Gambit Modules

Frédéric Hamel

DIRO
University of Montreal

October 12, 2019

1. Our Motivation

- Code sharing between different users
- Code distribution
- Compiling and loading compiled code on the fly to nodes of a distributed system

2. Termite Scheme

- Created by Guillaume Germain
- A distributed programming language in Gambit
- Was inspired by the Erlang programming language
- In Gambit, most objects are serializable

2.1 Termite Scheme and Modules

Transmitting module

For correct operation, modules need to have a globally unique name (module-ref)

3. Modularization Approaches

Definition

A module is component of a system and is composed of variables, procedures and macros

Different kind of Scheme modules

- Old-fashionned modules
- Primitive modules
- R7RS compatible modules

3.1 Old-fashionned modules

```
;; hello.scm
(define (hi name)
  (display
    (string-append "hello " name "!"))
  (newline))
```

```
;; main.scm
(load "hello.scm")
(hi "marc")
```

Flaws

- Codes duplication if a module is loaded more than once
- Loading a module requires knowing its location on the filesystem which hinder code sharing among programs and users
- The global environment is polluted with all the definitions inside the module

3.2 Primitive Modules

Special forms

- `(##import <module-ref>)`
- `(##supply-module <module-ref>)` and `(##demand-module <module-ref>)`
- `(##namespace ("<ns>" ...))`

Advantages

- The import uses a search algorithm to find where the module is in the filesystem
- Multiple imports of the same module will result in only one load. There is no duplicate execution
- The use of namespaces removes the pollution of the global environment from the internal procedures of the module

3.3 Special Form ##namespace

The ##namespace form creates symbol aliases lexically

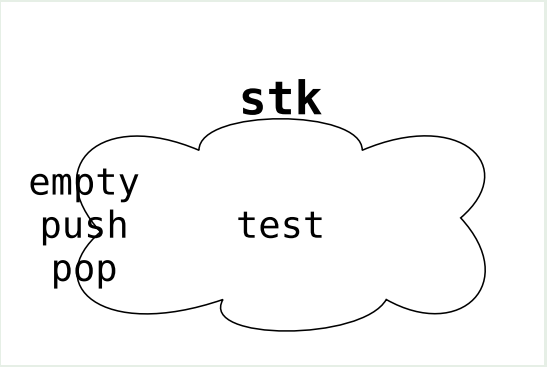
```
(##namespace ("foo#"))
;; <symbol-name> -> foo#<symbol-name>

(##namespace ("X#" A B))
;; only A and B are affected
;; A -> X#A
;; B -> X#B

(##namespace ("X#" (old new)))
;; old -> X#new
```

3.3.1 Primitive Modules (Example)

Stack Module Example

```
;; stk#.scm
(##namespace ("stk#" empty push pop))
```

```
;; stk.scm
(##supply-module stk)
(##namespace ("stk#"))
(##include "~~lib/gambit#.scm")
(##include "stk#.scm")

(define (empty) '())
(define (push x s) (cons x s))
(define (pop s) (cdr s))

(define (test)
  (if (equal? (push 1 (empty))
              '(1))
      "good!"
      "bad!"))
```




```
empty --> stk#empty
push  --> stk#push
pop   --> stk#pop
test  --> stk#test
```

3.3.2 Primitive ##import

```
(##import stk)

(define s (empty))

(set! s (push 1 s))
(set! s (push 3 s))
(set! s (push 5 s))

(set! s (pop s))
```

3.3.3 Primitive ##import Expansion

```
(##include "/some/path/stk#.scm")
(##demand-module stk)

(define s (empty))

(set! s (push 1 s))
(set! s (push 3 s))
(set! s (push 5 s))

(set! s (pop s))
```

3.4 Compatible R7RS Modules

Syntax

```
(define-library <library-name>
  <library-declarations...>)
```

Library declarations kind

- `(import <import-set> ...)`
- `(export <export-spec> ...)`
- `(begin <code>)`
- `(<include-kind> <filename> ...)`
  - `include` and `include-ci`
  - `include-library-declarations`
- `(cond-expand <ce-clause_1> <ce-clause_2> ...)`

3.4.1 Export Declaration

Syntax

```
(export <export-spec> ...}
```

<export-spec> kinds

- `(rename <identifier_1> <identifier_2>)`
- `<identifier>`

3.4.2 Import declaration

Syntax

```
(import <import-set> ...)
```

<import-set> kinds

- <library-name>
- (only <import-set> <identifier> ...)
- (except <import-set> <identifier> ...)
- (rename <import-set> (<id1> <id2>) ...)

3.5 Define Library Example

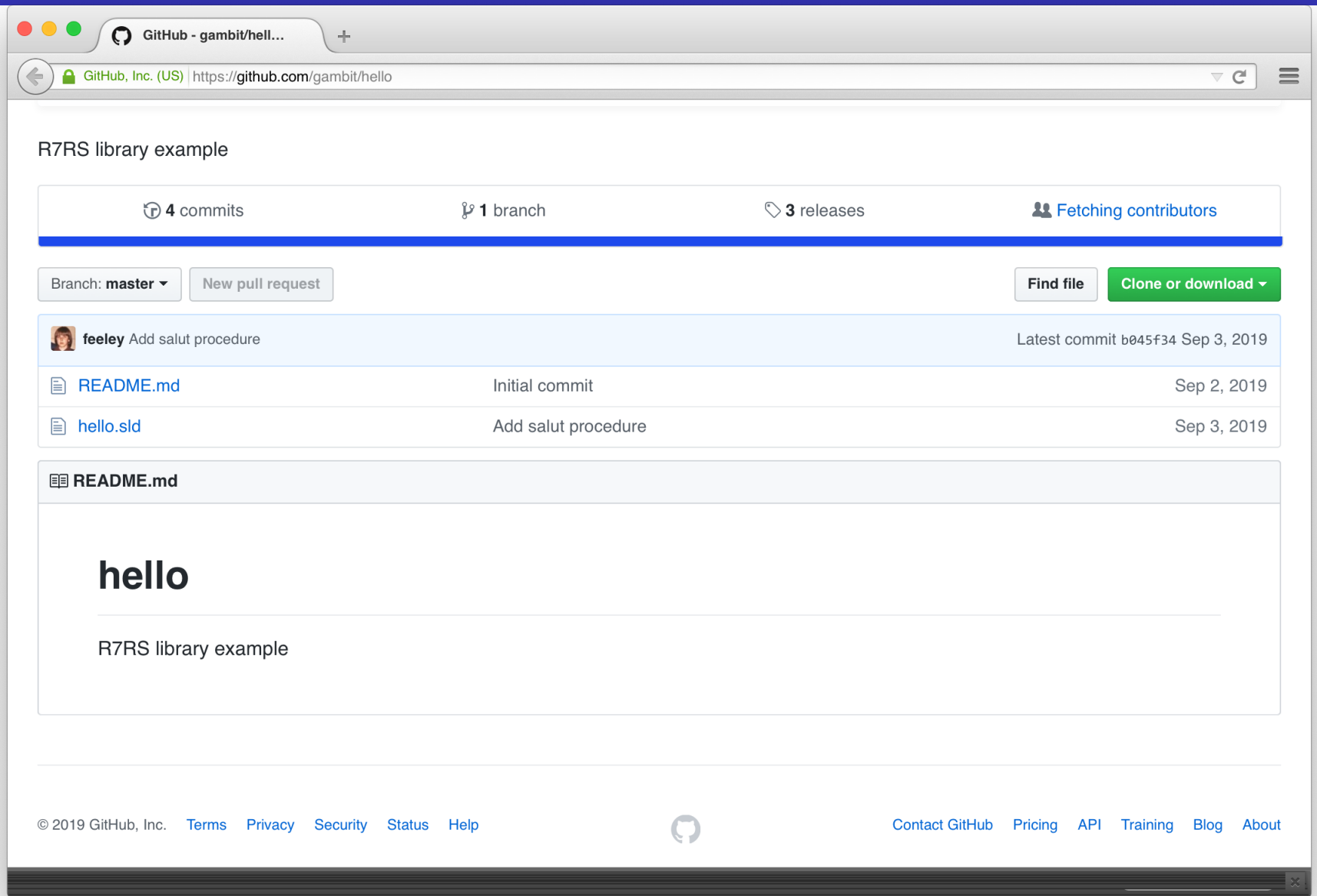
Module `hello`

```
(define-library (hello)

  (import (scheme base) (scheme write))

  (export hi salut)

  (begin

    (define (exclaim msg1 msg2) (display msg1) (display msg2) (display

    (define (hi name) (exclaim "hello " name))

    (define (salut name) (exclaim "bonjour " name))

    ;; it is best for a library to not have side-effects...
    #;(salut "le monde")))
```

3.5.1 Define Library Example (Expansion)

```
;; expansion of (define-library (hello) ...)
(##declare (block))
(##supply-module github.com/gambit/hello)
(##namespace ("github.com/gambit/hello#"))
(##namespace ("" ... define ...))
(##namespace ("" display write write-shared write-simple))
(define (exclaim msg1 msg2)
    (display msg1) (display msg2) (display "!\n"))
(define (hi name) (exclaim "hello " name))
(define (salut name) (exclaim "bonjour " name))
(##namespace (""))
```

3.6 Module Hello Example (Github Page)



3.7 Gambit Extensions to R7RS Modules

Gambit library declarations

- `(namespace <ns>)`
- Native compiler options
  - `(cc-options <c-flags> ...)`
  - `(ld-options <ld-flags> ...)`
  - `(ld-options-prelude <ld-flags> ...)`
- Native library options
  - `(pkg-config <libname> ...)`
  - `(pkg-config-path <path> ...)`

3.7.1 Namespaces Declaration

R7RS (scheme time) Library Example

```
(define-library (scheme time)

  (namespace "")

  (export
      current-jiffy
      current-second
      jiffies-per-second))
```

3.7.2 Platform Specific Options

SDL2 Library Example

```
(define-library (SDL2)

  (export SDL_Init SDL_Quit ...)

  (cc-options "-O3")
  (ld-options "-lGL")

  (pkg-config "sdl2")
  (pkg-config-path "/usr/local/X11/pkg")

  (include "sdl2.scm"))
```

4. Module Management

Modules installation is done with gsi (Gambit Software Installer!)

- Module name syntax
- Module search-order
- Installation/uninstallation
- Execution/compilation
- Testing modules
- Update

4.1 Module Name Syntax

Syntax of Module References (module-ref)

`<id1>/<id2>/<id3>/.../<idn>[@tag]`

- Hosted module, e.g. `github.com/gambit/hello`
  - `<id1>` = domain (with $\geq$ 1 dot)
  - `<id2>` = account
  - `<id3>` = repository
- Local module otherwise, e.g. `clock` and `my/app`

4.2 Module search-order

Search-order

- `~~lib` contains the system libraries
- `~~userlib` contains all user libraries
- `gsi -:search=/my/lib ...`

4.3 Installation/Uninstallation of Hosted Modules

```
% gsi -install github.com/gambit/hello
installing github.com/gambit/hello to ~/.gambit_userlib/
```

```
% gsi -install github.com/gambit/hello@1.0
installing github.com/gambit/hello@1.0 to ~/.gambit_userlib/
```

```
% gsi -uninstall github.com/gambit/hello
uninstalling github.com/gambit/hello from ~/.gambit_userlib/
```

```
% gsi -uninstall github.com/gambit/hello@1.0
uninstalling github.com/gambit/hello@1.0.0 from ~/.
    gambit_userlib/
ERROR -- Module github.com/gambit/hello@1.0 is not installed
```

4.4 Execution/Compilation

```
% gsi github.com/gambit/hello@1.0
hello world!
```

```
% gsi github.com/gambit/hello@2.0    # auto install!
bonjour le monde!
```

```
% gsc github.com/gambit/hello@2.0    # build module with gsc
% gsi github.com/gambit/hello@2.0    # execute compiled module
bonjour le monde!
```

```
% gsi /my/lib/ hello                 # add /my/lib to search-order
bonjour!
% gsi . hello                        # add . to search-order
hola!
```

4.5 Installing/Uninstalling Local Modules

```
% gsi -install A/B          # install module B
installing A/B to ~/.gambit_userlib/

% gsi -install B@1.0.0      # install module B@1.0.0
installing B@1.0.0 to ~/.gambit_userlib/

% gsi -uninstall B
uninstalling B from ~/.gambit_userlib/
```

4.6 Testing Modules

```
% gsi _digest/test                # execute _digest unit-tests
% gsi github.com/gambit/hello/test@2.1
```

Module (_test)

This module contains macros to help writing unit tests

- check-equal? and check-not-equal?
- check-eqv? and check-not-eqv?
- check-eq? and check-not-eq?
- check-=
- check-true and check-false
- check-not-false
- check-exn and check-tail-exn

4.7 Testing Module (Example)

```
(define-library (github.com/gambit/hello test)

  (import (.. hello)) ;; relative import (preserves the version)
  (import (_test))    ;; for check-equal? and check-tail-exn
  (import (gambit))   ;; for lambda, with-output-to-string, and
                      ;; wrong-number-of-arguments-exception?

  (begin

    (check-equal? (with-output-to-string (lambda () (hi "you")))
                  "hello you!\n")

    (check-equal? (with-output-to-string (lambda () (salut "hi")))
                  "bonjour hi!\n")

    (check-tail-exn wrong-number-of-arguments-exception?
                    (lambda () (hi)))

    (check-tail-exn wrong-number-of-arguments-exception?
                    (lambda () (hi "foo" "bar")))))
```

4.8 Update

```
% gsi -update github.com/gambit/hello
updating github.com/gambit/hello in ~/.gambit_userlib/
```

```
% gsi -update B          # updating module B
updating B in ~/.gambit_userlib/
```

5. Demo

- hello/demo
- Xlib/demo
- Clock (termite-clock)

5.1 Xlib Demo