Module Issues

Mike Sperber

What are Modules?

Matthew: separate compilation, managing universal namespace, importing implementations as opposed to importing interfaces

Will: putting pieces together without name clashes, hierarchical

Mike: interchangeable parts

Marc: encapsulation, managing namespace

Manuel: to organize programs in files, initialize things, name entry points, etc.

Kent: all of the above except maybe files; namespace management, analyzability, compile-time link

Bigloo

```
(module m1
   (export foo bar))
(define foo 23)
(define bar 42)

(module m2
   (export baz)
  (import m1))
(define baz (+ foo bar))
```

Chez Scheme

```
(module m1 (foo bar)
  (define foo 23)
  (define bar 42))

(module m2 (baz)
  (import m1)
  (define baz (+ foo bar)))
```

MzScheme

```
(module m1 mzscheme
  (provide foo bar)
  (define foo 23)
  (define bar 42))

(module m2 mzscheme
  (provide baz)
  (require m1)
  (define foo 23)
  (define bar 42))
```

Scheme 48

```
(define-interface m1-interface
  (export foo bar))
(define-structure m1 m1-interface
  (open scheme)
  (begin
    (define foo 23)
    (define bar 42)))

(define-interface m2-interface
  (export baz))
(define-structure m2 m2-interface
  (open scheme
        m1)
  (begin
    (define baz (+ foo bar))))
```

Making Module Definitions Available

Bigloo

m1.scm:

```
(module m1
    (export foo bar))
(define foo 23)
...
```

Module access file:

```
((m1 "m1.scm")
 ...)
```

Making Module Definitions Available

Chez Scheme

m1.scm

```
(module m1 (foo bar)
  (define foo 23)
  ...)
```

```
(load "m1.scm")
```

Making Module Definitions Available

MzScheme

m1.ss:

```
(module m1 mzscheme
  (provide foo bar)
  (define foo 23)
  ...)
```

```
(require (lib "m1.ss" "libmike"))
```

Making Module Definitions Available

Scheme 48

m1-config.scm:

```
(define-interface m1-interface
  (export foo bar))
(define-structure m1 m1-interface
  (open scheme)
  (begin
    (define foo 23)
    ...))
```

REPL:

```
,config ,load m1-config.scm
```

Separate Configuration Language

Bigloo

```
(module m1
   (export foo bar))
(define foo 23)
(define bar 42)

(module m2
   (export baz)
  (import m1))
(define baz (+ foo bar))
```

Separate Configuration Language

Chez Scheme

```
(module m1 (foo bar)
  (define foo 23)
  (define bar 42))
```

```
(module m2 (baz)
  (import m1)
  (define baz (+ foo bar)))
```

Separate Configuration Language

MzScheme

```
(module m1 mzscheme
  ____________________
  (provide foo bar)
  (define foo 23)
  (define bar 42))

(module m2 mzscheme
  ____________________
  (provide baz)
  (require m1)
  (define foo 23)
  (define bar 42))
```

Separate Configuration Language

Scheme 48

```
(define-interface m1-interface
  (export foo bar))
(define-structure m1 m1-interface
  (open scheme)
  (begin
    ____________________
    (define foo 23)
    (define bar 42)))
```

Separate Configuration Language

Scheme 48

```
(define-interface m2-interface
  (export baz))
(define-structure m2 m2-interface
  (open scheme
        m1)
  (begin
    (define baz (+ foo bar))))
```

Neat Stuff with Local Modules

```
(define-syntax module*
  (syntax-rules ()
    ((_ (id ...) form ...)
     (begin
       (module* tmp (id ...) form ...)
       (import tmp)))
    ((_ name (id ...) form ...)
     (module name (id ...) form ...))))
```

Neat Stuff with Local Modules

```
(define-syntax import*
  (syntax-rules ()
    ((_ M) (begin))
    ((_ M (new old))
     (module* (new)
       (define-alias new tmp)
       (module* (tmp)
         (import M)
         (define-alias tmp old))))
    ((_ M id)
     (module* (id) (import M)))
    ((_ M spec0 spec1 ...)
     (begin
       (import* M spec0)
       (import* M spec1 ...)))))
```

Ambiguities with Internal Definitions

```
(let ((x 10))
  (let-syntax ((foo (syntax-rules (x)
                      ((foo x d)
                       (define d 'outer))
                      ((foo n d)
                       'inner))))
    (let ()
      (foo x y)
      (define z (foo x y))
      (define x 5)
      (list y z))))
```

Chez Macro Expansion Algorithm

Chez Scheme processes body forms from left to right and adds macro definitions to the compile-time environment as it proceeds. [...] define rhs expressions are expanded, along with the body expressions that follow the definitions, only after the set of definitions is determined.

Local Import vs. Hygiene

```
(module m (foo bar)
  (define foo 'm))

(define-syntax baz
  (syntax-rules ()
    ((baz) (import m))))

(let ((foo 'bar))
  (baz)
  foo)
=> m
```

"Implicit" Exports

```
(define-syntax foo
  (syntax-rules ()
    ((foo) a)))
(define a 5)
...
; export foo
```

"Implicit" Exports

```
(define-syntax foo
  (syntax-rules ()
    ((foo ?x) (?x a))))
(define a 5)
...
; export foo
... (foo quote) ...
```

Phase Separation

```
(module syntax-helpers mzscheme
  (provide syntax2list)
  (define syntax2list
    (lambda (x)
      (syntax-case x ()
        [() '()]
        [(a . d) (cons #'a (syntax2list #'d))]))))

    (module m mzscheme
      (require-for-syntax syntax-helpers)
      (define-syntax foo
        (lambda (x)
          (syntax-case x ()
            ((_)
             ... (syntax2list ...) ...)))))
```

Export Annotation

Scheme 48:

```
(define-interface foo-interface
  (export v
          (m :syntax)
          (write-string
           (proc (:string
                  &opt :value
                       :exact-integer :exact-integer)
                 :unspecific))))
```

Modules vs. Files

- Is it possible to define more than one module in a single file?
- Is it possible to define a module in a single file?
- Is it possible to have an import refer to different modules depending on context?
- Does the association between module identifiers (in whatever format) and modules always happen as an implicit part of module definition, or is it specified separately?

What's an Import?

Bigloo

```
(import m)
```

Chez Scheme

```
(import m)
```

MzScheme

```
(require (lib "m.ss" "libmike"))
```

Scheme 48

```
(open m)
```

The Missing Link

- imports are of interfaces, not modules
- linking is implicit, like C
- explicit links only needed for conflicts

Separate vs. Independent Compilation

```
(define-structure foo (export (m :syntax))
  (define-syntax m
    (syntax-rules ()
      ((m) 1))))
```

Exported Macros as Part of the Interface

```
  (define-interface promises-interface
    (export force
            (define-syntax delay
              (syntax-rules ()
                ((delay ?exp)
                 (make-promise
                  (lambda () ?exp)))))))

(define-module promises promises-interface
  ...
  (begin
    (define make-promise ...)
    (define force ...)))
```

Miscellaneous Issues

- interactive toplevel
- **`eval`**
- tying module names to file names
- small executables
- optional initialization
- initialization order