Lightning Talks I

ELS 2016

- **01 QTools UI - *Nicolas Hafner***
- 02 Contributing to the CL Ecosystem - *Daniel Kochmański*
- 03 A Job - *Irène Durand*
- 04 Lisp and AWS - *Andrew Lawson*
- 05 Backward Compatibility - *Anton Vodonosov*
- 06 Symbols as Namespaces - *Alessio Stalla*
- 07 FoCus - *Didier Verna*

Nicolas Hafner

@Shinmera

`https://everything.shinmera.com`

Last Year...

- Qtools makes using Qt from CL neat

Last Year...

- Qtools makes using Qt from CL neat
- But Qt itself is awkward sometimes:
- Layouts are lacklustre
- Some features can't be changed
- Widgets are not extensible enough

What to Do?

- Qt sources can’t feasibly be changed
- There’s no way to hack fixes in otherwise

What to Do?

- Qt sources can’t feasibly be changed
- There’s no way to hack fixes in otherwise
- So we need to rewrite it all
- but this time in Lisp!

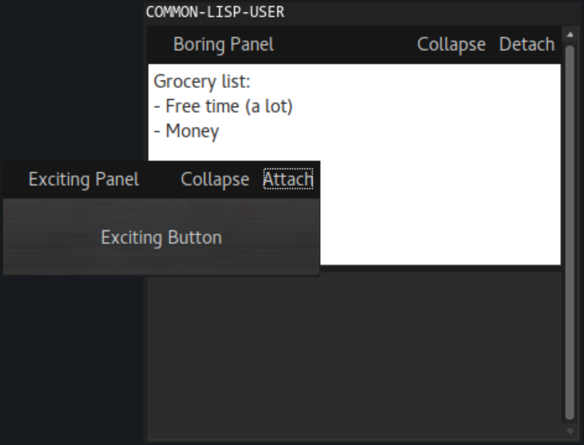
What to Do?

- Qt sources can’t feasibly be changed
- There’s no way to hack fixes in otherwise
- So we need to rewrite it all
- but this time in Lisp!   \o/

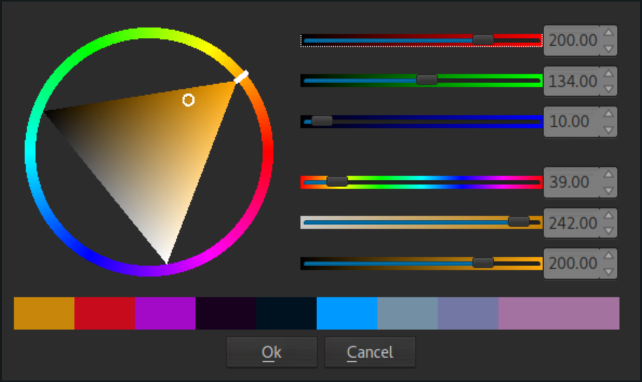
Enter Qtools-UI

- Offers a new, dynamic layouting protocol
- Implements a few standard layouts and widgets
- Includes useful base-classes to work with
- And completely pre-made components

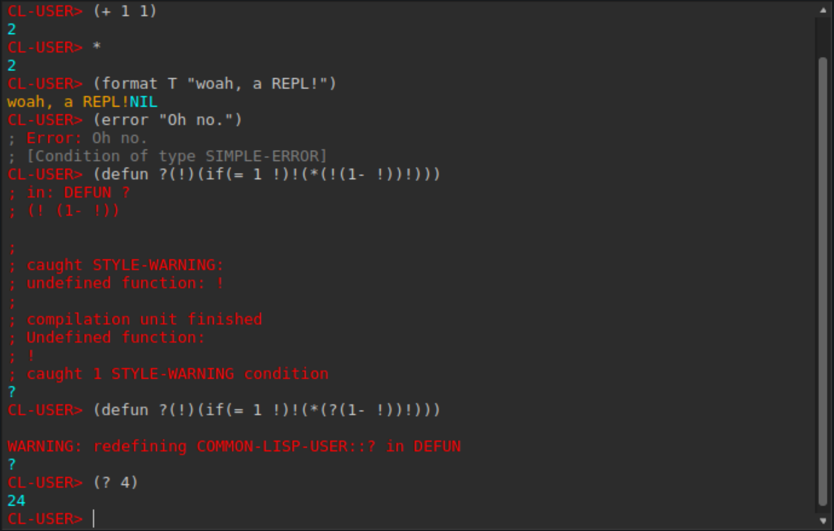
Things Like...



Grocery list:
- Free time (a lot)
- Money

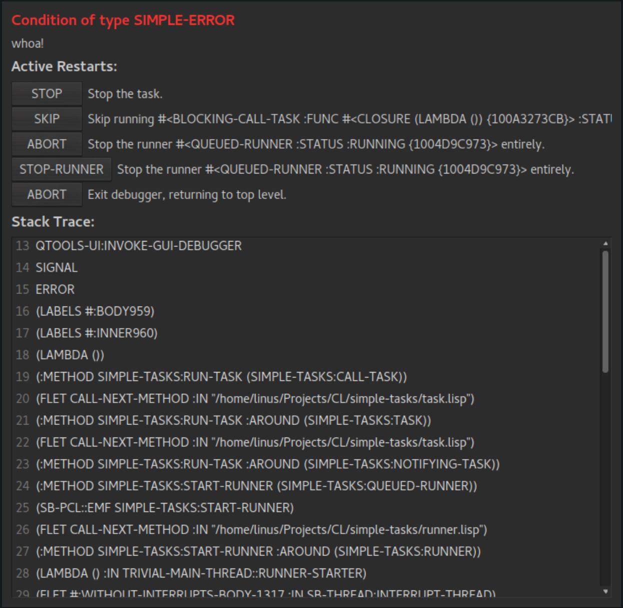


200.00
134.00
10.00
39.00
242.00
200.00



```
CL-USER> (+ 1 1)
2
CL-USER> *
2
CL-USER> (format T "woah, a REPL!")
woah, a REPL!NIL
CL-USER> (error "Oh no.")
; Error: Oh no.
; [Condition of type SIMPLE-ERROR]
CL-USER> (defun ?(!)(if(= 1 !)!(*(!(1- !))!)))
; in: DEFUN ?
; (! (1- !))
;
; caught STYLE-WARNING:
; undefined function: !
;
; compilation unit finished
; Undefined function:
; !
; caught 1 STYLE-WARNING condition
?
CL-USER> (defun ?(!)(if(= 1 !)!(*(?(1- !))!)))

WARNING: redefining COMMON-LISP-USER::? in DEFUN
?
CL-USER> (? 4)
24
CL-USER> |
```



```
Condition of type SIMPLE-ERROR
whoa!
Active Restarts:
STOP         Stop the task.
SKIP         Skip running #<BLOCKING-CALL-TASK :FUNC #<CLOSURE (LAMBDA ()) {100A3273CB}> :STATI
ABORT        Stop the runner #<QUEUED-RUNNER :STATUS :RUNNING {1004D9C973}> entirely.
STOP-RUNNER  Stop the runner #<QUEUED-RUNNER :STATUS :RUNNING {1004D9C973}> entirely.
ABORT        Exit debugger, returning to top level.
Stack Trace:
13 QTOOLS-UI:INVOKE-GUI-DEBUGGER
14 SIGNAL
15 ERROR
16 (LABELS #:BODY959)
17 (LABELS #:INNER960)
18 (LAMBDA ())
19 (:METHOD SIMPLE-TASKS:RUN-TASK (SIMPLE-TASKS:CALL-TASK))
20 (FLET CALL-NEXT-METHOD :IN "/home/linus/Projects/CL/simple-tasks/task.lisp")
21 (:METHOD SIMPLE-TASKS:RUN-TASK :AROUND (SIMPLE-TASKS:TASK))
22 (FLET CALL-NEXT-METHOD :IN "/home/linus/Projects/CL/simple-tasks/task.lisp")
23 (:METHOD SIMPLE-TASKS:RUN-TASK :AROUND (SIMPLE-TASKS:NOTIFYING-TASK))
24 (:METHOD SIMPLE-TASKS:START-RUNNER (SIMPLE-TASKS:QUEUED-RUNNER))
25 (SB-PCL::EMF SIMPLE-TASKS:START-RUNNER)
26 (FLET CALL-NEXT-METHOD :IN "/home/linus/Projects/CL/simple-tasks/runner.lisp")
27 (:METHOD SIMPLE-TASKS:START-RUNNER :AROUND (SIMPLE-TASKS:RUNNER))
28 (LAMBDA () :IN TRIVIAL-MAIN-THREAD::RUNNER-STARTER)
29 (FLET #:WITHOUT-INTERRUPTS-BODY-1317 :IN SB-THREAD:INTERRUPT-THREAD)
```

What’s Next?

- More layouts
- More components

What’s Next?

- More layouts
- More components
- More everything!

What’s Next?

- More layouts
- More components
- More everything!
- More extensive testing needed
- Contributors are more than welcome

QtoolsUI

`https://shinmera.github.io/qtools-ui`

Available on Quicklisp

- 01 QTools UI - *Nicolas Hafner*
- **02 Contributing to the CL Ecosystem *- Daniel Kochmański***
- 03 A Job - *Irène Durand*
- 04 Lisp and AWS - *Andrew Lawson*
- 05 Backward Compatibility - *Anton Vodonosov*
- 06 Symbols as Namespaces - *Alessio Stalla*
- 07 FoCus - *Didier Verna*

Contributing to the Common Lisp ecosystem

Daniel "jackdaniel" Kochmański

May 4, 2016

TurtleWare

Outline

Write the software

Pros

- eternal fame
- growing ecosystem
- freedom of choice

Cons

- yaXI - yet another X library
- NIH syndrome
- Reinventing the wheel

Duties

- State of the art
- Write portable code and tests
- Use portability layers if needed
- Write documentation

Improve the existing software

- Provide the bug fixes
- Implement new features
- Maintain abandoned packages
- Write documentation
- Create tutorials
- Improve portability
- Perform cl-test-grid tests on your platform/implementation

TurtleWare

Teach the Common Lisp

- Write a tutorial, a blog post or an essay
- Improve existing resources (cliki, llthw, . . .)
- Help new CL programmers (#clnoobs, #lisp, stackoverflow, . . .)
- Participate in ECL Quarterly (shameless plug goes here)
- Create local Common Lisp group
- Promote CL in your work environment

TurtleWare

Get involved with the existing implementations

- Report bugs
- Write tests
- Improve documentation
- Fix bugs, improve code and implement features
- Build and run tests on your platform

Improve implementations compatibility

- Contribute to the portability layers
- Before implementing the new exciting feature write a specification and the test suite for it
- Contribute to ansi-test, cl-test-grid, portability layers, . . .

Priorities (highly opinionated)

- Documenting and writing the specification for ASDF and UIOP
- Finding the CLISP maintainer
- Refactoring the libraries to use the portability layers instead of maintaining their own hand-written implementations
- Providing a good learn resources for the beginners *Learn Lisp the Hard Way*
- Creating a non-Emacs IDE for Common Lisp

TurtleWare

Still don't know how to contribute?

- If you have an extra money pass it to the Common Lisp Foundation
- If you are a musician write a song about lisp
- If you have a project to outsource hire the Common Lisp hackers
- If you don't know Common Lisp yet go and learn some!

Questions?

This presentation is available at

www.turtleware.eu/static/talks/els-2016-lt.pdf
www.turtleware.eu/static/talks/els-2016-lt.org

Contact

Daniel "jackdaniel" Kochmański
daniel@turtleware.eu

- 01 QTools UI - *Nicolas Hafner*
- 02 Contributing to the CL Ecosystem - *Daniel Kochmański*
- **03 A Job - *Irène Durand***
- 04 Lisp and AWS - *Andrew Lawson*
- 05 Backward Compatibility - *Anton Vodonosov*
- 06 Symbols as Namespaces - *Alessio Stalla*
- 07 FoCus - *Didier Verna*

Software Development Engineer (Temporary Job)

Irène Durand	`dept-info.labri.fr/~idurand`
ANR GRAPHEN	`graphen.isima.fr`
LaBRI	`www.labri.fr`
University of Bordeaux	`www.u-bordeaux.fr`

- Design a dynamic Web interface for the TRAG system (Term Rewriting Automata and Graphs – written in `CL`).
- (if time left) Set up a database to store submitted examples

---

Required education:	at least a Master degree in Computer Science
Montly salary:	€1972.5 (gross) €1595.82 (net)
Duration:	18 months (start asap, end before September 2019)
Skills:	Programming (all paradigms), Web programming and technologies, Databases

- 01 QTools UI - *Nicolas Hafner*
- 02 Contributing to the CL Ecosystem - *Daniel Kochmański*
- 03 A Job - *Irène Durand*
- **04 Lisp and AWS - *Andrew Lawson***
- 05 Backward Compatibility - *Anton Vodonosov*
- 06 Symbols as Namespaces - *Alessio Stalla*
- 07 FoCus - *Didier Verna*

Lisp & AWS

An Experience

Andrew Lawson
RavenPack International S.L.

Big Data

- Buzzwords
- Big Data means big infrastructure
- Which is difficult to manage and utilize
- But it is the future, until the next big thing

RavenPack

- Convert unstructured data to structured data
- Mainly for the financial industry
- News and social media
- Which means databases in the 10s of teras

The Traditional Database

- A convenient single-point of worry
- The DBA has control most of the time
- But also a single point of failure
- And easily saturated

Amazon Web Services

- Managed by the lovely people at Amazon
- Well documented, but still full of surprises
- Support is good though
- All the power that we could possibly need
- At a price

Our Problem

- If we want to launch a new version of the product
- We have to reanalyze our entire corpus
- Saturate the DB, 6 weeks
- Assuming that no bugs are found …
- And some of that machinery is idle 10-months a year

In AWS Land

- We use many services
    - S3, DynamoDB, EC2, etc etc …
- We’ve had to rewrite large parts of the system
- In Lisp, so quickly
- And learn … a lot
- The API is documented, but not all behavior is

Lisp Means …

- For S3, we can use Zach’s ZS3 library
- For everything else, we wrote it
- It’s web, lots of JSON
- Which is really sort-of-s-expression-ish, if you squint

DynamoDB Looks Like

```
(defmethod dynamodb-put-item ((context aws-context) table-name items)
  (let* ((target "DynamoDB_20120810.PutItem")
         (key-json-list
          (loop
             for (key type value) in items
             collect `(,(jsym key)
                        ,(cons (symbol->dtype type)
                               ;; All types must be supplied in strings
                               (typecase value
                                 (string value)
                                 (t (princ-to-string value)))))))
         (json-list `((:*TABLE-NAME . ,table-name)
                      (:*ITEM ,@key-json-list)
                      (:*RETURN-CONSUMED-CAPACITY . "TOTAL")
                      (:*RETURN-ITEM-COLLECTION-METRICS . "SIZE")
                      (:*RETURN-VALUES . "ALL_OLD"))))
    (let ((results (run-dynamodb-request context target json-list table-name)))
      ;; Returns two values. The first is t if a line was overwritten, nil if
      ;; the put did not overwrite a line.
      (values
       (when (assoc :*ATTRIBUTES results) t)
       results))))
```

Free Your Lisp

- We have lots of worker images
- Each containing a full Lisp system
- The only central resource is a metadata DB
- And that’s only accessed to distribute jobs
- Otherwise the workers access S3 and DynamoDB
- Which seem to scale more or less linearly

Winning

- Lisp is great for interactive development
- But for developing AWS interfaces it wins big
- Because we're just throwing JSON to see what sticks
- And that's easily manipulated at the REPL
- We translate AWS's errors to conditions automatically, so we even get nicely-formatted failures

But ..

HTTP Keep-Alive

- Don't forget to enable it, and you may have to implement it
- Your Ops team will thank you
- And so will the routers
- And you'll get fewer connection problems

```
(with-resource (socket (socket-pool aws-context))
  (do-some-aws-call)
  …)
```

Spot Instances

- Save money, bid for instances when you need them
- Then they take them away, when some big company wants to calculate their profits
- Inconvenient

Spot Instances (2)

- Distribute your machines across zones. The market is fairly zone-specific
- Make sure that your machines start quickly. Take advantage of the time that they give you
- Lisp wins again: Pre-prepared images downloaded from S3 on machine boot

DynamoDB Throughput

- You have to set it by hand
- High enough not to affect your systems
- Low enough not to pay a premium
- Major hassle … we want to automate this one day

So…

- We can run faster
- We don't need to go anywhere near our realtime systems
- We only pay when we need to do some work
- The hum of hundreds of Lisp servers can be heard for miles, it sounds like … victory

And …

We are always looking for good people

Lisp
Python
Java
Financial Mathematicians

http://ravenpack.com

Or talk to me

- 01 QTools UI - *Nicolas Hafner*
- 02 Contributing to the CL Ecosystem - *Daniel Kochmański*
- 03 A Job - *Irène Durand*
- 04 Lisp and AWS - *Andrew Lawson*
- **05 Backward Compatibility - *Anton Vodonosov***
- 06 Symbols as Namespaces - *Alessio Stalla*
- 07 FoCus - *Didier Verna*

Backward compatibility of libraries (case study in Common Lisp)

Anton Vodonosov

May 2016

There is a widespread belief, that if you change you library API in an incompatible way, all you need to do to protect clients is to upgrade the major version number:

```
  (asdf:defsystem :my-lib
    :version "1.1.1"
    ...)
```

becomes

```
 (asdf:defsystem :my-lib
    :version "2.1.1"
    ...)
```

That's not enough. Let's consider a dependency tree of an application which uses hunchentoot and postmodern:

```
my-application
    hunchentoot
        bordeaux-threads
            alexandria
        chunga
            trivial-gray-streams
        cl+ssl
            bordeaux-threads
                alexandria
            cffi
                alexandria
                babel
                    alexandria
                    trivial-features
                trivial-features
                uiop
            flexi-streams
                trivial-gray-streams
            trivial-garbage
            trivial-gray-streams
            uiop
```

```
     cl-base64
    cl-fad
        alexandria
        bordeaux-threads
            alexandria
    cl-ppcre
    flexi-streams
        trivial-gray-streams
    md5
    rfc2388
    trivial-backtrace
    usocket
postmodern
    bordeaux-threads
        alexandria
    cl-postgres
        md5
    closer-mop
    s-sql
       cl-postgres
           md5
```

We can see that many libraries (alexandria, md5, cl-postgres, flexi-streams, trivial-gray-streams, bordeaux-threamds, and so on) appear in the dependency tree several times, used by different clients:

```
my-application
    hunchentoot
        md5
    postmodern
        cl-postgres
            md5
```

Lets imagine we refactor md5 functions in an incompatible way and release it as md5 3.0.0.

Then cl-postgress adopts new md5

```
    (defsystem :cl-postgres
      :depends-on ((:version :md5 "3.0.0")
      ...
```

while hunchentoot still depends on old md5.

The application becomes broken as soon as we use new cl-postgres, because the dependency tree now includes two versions of md5.

```
    my-application
        hunchentoot
            md5         ;; version 2.x.x
        postmodern
            cl-postgres
                md5  ;; version 3.0.0
```

And the two versions can not be loaded simultaneously, because they share the same package name:

```
    (cl:defpackage #:md5
        ...
```

THE MAIN POINT:

If we want to save our clients from trouble, we need to ensure the new version of our library can be loaded simultaneously with the old version.

THE MAIN POINT:

If we want to save our clients from trouble, we need to ensure the new version of our library can be loaded simultaneously with the old version.

In Common Lisp this means we at least should use new package for new API. For example

```
    (cl:defpackage #:md5.v3
        ...
```

Another nuance.

IMHO the notion of major version number is redundant and unnecessary. If that's a new API let's name its ASDF system differently:

Instead of

```
(asdf:defsystem :my-lib
  :version "2.1.1"
  ...)
```

use

```
(asdf:defsystem :my-lib2
  ...)
```

Moreover.

It's the only way suitable for Common Lisp.

Major version numbers would only work if clients of old API specified dependencies with major version too:

```
(defsystem :hunchentoot
  :depends-on ((:version :md5 "2.0.0")
  ...

(defsystem :cl-postgres
  :depends-on ((:version :md5 "3.0.0")
  ...
```

But people don't do this. Actually:

```
(defsystem :hunchentoot
  :depends-on (:md5
  ...
```

Also, Quicklisp doesn't support several versions of the same ASDF system in one dist.

And it's more in the spirit of functional programming to avoid usage of the same name for different things.

CONCLUSION

- If you change you API incompatibly, allow implementations of the old API and of the new API to be used simultaneously. For CL this means at least use new package for new implementation.
- If you ASDF system doesn't include the old API, give that ASDF system new name.

Or...

don't break backward compatibility at all

Very often it's trivial to maintain backward compatibility.

E.g. instead of changing function parameters, leave the old function untouched (maybe deprecated) and introduce new function.

- 01 QTools UI - *Nicolas Hafner*
- 02 Contributing to the CL Ecosystem - *Daniel Kochmański*
- 03 A Job - *Irène Durand*
- 04 Lisp and AWS - *Andrew Lawson*
- 05 Backward Compatibility - *Anton Vodonosov*
- **06 Symbols as Namespaces - *Alessio Stalla***
- 07 FoCus - *Didier Verna*

SYMBOLS AS NAMESPACES ON ABCL

- Lisp is ultimately all about lists and **symbols**
- Symbols name things:
  - Have a string name (for reading and printing them)
  - Have properties, built-in and user-defined

SYMBOLS AS NAMESPACES ON ABCL

- Early Lisps: global namespace for symbols
  - Different softwares, same Lisp image = conflicts!
- **Packages** disambiguate: (not (eq ‘foo:bar ‘baz:bar))
- Packages:
  - Have a string name
  - Have built-in properties
  - Live in a global map and don’t exist outside of it

SYMBOLS AS NAMESPACES ON ABCL

- Common Lisp: global namespace for packages
  - Different softwares, same Lisp image = conflicts!
- Notice a pattern?
- Idea: **conflate symbols and packages**
  - Symbols have a string name, properties, and **can contain other symbols**
  - :foo:bar:baz

SYMBOLS AS NAMESPACES ON ABCL

- **Compatibility with CL is preserved**
  - Packages remain as a facade over symbols
  - Root symbol is the home of keywords
    - Nice :keyword syntax out of the box
  - Packages are symbols in :top-level-packages
    - Too little time to explain, ask me why!
  - ANSI test suite has same failures as regular ABCL

SYMBOLS AS NAMESPACES ON ABCL

- Why?!?
- Conceptual purity (one less concept)
- Nicely map to hierarchical names
  - file systems 'home:alessio:symbols-as-namespaces.pdf
    - we could get rid of pathnames too!
  - Foreign languages 'java.lang:String:valueOf

SYMBOLS AS NAMESPACES ON ABCL

- Bonus: symbol aliasing (for package nicknames)
- Malus: delete-package is a mess (should be deprecated in favor of unintern + GC)
- TODO
  - Further verify ANSI CL conformance
  - Polish (pun intended!)
  - Port to other Lisps (verify feasibility)
  - Standardize with a specification (CDR?)

- 01 QTools UI - *Nicolas Hafner*
- 02 Contributing to the CL Ecosystem - *Daniel Kochmański*
- 03 A Job - *Irène Durand*
- 04 Lisp and AWS - *Andrew Lawson*
- 05 Backward Compatibility - *Anton Vodonosov*
- 06 Symbols as Namespaces - *Alessio Stalla*
- **07 FoCus - *Didier Verna***

FoCus

Format Customizations

Didier Verna

@didierverna
didier@didierverna.net

ELS 2016 – Lightning Talk

Make `format` customizable / extensible

- `Declt` has a lot of this:
  ```
  (format t "~A" (escape str))
  ```
- I would like a custom directive:
  ```
  (format t "~`" str)
  ```
- But all I can do is this:

```
(format t
  "~/net.didierverna.declt::escape/"
  str)
```

FoCus: format customizations

- Wrapper around standard `format[ter]`
- Fully programmable case-sensitive directives
- *format-table*: similar to readtables

- Optional compile-time behavior
- Provides `in-format-table`
- Requires `asdf-flv`

Get it

- Lastest (Earliest) official version: 1.0 “Kokyu Ho”
- `http://github.com/didierverna/focus`