Title

`require-extension`

Authors

Felix L. Winkelmann and D.C. Frost

Status

This SRFI is currently in *final* status. Here is an explanation of each status that a SRFI can hold. To provide input on this SRFI, please send email to `srfi-55@srfi.schemers.org`. To subscribe to the list, follow these instructions. You can access previous messages via the mailing list archive.

- Received: 2004-05-03
- Draft: 2004-06-07--2004-08-05
- Revised: 2004-06-16
- Fianlized: 2004-11-05

Abstract

This SRFI specifies an extremely simple facility for making an extension or library available to a Scheme toplevel environment.

Rationale

The `(requires ...)` clause of SRFI 7 (*Feature-based program configuration language*) is one possible way for a program to declare that it requires certain features or extensions in order to run. There are two limitations with this facility: first, an implementation of SRFI 7 is not actually required to do anything as the result of encountering a `(requires ...)` clause. At the time of this writing, some implementations do in fact load and make available the specified extension in this case, but some do not. Second, the SRFI 7 language is most appropriate as an annotation to program text. It is not designed for interactive use.

This SRFI therefore defines a simple mechanism specifically for requiring that extensions be made immediately available to subsequent code at compile-time or runtime, as appropriate. In particular, the intent is that a trivial, portable means exist for loading SRFI functionality within a program or interactive session; but the mechanism described here is general enough to be used for other types of extensions, at the implementation's discretion.

Most implementations of Scheme include a form very similar to `require-extension`. This SRFI can therefore be viewed, in the context of those Schemes, as merely a standard naming convention.

It is possible for an implementation's design, contrary to common practice, to conflict with the semantics of `require-extension`. Such an implementation would provide an alternative means of specifying requirements. This SRFI does not aim to be ubiquitous, only to capture current idiom.

Specification

The `require-extension` form is used to make an extension available in the toplevel scope in which it appears. The definition of a "toplevel scope" and the exact meaning of what it means to make an extension available in one is implementation-defined, but we expect likely scopes will include the default scope in which program expressions are evaluated; the scope in which program expressions within a module are evaluated, as defined by a module system; and the interactive REPL ("read-eval-print loop"). When `require-extension` is used to make an extension available in a non-interactive context, it is implementation-defined whether the extension will be available to code lexically preceding the `require-extension` form in the same scope, but it should be available to code in the same scope lexically succeeding the `require-extension` form. An implementation should default to signalling a warning or an error in the event that a requested extension cannot be made available. An implementation is encouraged but not required to signal a warning or an error if the user attempts to access incompatible extensions simultaneously.

An implementation claiming to support this SRFI must support `require-extension` in at least one scope.

The syntax of `require-extension` is as follows:

```
(require-extension <clause> ...)
```

A clause has the form:

```
(<extension-identifier> <extension-argument> ...)
```

where `<extension-identifier>` is a symbol and the zero or more `<extension-argument>`s may be any Scheme values.

This SRFI defines only one `extension-identifier`, the identifier `srfi`, which implementations purporting to conform to this SRFI must support. The `extension-argument`s of a `srfi` clause may be any Scheme values, at an implementation's discretion, but an implementation must support nonnegative integer `extension-argument`s and should treat them as a directive to make the functionality of the indicated SRFIs available in the context in which the `require-extension` form appears. For example:

```
(require-extension (srfi 1))              ; Make the SRFI 1 List Library available
(require-extension (srfi 1 13 14))        ; Make the SRFI 1, 13 and 14 libraries available
```

Implementation

The implementation of `require-extension` is necessarily implementation-specific. Here is a (very simple) example implementation that is based on the optional R5RS `load` procedure:

```
;;;; Reference implementation for SRFI-55
;
; Requirements: SRFI-23 (error reporting)

(define available-extensions '())
```

```
(define (register-extension id action . compare)
  (set! available-extensions
    (cons (list (if (pair? compare) (car compare) equal?)
                id
                action)
          available-extensions)) )

(define (find-extension id)
  (define (lookup exts)
    (if (null? exts)
        (error "extension not found - please contact your vendor" id)
        (let ((ext (car exts)))
          (if ((car ext) (cadr ext) id)
              ((caddr ext))
              (lookup (cdr exts)) ) ) ) )
  (lookup available-extensions) )

(define-syntax require-extension
  (syntax-rules (srfi)
    ((_ "internal" (srfi id ...))
     (begin (find-extension '(srfi id) ...)) )
    ((_ "internal" id)
     (find-extension 'id) )
    ((_ clause ...)
     (begin (require-extension "internal" clause) ...)) ) )

; Example of registering extensions:
;
;   (register-extension '(srfi 1) (lambda () (load "/usr/local/lib/scheme/srfi-1.scm")))
```

Copyright

Editor: Mike Sperber
Last modified: Sun Jan 28 13:40:37 MET 2007