Package-Local Nicknames

Gleefre

July 5, 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 allow to use short and easy-to-use names for packages locally without potentially introducing name conflict as with normal nicknames.

1.1.2 Current state

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

Unfortunately, there are multiple inconsistencies between implementations, all implementations lose the **print-read** consistency to some extent, and there are multiple edge cases that aren't always implemented correctly.

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 this extension.

1.2 Description

Package-local nickname (or *local nickname*) has the same effects as a usual *package nickname* (later *global nickname*), except that these effects only apply when `*package*` is bound to a package for which the nickname has been defined.

That means that calls to `find-package` with a *local nickname* defined in the *current package* should return 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.

Local nickname is allowed to shadow a *package name* or a *global nickname*, except for the names `#:CL`, `#:COMMON-LISP` and `#:KEYWORD` which should always refer to their packages.

The consequences of adding *local nicknames* to the packages `#:COMMON-LISP` and `#:KEYWORD` are undefined.

1.3 API

1.3.1 defpackage

`defpackage` options are extended to include *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.

1. Arguments and Values: `nickname` must be a *string designator*.

   `package` must be a *package designator*.

2. Exceptional situations An error of type `package-error` is signaled when a package designated by `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.

3. Implementation dependent The behaviour is unspecified when a *local nickname* is specified for the package that is being defined. (See Issue 4.)

   The behaviour is unspecified when supplied *local nicknames* are at variance with the current state of the package that is being defined. An implementation might choose to remove all present *local nicknames* at the begining of each redefinition of the package.

1.3.2 make-package

(**PROPOSAL**: see Issue 6.)

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

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

`local-nicknames` defaults to an *empty list*.

`local-nicknames` must be a *list* each element of which must be a *list* of form `(nickname package)`. Specifies *local nicknames* in the new *package*.

1. Arguments and Values: `local-nicknames` must be a a *list* of pairs `(nickname package)`.

   `nickname` must be a *string designator*.

   `package` must be a *package designator*.

2. Exceptional situations An error of type `package-error` is signaled when a package designated by `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.

3. Implementation dependent The behaviour is unspecified 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
```

`designated-package` defaults to the *current package*.

Adds a *package-local nickname* `nickname` for the `actual-package` in the `designated-package`.

Returns the package designated by `designated-package`. (But also see Issue 1.)

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

1. Arguments and Values `nickname` must be a *string designator*.

   `actual-package` and `designated-package` must be *package designators*.

   `designated-package-object` is of type *package*.

2. Exceptional situations If a package designated by `actual-package` or a package designated by `designated-package` does not exists, an error of type *package-error* must be signaled.

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

   If `nickname` is a *local nickname* for a package different from `actual-package`, an error of type *package-error* must be signaled.

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

   (**PROPOSAL**: see Issue 5.)

   If `nickname` shadows the `designated-package`'s *package name* or one of its *global nicknames*, a style warning might signaled.

1.3.4 remove-package-local-nickname

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

`designated-package` defaults to the *current package*.
If `designated-package` has `old-nickname` as a *local nickname*, it is removed.
Returns *true* if the `old-nickname` existed and was removed, and `NIL` otherwise. (But also see Issue 1.)

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

   `designated-package` must be a *package designator*.

   `nickname-removed-p` is a *generalized boolean*.

2. Exceptional situations If a package designated by `designated-package` does not exists, an error of type *package-error* must be signaled.

1.3.5 package-local-nicknames

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

Returns an *alist* describing *local nicknames* defined in the package designated by `package`.

1. Arguments and Values `package` must be a *package designator*.

   `local-nicknames-alist` is an *alist* with keys of type *string* and values of type *package*.

   `nickname` must be a *string*.

   `package` must be a *package*.

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

3. 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)
  => packages-list
```

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

1. Arguments and Values `package` must be a *package designator*.

   `packages-list` is a *list* with elements of type *package*.

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

3. 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

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

(**PROPOSAL**: see Issue 8.)

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

1.4.4 rename-package

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

1. Implementation dependent (**PROPOSAL**: 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 redefined, a warning might be signaled.

1.4.5 delete-package

When a package is deleted with `delete-package` all *local nicknames* defined in other packages that it was nicknamed by must be removed, as well as all *local nicknames* defined in the package that is being deleted.

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-15` (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

Return values of `add-package-local-nickname` and `remove-package-local-nickname` are inconsistent. The first one always returns *designated package*, while the second one returns *true* (generalized boolean) if a nickname was removed.

Moreover, there is no consensus among current implementations as to what the second function (`remove-package-local-nickname`) should return.

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` if a nickname was removed, returns *true* (more specifically - the package that was nicknamed by that nickname), and `NIL` otherwise.

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

2.1.4 Proposal DESIGNATED-PACKAGE-IF-SUCCESSFUL

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

2.1.5 Proposal ALWAYS-T

Similar to `use-package` and `unuse-package`:

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

2.1.6 Proposal ELIMINATE-GENERALIZED-BOOLEAN

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

2.2 Issue 2 (PRINT-READ consistency)

2.2.1 Description

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 symbol's *home package*.

There are several situations to consider:

1. Symbol is *apparently uninterned*.

   In this case it must be printed without any package prefix, preceeded by `#:`.

2. Symbol is accessible in the *current package*.

   In this case it must be printed without any package prefix.

3. Symbol's *home package name* or one of its *global nicknames* is not shadowed by any *local nickname* defined in the *current package*.

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

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

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

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

   It is not clear how the symbol must be 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 exists in the *current package*, a *local nickname* is used as a package prefix when printing a symbol.

ecl, acl, clasp: *local nickname* is never used as a package prefix when printing a symbol.

2.2.4 Proposal SHARPSIGN-DOT

In the case (5) the symbol must be 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"))
```

Note that `#:KEYWORD` name is reserved for the `#:KEYWORD` package and cannot be used as a *local nickname* thus this expression will always evaluate to the symbol `foo::bar`.

If `*read-eval*` is *false* and `*print-readably*` is *true* an error of type `print-not-readable` must be signalled.

2.2.5 Proposal SHARPSIGN-COLON

In case (5) the symbol should be printed using the *extended `#:` syntax*:

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

Shinmera's idea.

2.2.6 Proposal SHARPSIGN-BACKQUOTE

In case (5) the symbol must be 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 case (5) the symbol must be 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` must be signalled.

2.2.8 Proposal THREE-FOUR-PACKAGE-MARKERS

In the case (5) the symbol must be printed using `:::` and `::::` syntax as follows:

```
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 resolution of package designators provided in `make-package` and `defpackage`, as well as other functions and macros taking a package designator as an argument (`in-package`, `add-package-local-nickname` and others).

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

sbcl, lispworks: only `:local-nicknames` clause is **not** affected by *local nicknames*.

ccl, acl, abcl: all options are affected.

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

clasp: only keyword argument `:use` is affected.

Known exceptions in other functions/macros: sbcl: `add-package-local-nickname` is not affected. lispworks: `in-package` is not affected.

2.3.4 Proposal ALL-AFFECTED

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 by the *local nicknames* of the *current package*.

All functions and macros taking a package designator as an argument must be affected as well.

A non-exhaustive list of affected functions and macros: `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`, `package-locally

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: not affected

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

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

2.4.4 Proposal NO-EFFECT

Local nicknames of a package being defined should not affect other defpackage clauses (`:use`, `:local-nicknames`, `:import-from`, `:shadowing-import-from`).

The keyword argument `:local-nicknames` to `make-package` should 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 is valid to have a *local nickname* in a package shadowing its own name or nickname.

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 and lispwork signal an error.

2.5.4 Proposal ALLOW

It should be allowed to use package's own name or global nickname, but a style-warning can be signalled.

1. Rationale Such local nicknames are not likely to break anything. Even though they can be a bit confusing, this fact alone doesn't warrant an error being signalled.

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

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

2.6.1 Current behavior

sbcl, ccl, abcl, clasp, lispworks: no additional key 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)*)`.

2.6.2 Proposal EXTRA-KEYWORD-ARGUMENT

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

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

`nickname` must be a *string designator*. `package` must be a *package designator*.
`local-nicknames` defaults to an *empty list*.
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 lists 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` must 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 `format`'s `\~//` directive.

First, it would be inconvinient if it wouldn't be possible to use *local nicknames* with the `\~//` directive.

Secondly, it would be unintuitive if the function used would depend on the *current package* at the **execution** time. This also might break [existing] code, if a local nickname in the *current package* shadows a package that contains a function used by a control string in another function.

Finally, if the call to `format` is not compiled, it is hard to impossible to find the function using *local nicknames* of the package that was the *current package* at the **compile** time.

2.8.2 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)
```

2.8.3 Proposal NO-LOCAL-NICKNAMES

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

Rationale: In the spirit of how when the package is not specified, the symbol is not looked up in the *current package*, but instead in the `#:CL-USER` package; the *tilde slash* directive should not depend on the value of `*package*` at any time. Specifying it to use *local nicknames* of the `#:CL-USER` package instead would risk breaking the existing code when adding local nicknames to that package.

2.8.4 Links

See CLHS 22.3.5.4 Tilde Slash: Call Function.

2.9 Issue 9 (Empty package local name)

2.9.1 Description

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

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: `||:xxxx` and `:xxxx` are read as a keyword.

acl: `:xxxx` and `||:xxxx` are read as a symbol in the package named or nicknamed `""`. `""` is by default a global nickname for the `#:KEYWORD` package.

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

2.9.4 Proposal ALLOW-BUT-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 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)

4 Copying and License

[TODO]