Issue
With-Readtable-Iterator

Forum

Common Lisp Document Repository (CDR)

Status

Draft.

References

- CLHS 2.1.1 Readtables
- X3J13 Issue #188
- `with-package-iterator`, `with-hash-table-iterator` (macros)

Category

Addition.

Edit History

02-Oct-2008 by Rittweiler (Draft)
03-Oct-2008 by Rittweiler (Updated)

Problem Description

Even though the ANSI Common Lisp standard provides simple getters for readtables (`get-macro-character`, `get-dispatch-macro-character`), the standard does not provide any means to efficiently get at all the macro characters and dispatch macro characters defined in a readtable.

The omission of any iteration facility for readtables makes readtables unnecessarily opaque, and keeps users from writing libraries that try to deal with readtables in a general way. For example, the author discovered that this lack of an iteration form is the main obstacle to writing an otherwise portable library that establishes an organizational namespace for readtables akin to the namespace that is provided for packages.

Proposal (With-Readtable-Iterator:Add-Generator)

Add a macro `with-readtable-iterator` that establishes a generator in its scope; each invocation of this generator returns a macro character or a dispatch macro character from the readtable the generator was established for—along side some additional information.

A detailed specification of `with-readtable-iterator` can be found in the appendix of this document.

Rationale

The proposed macro `with-readtable-iterator` represents a general iteration facility for readtables that can be used to implement a variety of iteration forms. The proposal is closely modelled on the macros `with-hash-table-iterator` and `with-package-iterator`. The semantics of `with-readtable-iterator` should thus be intuitive to any experienced Common Lisp programmer.

Notes

The proposal does deliberately say nothing about the home-package of the symbol `with-readtable-iterator`. However, implementors are encouraged to export this symbol from their extensions package (often called "`EXT`") or another appropriate package—unless a later CDR document specifies a more explicit location.

Current Practice

No implementation the author is aware of provides a way to iterate through a readtable.

The author implemented the proposal for SBCL, and sent the relevant patches upstream; the patches are currently waiting to be integrated into mainline. Ariel Badichi implemented the proposal for CLISP, and is going to send his work upstream shortly. Stephen Compall did an implementation for Clozure CL which needs to be somewhat revised to fully conform to the proposal as presented in this document.

Cost to implementators

The macro `with-readtable-iterator` should be straightforwardly implementable. Extrapolating from actual experience, people—who were previously not acquainted with the relevant code sections—were able to implement it in a couple of hours.

Discussion

- Ariel Badichi proposed coalescing a generator's fourth return value (indicating if the returned character is a dispatch macro character) with its first return value (indicating if the generator is exhausted.)

  There is technically no reason that speaks against doing so; in fact, a generator could return one value less this way—which may lead to positive performance characteristics on register-anemic processor architectures.

  Stephen Compall and the author opposed such a change mostly for idiomatic reasons, as both `with-hash-table-iterator` and `with-package-iterator`, the generator-establishing macros specified by the ANSI standard, return a boolean exhaustion flag as first value. In particular, `with-package-iterator` does *not* coalesce the accessibility type (third return value) with the exhaustion flag (first return value.)
- The author notes that allowing `:terminating`, and `:non-terminating` as valid *macro-char-types* was considered, but rejected for reasons of simplicity. It is not apparent that there is a real necessity for supporting these out of the box.

Acknowledgements

The author wants to specially credit Ariel Badichi and Stephen Compall. In spirit of true hackerism, they promptly agreed to hack an early version of the proposal into the implementations of their choice, and provided valuable comments on revising this document.

Appendix

`with-readtable-iterator` [Macro]

`(name readtable &rest macro-char-types) declaration* form*`

⇒■ `results`

Arguments and Values

name A symbol.

readtable A form, evaluated once to produce a readtable designator.

macro-char-type
One of the symbols `:macro-char`, or `:dispatch-macro-char`.

declaration
A `declare` expression; not evaluated.

forms An implicit progn.

results The values of the *forms*.

Description

Within the lexical scope of the body *forms*, the *name* is defined via `macrolet` such that successive invocations of `(name)` will return the macro characters from the *readtable*, one by one but each one once. The order of the macro characters returned is implementation-dependent.

The variable *macro-char-types* controls which macro characters are returned:

`:macro-char`
All macro characters in *readtable* that are **not** dispatch macro characters.

`:dispatch-macro-char`
All dispatch macro characters in *readtable*.

Multiple occurrences of the same symbol are allowed in *macro-char-types*. If no explicit *macro-char-types* are supplied, both `:macro-char` and `:dispatch-macro-char` is assumed.

An invocation of `(name)` returns the following five values:

1. A generalized boolean that is true if a macro character is returned.
2. A macro character that is defined in *readtable*.
3. A reader macro function of the macro character returned.
4. A generalized boolean that is true if the macro character is a dispatch macro character.
5. An association list with the "sub-characters" of the dispatch macro character as keys, and their corresponding reader macro functions as values.

After all macro characters have been returned by successive invocations of `(name)`, only one value is returned, namely `nil`.

Consequences are undefined if the association list returned as fifth value is modified.

Consequences are undefined if *readtable* is modified in a way that might affect an ongoing traversal operation. Yet conforming programs may modify the current macro character in the readtable under traversal by means of `set-macro-character`, and `set-dispatch-macro-character`.[1]

It is unspecified what happens if any of the implicit interior state of an iteration is returned outside the dynamic extent of the `with-readtable-iterator` form such as by returning some closure over the invocation form.

Any number of invocations of `with-readtable-iterator` can be nested, and the body of the innermost one can invoke all of the locally established macros, provided all those macros have distinct names.

Examples

```
(let (macro-chars)
  (with-readtable-iterator (next-entry nil)
    (loop (multiple-value-bind (more? ch) (next-entry)
            (unless more? (return (sort macro-chars #'char>)))
            (push ch macro-chars)))))
⇒ (#\‘ #\; #\, #\) #\( #\’ #\# #\")

(with-readtable-iterator (next-entry nil :dispatch-macro-char)
  (let* ((disp-table (nth-value 4 (next-entry)))
         (sub-chars  (mapcar #'car disp-table)))
    (sort (remove-if-not #'graphic-char-p sub-chars) #'char>)))
⇒ (#\| #\\ #\X #\S #\R #\P #\O #\C #\B #\A #\= #\< #\: #\.
   #\- #\+ #\* #\) #\( #\’ #\# #\Space)
```

Exceptional Situations

Signals an error of type `program-error` if a *macro-char-type* is supplied that is not recognized by the implementation.

An error of type `type-error` is signaled if *readtable* does not evaluate to a readtable designator.

See Also

Readtables (CLHS 2.1.1), Traversal Rules and Side Effects (CLHS 3.6)

Notes

Implementations may extend the syntax of `with-readtable-iterator` by recognizing additional macro character types.

[1] This does not entail the permission to modify the standard readtable; CLHS 2.1.1.2 prevails.