Package-Local Nicknames

Gleefre

July 6, 2024

Contents

1 Specification

1.1 Introduction

This is a specification for the `Package-Local Nicknames` extension in Common Lisp.

1.1.1 Rationale

Package-local nicknames make it possible to use short and easy-to-use names without potentially introducing name conflicts as can happen with usual nicknames.

1.1.2 Current state

Package-local nicknames are implemented in some form in `SBCL`, `CCL`, `ECL`, `Clasp`, `ABCL`, `Allegro CL`, `LispWorks`. There is also a pending MR for the `CLISP` implementation.

Unfortunately, there are multiple inconsistencies between implementations. All of them lose *print-read consistency* to some extent, and there are multiple edge cases that aren't always implemented correctly or in the same way.

1.1.3 Goal

The purpose of this document is to standardize the `Package-Local Nicknames` extension and to address some existing issues.

[TODO] This CDR also aims to provide an extensive test suite for the extension.

1.2 Description

A *package-local nickname* (or a *local nickname*) defined in some *designated package* has the same effects as a usual *package nickname* (later referred to as a *global nickname*), except that these effects only apply when `*package*` is bound to that *designated package*.

This means that a call to `find-package` with a *local nickname* that is defined in the *current package* returns the package nicknamed by this nickname. This also affects all implied calls to `find-package`, including those performed by the Lisp reader.

In addition, to maintain *print-read consistency*, the Lisp printer is affected by *local nicknames* defined in the *current package*. For details see Issue 2.

A *local nickname* is allowed to shadow a *package name* or a *global nickname*, except for the names `#:CL`, `#:COMMON-LISP` and `#:KEYWORD` which must always refer to their packages. The consequences of adding *local nicknames* to the packages `#:COMMON-LISP` and `#:KEYWORD` are also undefined.

1.3 API

1.3.1 defpackage

1. Description The `defpackage` options are extended to include the *local-nicknames-option*:

`local-nicknames-option ::= (:local-nicknames (nickname package)*)`

Each pair specifies a *local nickname* `nickname` for the corresponding `package`.

This option may appear more than once.

2. Arguments and Values: `nickname` — a *string designator*.

   `package` — a *package designator*.

3. Exceptional situations An error of type `package-error` is signaled when a package designated by the `package` does not exists.

   Name conflict errors are handled by the underlying calls to `add-package-local-nickname`.

   See add-package-local-nickname: exceptional situations.

4. Implementation dependent The consequences are undefined when a *local nickname* is specified for thex package that is being defined. (See Issue 4.)

   The consequences are undefined when supplied *local nicknames* are at variance with the current state of the package. An implementation might choose to remove all existing *local nicknames* at the beginning of each redefinition of the package.

1.3.2 make-package

1. Description (**Contains proposals**: see Issue 6.)

   The `make-package` lambda list is extended to include an additional keyword argument `:local-nicknames`:

   `local-nicknames ::= ((nickname package)*)`

   `local-nicknames` specifies zero or more *local nicknames* to be defined in the new *package*.

2. Arguments and Values: `local-nicknames` — a *list* of pairs of form `(nickname package)`. The default is an *empty list*.

   `nickname` — a *string designator*.

   `package` — a *package designator*.

3. Exceptional situations An error of type `package-error` is signaled when a package designated by the `package` does not exist.

   Name conflict errors are handled by the underlying calls to `add-package-local-nickname`.

   See add-package-local-nickname: exceptional situations.

4. Implementation dependent The consequences are undefined when a *local nickname* is specified for the package that is being defined. (See Issue 4.)

1.3.3 add-package-local-nickname

```
(add-package-local-nickname nickname actual-package &optional designated-package)
  => designated-package-object
```

1. Arguments and Values `nickname` — a *string designator*.

   `actual-package` — a *package designator*.

   `designated-package` — a *package designator*. The default is the *current package*.

   `designated-package-object` — a *package*.

2. Description Defines a *package-local nickname* `nickname` for the `actual-package` in the `designated-package`.

   [Also see Issue 1.] Returns the package designated by the `designated-package`.

   If the *nickname* is already defined, checks that it is defined for the package designated by the `actual-package`.

3. Exceptional situations An error of type *package-error* is signaled when a package designated by the `actual-package` or the `designated-package` does not exist.

   If the `nickname` is one of the names `#:CL`, `#:COMMON-LISP` or `#:KEYWORD`, an error of type *package-error* is signaled.

   If the `nickname` is already defined to be a *local nickname* for a package different from the `actual-package`, a *correctable* error of type *package-error* is signaled.

4. Implementation dependent The consequences are undefined when the `designated-package` designates the `#:COMMON-LISP` package or the `#:KEYWORD` package.

   (**Contains proposals**: see Issue 5.)

   If the `nickname` shadows the *package name* or one of the *global nicknames* of the `designated-package`, a style warning might be issued.

1.3.4 remove-package-local-nickname

```
(remove-package-local-nickname old-nickname &optional designated-package)
  => nickname-removed-p
```

1. Arguments and Values `old-nickname` — a *string designator*.

   `designated-package` — a *package designator*. The default is the *current package*.

   `nickname-removed-p` — *generalized boolean*.

2. Description If `old-nickname` is defined to be a *local nickname* in the `designated-package`, it is removed.

   [Also see Issue 1.] Returns *true* if it removes a nickname, and `NIL` otherwise.

3. Exceptional situations An error of type *package-error* is signaled when a package designated by the `designated-package` does not exist.

1.3.5 package-local-nicknames

```
(package-local-nicknames package-designator)
  => local-nicknames-alist
local-nicknames-alist ::= ((nickname . package)*)
```

1. Arguments and Values `package-designator` — a *package designator*.

   `local-nicknames-alist` — an *alist*.

   `nickname` — a *string*.

   `package` — a *package*.

2. Description Returns an *alist* describing *local nicknames* defined in the package designated by the `package-designator`.

3. Exceptional situations An error of type `package-error` is signaled when a package designated by the `package-designator` does not exist.

4. Notes The returned *alist* must be safe to be modified by the user.

1.3.6 package-locally-nicknamed-by-list

```
(package-locally-nicknamed-by-list package-designator)
  => packages-list
```

1. Arguments and Values `package-designator` — a *package designator*.

   `packages-list` — a *list* of *package* objects.

2. Description Returns a *list* of packages that have a *local nickname* defined for the package designated by the `package-designator`.

3. Exceptional situations An error of type `package-error` is signaled when a package designated by the `package-designator` does not exist.

4. Notes The returned *list* must be safe to be modified by the user.

1.4 Affected symbols

1.4.1 defpackage

See defpackage.

1.4.2 make-package

See make-package.

1.4.3 find-package

(**Contains proposals**: see Issue 3, Issue 8.)

When the argument to `find-package` is a *local nickname* defined in the *current package*, it returns the package nicknamed by this nickname.

This also affects all implied calls to `find-package`, including but not limited to those performed by the lisp reader as well as those performed by `defpackage`, `make-package`, `export`, `find-symbol`, `import`, `rename-package`, `shadow`, `shadowing-import`, `delete-package`, `with-package-iterator`, `unexport`, `unintern`, `in-package`, `unuse-package`, `use-package`, `do-symbols`, `do-external-symbols`, `do-all-symbols`, `intern`, `package-name`, `package-nicknames`, `package-shadowing-symbols`, `package-use-list`, `package-used-by-list`.

`add-package-local-nickname`, `remove-package-local-nickname`, `package-local-nicknames` and `package-locally-nicknamed-by` are also affected.

The only exception is the *tilde slash* directive of `format`, which should **not** use *local nicknames* from any package when looking up the specified symbol.

1.4.4 rename-package

When a package is renamed with `rename-package`, it retains all *local nicknames* it has defined, as well as all *local nicknames* by which it is nicknamed.

1. Implementation dependent (**Contains proposals**: see Issue 5.)

   If the *new-name* or one of the *new-nicknames* is shadowed by one of the *local nicknames* of the package being renamed, a style warning might be issued.

1.4.5 delete-package

When a package is deleted with `delete-package`, all *local nicknames* defined in that package are removed, as well as all *local nicknames* by which it is nicknamed.

This also means that a deleted package must not be available via calls to `package-locally-nicknamed-by-list` and `package-local-nicknames`.

1.4.6 format

See Issue 8.

1.4.7 *features*

If an implementation supports package-local nicknames, it should add symbols `:package-local-nicknames` and `:cdr-NN` (per CDR 14) to `*features*`.

1.5 Examples

[TODO]

2 ISSUES

2.1 Issue 1 (ADD-/REMOVE-PACKAGE-LOCAL-NICKNAME return values)

2.1.1 Description

Functions `add-package-local-nickname` and `remove-package-local-nickname` have inconsistent return values.

The first one always returns the *designated package*, while the second one returns *true* (a *generalized boolean*) that indicates whether a nickname was removed.

Furthermore, there is no consensus among existing implementations as to what this *generalized boolean* represents.

In comparison,the functions `use-package` and `unuse-package` always return `T`, and the functions `export` and `unexport` have the same behavior. There are also functions `unintern` and `delete-package` which return a *generalized boolean* indicating if the operation changed something, but their counterparts `intern` and `make-package`, unlike with local nicknames, create and return an *object* (a *symbol* or a *package*).

2.1.2 Examples

```
(defpackage #:foo (:use))
(defpackage #:bar (:use))

(add-package-local-nickname '#:nick '#:bar '#:foo)
; => #<PACKAGE "FOO">  (sbcl, ccl, ecl, acl, abcl, clasp, lispworks)
(add-package-local-nickname '#:nick '#:bar '#:foo)
; => #<PACKAGE "FOO">  (sbcl, ccl, ecl, acl, abcl, clasp, lispworks)
(remove-package-local-nickname '#:nick '#:foo)
; => T  (sbcl, ccl, ecl, acl, clasp)
; => #<PACKAGE "BAR">  (abcl)
; => NIL  (lispworks)
(remove-package-local-nickname '#:nick '#:foo)
; => NIL  (sbcl, ccl, ecl, acl, abcl, clasp, lispworks)
```

2.1.3 Current behavior

sbcl, ccl, ecl, acl, abcl, clasp, lispworks: `add-package-local-nickname` always returns *designated package*.

sbcl, ccl, ecl, acl, clasp: `remove-package-local-nickname` returns `T` if a nickname was removed, and `NIL` otherwise.

abcl: `remove-package-local-nickname` returns *true* if a nickname was removed, (more specifically, it returns the package nicknamed by the removed local nickname), and `NIL` otherwise.

lispworks: `remove-package-local-nickname` always returns `NIL`.

2.1.4 Proposal ALWAYS-T

`add-package-local-nickname` should always return `T`.

`remove-package-local-nickname` should always return `T`.

2.1.5 Proposal ELIMINATE-GENERALIZED-BOOLEAN

`add-package-local-nickname` should return the *designated package*.

`remove-package-local-nickname` should return `T` if a nickname was removed and `NIL` otherwise.

2.1.6 Proposal DESIGNATED-PACKAGE-IF-SUCCESSFUL

`add-package-local-nickname` should return the *designated package* if a new nickname was added, and `NIL` otherwise.

`remove-package-local-nickname` should return the *designated package* if a nickname was removed, and `NIL` otherwise.

2.1.7 Proposal NICKNAMED-PACKAGE-IF-SUCCESSFUL

`add-package-local-nickname` should return the *nicknamed package* if a new nickname was added, and `NIL` otherwise.

`remove-package-local-nickname` should return the previously *nicknamed package* if a nickname was removed, and `NIL` otherwise.

2.2 Issue 2 (PRINT-READ consistency)

2.2.1 Description

The Lisp reader uses `find-package` when reading a symbol, which is affected by the *local nicknames* of the *current package*. That means that to maintain **print-read** consistency when printing a symbol, a good *package prefix* must be used - such that calling `find-package` on it in the *current package* returns the *home package* of the symbol.

There are several situations to consider:

1. The symbol is *apparently uninterned*.

   In this case it is printed without the package prefix, using the uninterned symbol syntax `#:`.

2. The symbol is accessible in the *current package*.

   In this case it is printed without any package prefix.

3. The *name* or one of the *global nicknames* of the *home package* of the symbol is not shadowed by any *local nickname* defined in the *current package*.

   In this case that name or global nickname might be used as the package prefix.

4. There exists a *local nickname* defined in the *current package* for the *home package* of the symbol.

   In this case that local nickname might be used as the package prefix.

5. Both the *name* and all *global nicknames* of the *home package* of the symbol are shadowed by *local nicknames* of the *current package*, and there is no *local nickname* defined in the *current package* for the *home package*.

   It is not clear how the symbol is printed, see PROPOSALS.

2.2.2 Examples

```
(defpackage #:foo
  (:use)
  (:export #:+))

(defpackage #:bar
  (:use #:cl)
  (:local-nicknames (#:foo #:cl)))

(let ((*package* (find-package '#:bar)))
  (print 'foo:+))
; >> FOO:+  (sbcl, ccl, ecl, acl, abcl, clasp, lispworks)

;; In the package #:BAR symbol FOO:+ refers to CL:+

(defpackage #:foo-a (:use) (:export #:quux))
(defpackage #:foo-b (:use) (:export #:quux))

(defpackage #:bar
```

```
  (:use)
  (:local-nicknames (#:foo-a #:foo-b)
                    (#:foo-b #:foo-a)))

(let ((*package* (find-package '#:bar)))
  (print 'foo-a:quux))
; >> FOO-B:QUUX  (sbcl, ccl, abcl, lispworks)
; >> FOO-A:QUUX  (ecl, acl, clasp)

;; In the package #:BAR symbol FOO-A:QUUX refers to FOO-B:QUUX
```

2.2.3 Current behavior

sbcl, ccl, abcl, lispworks: When possible, a *local nickname* is used as a package prefix.

ecl, acl, clasp: *Local nicknames* are never used as a package prefix.

2.2.4 Proposal SHARPSIGN-DOT

In the 5th case the symbol is printed using the `#.` syntax:

```
#.(cl:let ((cl:*package* (cl:find-package "KEYWORD")))
    (cl:find-symbol "BAR" "FOO"))
;; or
#.(cl:let ((cl:*package* (cl:find-package "KEYWORD")))
    (cl:intern "BAR" "FOO"))
```

If `*read-eval*` is *false* and `*print-readably*` is *true*, an error of type `print-not-readable` is signaled.

1. Note Since `#:KEYWORD` cannot be used as a *local nickname*, and no *local nicknames* can be defined in the `#:KEYWORD` package, this expression is guaranteed to evaluate to the symbol in the correct package.

2.2.5 Proposal SHARPSIGN-COLON

In the 5th case the symbol is printed using the *extended `#:` syntax*:

```
#:(package name)
#::(package name)
```

Shinmera's idea.

2.2.6 Proposal SHARPSIGN-BACKQUOTE

In the 5th case the symbol is printed using the new `#‘` syntax for reading an expression ignoring *local nicknames* in the *current package*:

```
#‘foo:bar
#‘foo::bar
```

It can be implemented roughly as follows:

```
(defun |#‘-reader| (stream subchar arg)
  (declare (ignore subchar arg))
  (let* ((current-package *package*)
         (local-nicknames (package-local-nicknames current-package)))
    (loop for (nick . package) in local-nicknames
          do (remove-package-local-nickname nick current-package))
    (unwind-protect
      (read stream t nil t)
      (loop for (nick . package) in local-nicknames
            do (add-package-local-nickname nick package current-package)))))

(set-dispatch-macro-character #\# #\‘ #’|#‘-reader|)
```

It is *implementation-dependent* whether *local nicknames* are actually removed from the *current package* or not.

2.2.7 Proposal PRINT-UNREADABLY

In the 5th case the symbol is printed unreadably using the `#<` syntax:

```
#<SYMBOL IN THE SHADOWED PACKAGE FOO:BAR>
#<SYMBOL IN THE SHADOWED PACKAGE FOO::BAR>
```

(Specifics are *implementation-dependent*.)
If `*print-readably*` is *true*, an error of type `print-not-readable` is signaled.

2.2.8 Proposal THREE-FOUR-PACKAGE-MARKERS

In the 5th case the symbol is printed using the extended symbol token syntax:

```
foo:::bar   ; same as (cl:find-symbol "BAR" "FOO") in the #:KEYWORD package
foo::::bar  ; same as (cl:intern "BAR" "FOO") in #:KEYWORD package
```

2.2.9 Links

See CLHS 22.1.3.3.1 Package Prefixes for Symbols.

2.3 Issue 3 (Local nicknames effect on DEFPACKAGE, MAKE-PACKAGE and others)

2.3.1 Description

It is not clear whether *local nicknames* of the *current package* should affect the package lookup in operators that accept a package designator as an argument. This mainly concerns `make-package` and `defpackage`. See also Notes.

2.3.2 Examples

```
(defpackage #:foo-a (:use) (:export #:x))
(defpackage #:foo-b (:use) (:export #:x))
```

```
(defpackage #:bar
  (:use #:cl)
  (:local-nicknames (#:foo-a #:foo-b)
                    (#:foo-b #:foo-a)))

(in-package #:bar)

(defpackage #:quux-1
  (:use #:foo-a))
(package-name (symbol-package 'quux-1::x))
; => "FOO-B"  (sbcl, ccl, acl, abcl, lispworks)
; => "FOO-A"  (ecl, clasp)

(make-package '#:quux-2 :use '(#:foo-a))
(package-name (symbol-package 'quux-2::x))
; => "FOO-B"  (sbcl, ccl, ecl, acl, abcl, clasp, lispworks)

(defpackage #:quux-3
  (:use)
  (:local-nicknames (#:foo #:foo-a)))
(let ((*package* (find-package '#:quux-3)))
  (package-name (find-package '#:foo)))
; => "FOO-B"  (ccl, ecl, acl, abcl)
; => "FOO-A"  (sbcl, clasp, lispworks)

(import (car (find-all-symbols (string '#:add-package-local-nickname))))
(defpackage #:quux-4
  (:use))
(add-package-local-nickname '#:foo '#:foo-a '#:quux-4)
(let ((*package* (find-package '#:quux-4)))
  (package-name (find-package '#:foo)))
; => "FOO-B"  (ccl, ecl, clasp, abcl, lispworks)
; => "FOO-A"  (sbcl)

(use-package '#:foo-a '#:quux-4)
(package-name (symbol-package 'quux-4::x))
; => "FOO-B"  (sbcl, ccl, ecl, clasp, abcl, lispworks)
```

2.3.3 Current behavior

- `defpackage` and `make-package` sbcl, lispworks: Only the `:local-nicknames` clause is **not** affected by *local nicknames*.

  ccl, acl, abcl: All clauses and keyword arguments are affected.

  ecl: Only the `:local-nicknames` clause and all keyword arguments (`:use` and `:local-nicknames`) are affected.

  clasp: Only the keyword argument `:use` is affected.

- Known exceptions in other operators: sbcl: `add-package-local-nickname` is not affected.

lispworks: `in-package` is not affected.

2.3.4 Proposal ALL-AFFECTED

All operators taking a package designator as an argument must be affected by *local nicknames* of the *current package*.

In particular, all `defpackage` clauses (`:use`, `:local-nicknames`, `:import-from`, `:shadowing-import-from`) as well as all keyword arguments to `make-package` (`:use` and `:local-nicknames`) must be affected.

2.3.5 Notes

A non-exhaustive list of possibly affected by this issue operators: `defpackage`, `make-package`, `export`, `find-symbol`, `import`, `rename-package`, `shadow`, `shadowing-import`, `delete-package`, `with-package-iterator`, `unexport`, `unintern`, `in-package`, `unuse-package`, `use-package`, `do-symbols`, `do-external-symbols`, `do-all-symbols`, `intern`, `package-name`, `package-nicknames`, `package-shadowing-symbo` `package-use-list`, `package-used-by-list`, `add-package-local-nickname`, `remove-package-local-nickname`, `package-local-nicknames`, `package-locally-nicknamed-by`.

`format` is **not** affected by this issue, see instead Issue 8.

2.4 Issue 4 (Local nicknames of the package being defined)

2.4.1 Description

It is not clear whether *local nicknames* of the package **being defined** should affect `make-package` or `defpackage`.

2.4.2 Examples

```
(defpackage #:foo-a (:use) (:export #:x))
(defpackage #:foo-b (:use) (:export #:x))

(defpackage #:bar
  (:local-nicknames (#:foo-a #:foo-b)
                    (#:foo-b #:foo-a))
  (:use #:foo-a))

(package-name (symbol-package 'bar::x))
; => "FOO-A"  (sbcl, ccl, acl, abcl, clasp, lispworks)
; => "FOO-B"  (ecl)
```

2.4.3 Current behavior

sbcl, ccl, acl, abcl, lispworks: Nothing is affected.

ecl: `:use`, `:import-from` and `:shadowing-import-from` clauses are affected.

clasp: `:local-nicknames` clause is affected by *local nicknames* introduced by previous `:local-nicknames` clauses.

2.4.4 Proposal NO-EFFECT

Local nicknames of the package being defined must not affect any of the defpackage clauses (`:use`, `:local-nicknames`, `:import-from`, `:shadowing-import-from`).

The keyword argument `:local-nicknames` to `make-package` must not affect the `:use` keyword argument either.

2.5 Issue 5 (Local nickname shadowing package's own name)

2.5.1 Description

It is not clear whether it should be allowed to define a *local nickname* that shadows thex *name* or one of the *global nicknames* of the package that it is defined in.

2.5.2 Examples

```
(defpackage #:foo
  (:use)
  (:nicknames #:bar)
  (:local-nicknames (#:foo #:cl)
                    (#:bar #:cl)))
; => continuable error  (sbcl, ccl, abcl)
; => error  (lispworks)
; => ok  (ecl, acl, clasp)
```

2.5.3 Current behavior

sbcl, ccl, abcl: A correctable error is signaled by `defpackage`.

lispworks: An error is signaled by `defpackage`. A correctable error is signaled by `add-package-local-nickname`.

ecl, acl, clasp: No errors are signaled.

2.5.4 Proposal ALLOW-WITH-STYLE-WARNING

It should be allowed to use the name or one of the global nicknames of the package as a local nickname, but a style warning might be issued.

1. Rationale Such local nicknames are not likely to break anything. Even though they can be a bit confusing, this alone does not warrant an error to be signaled.

   Moreover, on all implementations it is possible to obtain such nicknames, even if on some of them invoking the `continue` restart is required. This suggests that cost of adoption is very low.

2.5.5 Proposal DISALLOW-WITH-CORRECTABLE-ERROR

Attemts to create such a nickname should result in a correctable error being signaled. The `continue` restart can be used to create the nickname anyway.

2.6 Issue 6 (Additional keyword argument to MAKE-PACKAGE)

2.6.1 Descriptions

An additional keyword argument to `make-package` similar to the `:use` keyword argument would be a nice feature. There are only two implementations introducing this extension, unfortunately they use different API for this argument.

Possible questions:

- Does it accept an alist or a nested list?
- Is the default value implementation-dependent, or is it an empty list?

2.6.2 Current behavior

sbcl, ccl, abcl, clasp, lispworks: no additional keyword argument.

ecl: has an additional keyword argument `:local-nicknames`, but it is undocumented and it segfaults on incorrect usage. The expected value is a list of conses: `((nickname . package)*)`.

acl: has an additional keyword argument `:local-nicknames`. The expected value is a list of lists: `((nickname package)*)`.

2.6.3 Proposal EXTRA-KEYWORD-ARGUMENT-LIST

Add `:local-nicknames` keyword argument to `make-package`:

```
local-nicknames ::= ((nickname package)*)
```

`nickname` — a *string designator*.
`package` — a *package designator*.
`local-nicknames` defaults to an *empty list*.

2.6.4 Links

See make-package.

2.7 Issue 7 (Multiple local nicknames)

2.7.1 Description

It is not clear whether `package-locally-nicknamed-by-list` should be allowed to return a list with duplicate entries (when there are multiple local nicknames in one package).

2.7.2 Examples

```
(defpackage #:foo
  (:use)
  (:local-nicknames (#:bar #:cl)
                    (#:baz #:cl)))
(mapcar #'package-name (package-locally-nicknamed-by-list '#:cl))
; => ("FOO")  (sbcl, acl, clasp, lispworks)
; => ("FOO" "FOO")  (ccl, ecl, abcl)
```

2.7.3 Current behavior

sbcl, acl, clasp, lispworks: `package-locally-nicknamed-by-list` never contains duplicate entries.
ccl, abcl, ecl: `package-locally-nicknamed-by-list` might contain duplicate entries.

2.7.4 Proposal NO-DUPLICATES

`package-locally-nicknamed-by-list` should return a list without duplicate entries.

2.8 Issue 8 (Interaction with FORMAT)

by /3b/

2.8.1 Description

It is not clear how *local nicknames* should affect the tilde slash `\~//` directive of `format`.

2.8.2 Notes

- If `\~//` directive wasn't affected by *local nicknames*, it would mean that using [long and not-so-easy-to-use] package names is necessary.
- It would be counterintuitive if the function used by the `\~//` directive could change based on the *current package* at **execution time** (that is, if the symbol lookup was affected by *local nicknames* of the *current package*).

  This could also break existing code if a *local nickname* in the *current package* shadows the name of a package containing a function used in a format control string in the called function. The only way to prevent this issue would be to rebind the `*package*` variable around any call to `format` where the `\~//` directive is used.
- Finally, if the call to `format` is not compiled, it would be hard to impossible to find the function using *local nicknames* of the package that was the *current package* at **compile time**.

2.8.3 Examples

```
(defpackage #:foo-a (:use) (:export #:ff))
(defpackage #:foo-b (:use) (:export #:ff))

(defun foo-a:ff (stream &rest args)
  (declare (ignore args))
  (format stream "FOO-A:FF"))

(defun foo-b:ff (stream &rest args)
  (declare (ignore args))
  (format stream "FOO-B:FF"))

(defpackage #:bar-a
  (:use #:cl)
  (:local-nicknames (#:nick #:foo-a)))
```

```
(defpackage #:bar-b
  (:use #:cl)
  (:local-nicknames (#:nick #:foo-b)))

(in-package #:bar-a)

(defun test ()
  (format t "Called ~/nick:ff/ & " nil)
  (let ((*package* (find-package (quote #:bar-a))))  ; or #.*package*
    (format t "~/nick:ff/~%" nil)))

(test)
; => "Called FOO-A:FF & FOO-A:FF"  (sbcl, ccl, ecl, acl, abcl, clasp)
; lispworks errors (NICK package not found)
(let ((*package* (find-package (quote #:bar-b))))
  (test))
; => "Called FOO-A:FF & FOO-A:FF"  (sbcl, clasp)
; => "Called FOO-B:FF & FOO-A:FF"  (ccl, ecl, acl, abcl)
; lispworks errors (NICK package not found)
```

2.8.4 Proposal NOT-AFFECTED-BY-LOCAL-NICKNAMES

The tilde slash `\~//` directive of `format` must **not** use *local nicknames* of any package when looking up the specified symbol.

1. Rationale When the package prefix is not specified, the specified symbol is not looked up in the *current package*, but instead in the `#:CL-USER` package. That suggests that the tilde slash `\~//` directive should not depend on the value of `*package*` at any time.

   Note that specifying it to use *local nicknames* of the `#:CL-USER` package would risk breaking the existing code, since it is allowed to add local nicknames to the `#:CL-USER` package.

2.8.5 Links

See CLHS 22.3.5.4 Tilde Slash: Call Function.

2.9 Issue 9 (Empty string as a local name)

2.9.1 Description

It is not clear whether it should be allowed to use the empty string `""` as a local nickname, and in the case it is allowed, whether it should affect the keyword symbol syntax `:xxxx`.

2.9.2 Examples

```
(defpackage #:foo
  (:use #:cl)
  (:local-nicknames ("" #:cl)))

(in-package #:foo)
```

```
(package-name (symbol-package ':*package*))
; => "KEYWORD"  (sbcl, ccl, ecl, abcl, clasp, lispworks)
; => "COMMON-LISP"  (acl)

(package-name (symbol-package '||:*package*))
; => "KEYWORD"  (ecl, clasp, lispworks)
; => "COMMON-LISP"  (sbcl, ccl, acl)
; abcl errors
```

2.9.3 Current behavior

sbcl, ccl: `:xxxx` is read as a keyword; `||:xxxx` is read as a symbol in the package named or nicknamed `""`.

ecl, clasp, lispworks: Both `||:xxxx` and `:xxxx` are read as a keyword.

acl:

Both `:xxxx` and `||:xxxx` are read as a symbol in the package named or nicknamed `""` which is a *global nickname* for the `#:KEYWORD` package, but can be shadowed by a *local nickname*.

abcl: `:xxxx` is read as a keyword; `||:xxxx` syntax cannot be read (attemts result in an error).

2.9.4 Proposal ALLOW-KEEP-KEYWORDS

The `""` local nickname should be explicitely allowed. `:xxxx` should be always read as a keyword regardless of package names or nicknames. `||:xxxx` should be read as a symbol in the package named or nicknamed by `""`.

2.9.5 Proposal ALLOW-FUN

The `""` local nickname should be explicitely allowed. Both `:xxxx` and `||:xxxx` should be read as a symbol in the package named or nicknamed by `""`.

2.9.6 Links

See also the WSCL issue 63.

3 Links

3b's notes on package-local nicknames.

phoe's tests.

SBCL's manual entry.

Section 4.3 of the ABCL's manual. (TEX file on github)

Allegro CL's documentation.

4 Copying and License

[TODO]