisp sbcl ecl
                 lispworks abcl cmucl gcl scl xcl)
   collect
   (run `(pipe (cl -l ,impl (>& 2 1)
                    ("(format t \"" ,impl " ~S~%\" "
                        ,form ")"))
            (grep ("^" ,impl " "))) :output :forms))
```

Standards-based portability

```
`#5(1 ,@`(2 3))
```

```
((ALLEGRO #(1 2 3 2 3 2 3 2 3))
 (CCL #(1 2 3 2 3 2 3 2 3)))
 (CLISP #(1 2 3 2 3 2 3 2 3)))
 (SBCL #(1 2 3 2 3 2 3 2 3)))
 (ECL #(1 2 3 3 3)))
 (LISPWORKS #(1 2 3 3 3)))
 (ABCL #(1 2 3)))
 (CMUCL #(1 2 3)))
 (GCL #(1 2 3)))
 (SCL #(1 2 3)))
 (XCL #(1 2 3))))
```

Now Portable

invocation: main, argv

modularity: image dump and restore hooks

access: run-program

runtime: pathnames, getenv, temp files...

Users need not be developers

No editing paths in source files

No ./configure ; make ; make install

No object files in your or each other's way

Just invoke the script, click, etc.

Users need not be sysadmins

No editing paths in configuration files

No deploy step before use

Share source code, not object code

Sysadmining left to sysadmin and easy.

Modularity

defsystem

Just download source into a registered tree

Refer to code by name, not path

Cost: scanning tree

Lisp is the Virtual Machine

Bytecode 40: begin program

Bytecode 41: end program

fasl cache: per-user persistent file-grained JIT

Same (non) config for compile-time vs runtime

NOW much better than C, as good as Python

Easier delivery with bundle operations

Deliver an executable: **`cl-launch`**

Deliver a library: **`asdf:compile-bundle-op`**

Deliver code as only one or two files!

Image Life-cycle support

Need to use environment variables?

```
(uiop:register-image-dump-hook 'clear-env-vars)

(uiop:register-image-restore-hook 'init-env-vars)
```

Image Life-cycle support

Need to use environment variables?

```
(uiop:register-image-dump-hook 'clear-env-vars)

(uiop:register-image-restore-hook 'init-env-vars)
```

Many other uses

A standard interface *matters*

Scripting Language?

Scripting Language?

Low-overhead programming

No boilerplate

Write Once, Run Most-anywhere *unmodified*

No setup needed

Spawn or be spawned by other programs

call or be called by functions in other languages

What is it all about?

What is it all about?

ASDF 3 does nothing that cannot be done without it

What is it all about?

ASDF 3 does nothing that cannot be done without it

Neither does any piece of software

What is it all about?

ASDF 3 does nothing that cannot be done without it

Neither does any piece of software

Division of labor

What is it all about?

ASDF 3 does nothing that cannot be done without it

Neither does any piece of software

Division of labor

Enabling the division of labor

Beyond ASDF 3

Beyond ASDF 3

less overhead:

ASDF 3.1: **`asdf:package-inferred-system`**

more modularity:

ASDF 3.1: **`*readtable*`** protection

more access:

Integration with other languages?

Lessons for other languages

less overhead

more modularity

more access

Extended version of my ELS 2014 article...

The basic design of ASDF

Why it rocks / sucks compared with C build tools

Innovations in ASDF 1 2 2.26 3 3.1

The Problem with Pathnames

Lessons in Software Design including Pitfalls

A great bug chase story

http://github.com/fare/asdf3-2013

Share and Enjoy!

http://common-lisp.net/project/asdf/

http://cliki.net/cl-launch

http://cliki.net/inferior-shell

http://www.quicklisp.org/beta/

http://github.com/fare/asdf3-2013

Any Questions?