

# Package-Local Nicknames

Gleefre

June 12, 2023

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Rationale . . . . .	1
1.2	Current state . . . . .	2
1.3	Goal . . . . .	2
<b>2</b>	<b>Specification</b>	<b>2</b>
2.1	Description . . . . .	2
2.1.1	Concept . . . . .	2
2.1.2	API . . . . .	2
2.1.3	Affected symbols . . . . .	7
2.1.4	Edge cases . . . . .	8
2.1.5	<b>*FEATURES*</b> . . . . .	9
2.2	Examples . . . . .	9
<b>3</b>	<b>Links</b>	<b>10</b>
<b>4</b>	<b>Copying and License</b>	<b>10</b>
	[THIS IS A DRAFT]	

## 1 Introduction

This is a specification for package-local nicknames extension in Common Lisp.

### 1.1 Rationale

Package-local nicknames allow to use short and easy-to-use names without potentially introducing name conflict as with normal nicknames.

## 1.2 Current state

Package-local nicknames are implemented (at least partially) in SBCL, CCL, ECL, Clasp, ABCL, Allegro CL, LispWorks. Unfortunately, there are multiple inconsistencies between implementations, and all implementations lose **print-read** consistency to some extent.

## 1.3 Goal

The purpose of this document is to standardize the package-local nicknames extension and to address some existing issues (mostly **print-read** consistency).

[TODO]

This CDR also aims to provide an extensive test suite for this extension.

# 2 Specification

## 2.1 Description

### 2.1.1 Concept

*Package-local nickname* (or *local nickname*) has the same effects as a normal *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 PRINT-READ consistency.

*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.

### 2.1.2 API

- 
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 and is executed **after** all standard options.

To avoid confusion, *local nicknames* of the *current package* must be ignored in `defpackage`. In other words `defpackage` must **not** be affected by the `*package*` special variable.

- (a) Arguments and Values: `nickname` must be a *string designator*.

`package` must be a *package designator*.

- (b) Exceptional situations An error of type `package-error` is signaled when a package designated by `package` does not exist.

[OR DOES IT?]

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

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

- (c) Implementation dependent The behaviour is unspecified when a package designated by `package` does not exist.

[OR IT IS?]

The behaviour is unspecified when a *local nickname* is specified for the package that is being defined.

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 beginning of each redefinition of the package.

---

## 2. make-package **PROPOSAL**

To avoid confusion, *local nicknames* of the *current package* must be ignored during evaluation of `make-package`. In other words `make-package` must **not** be affected by the `*package*` special variable.

`make-package` lambda list is extended to include an additional key parameter: `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*.

- (a) Arguments and Values: `local-nicknames` must be a *string designator*.  
`nickname` must be a *string designator*.  
`package` must be a *package designator*.
  - (b) Exceptional situations An error of type `package-error` is signaled when a package designated by `package` does not exist.  
[OR DOES IT?]  
Name conflict errors are handled by the underlying calls to `add-package-local-nickname`.  
See `add-package-local-nickname`: exceptional situations.
  - (c) Implementation dependent The behaviour is unspecified when a package designated by `package` does not exist.  
[OR IT IS?]  
The behaviour is unspecified when a *local nickname* is specified for the package that is being defined.
  - (d) Notes It is still possible to specify a package designated by *local nickname* in `:use` and/or `:local-nicknames` parameters by calling `find-package` before calling `make-package`.
- 

### 3. `add-package-local-nickname`

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

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

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

Returns the package designated by `target-package`.

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

- (a) Arguments and Values `nickname` must be a *string designator*.  
`package` and `target-package` must be *package designators*.  
`target-package-object` is of type *package*.

- (b) Exceptional situations If a package designated by `package` or a package designated by `target-package` does not exist, 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 `package`, an error of type *package-error* must be signaled.

(c) Implementation dependent **PROPOSAL**

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

[Now: SBCL signals an error, but that doesn't make much sense, especially if a *local nickname* shadows the *global nickname* of a package and not its name. And it is possible to achieve a *local nickname* shadowing a *global nickname* or the *package name* by renaming the package.]

---

#### 4. `remove-package-local-nickname`

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

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

If `target-package` has `old-nickname` as a *local nickname*, it is removed.

Returns *true* if the `old-nickname` existed (and was removed), and *NIL* otherwise.

#### **ISSUE**

Inconsistency between return values of `add-package-local-nickname` and `remove-package-local-nickname`. Somewhat similar functions from the standard - `use-package` and `unuse-package` always return *T*.

#### **PROPOSAL**

- `add-package-local-nickname` and `remove-package-local-nickname` should always return *T*.
- `add-package-local-nickname` and `remove-package-local-nickname` should return two values: `target-package-object` and `operation-successful-p`.

- (a) Arguments and Values **old-nickname** must be a *string designator*.  
**target-package** must be a *package designator*.  
**nickname-removed-p** is a *generalized boolean*.
  - (b) Exceptional situations If a package designated by **target-package** does not exist, an error of type *package-error* must be signaled.
- 

## 5. package-local-nicknames

```
(package-local-nicknames package)  
=> local-nicknames-alist
```

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

Each cons cell in **local-nicknames-alist** is of the form (**nickname** . **package**) where **nickname** is of type *string* and **package** is of type *package*.

- (a) 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*.
  - (b) Exceptional situations An error of type **package-error** is signaled when a package designated by **package** does not exist.
  - (c) Notes The returned *alist* must be safe to be modified by the user.
- 

## 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**.

- (a) Arguments and Values **package** must be a *package designator*.  
**packages-list** is a *list* with elements of type *package*.
  - (b) Exceptional situations An error of type **package-error** is signaled when a package designated by **package** does not exist.
  - (c) Notes The returned *list* must be safe to be modified by the user.
-

### 2.1.3 Affected symbols

---

1. `defpackage` See `defpackage`.
- 

2. `make-package` See `make-package`.
- 

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

There are two exceptions: `make-package` and `defpackage` must **not** be affected by *local nicknames* of the *current package*.

---

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

- (a) Implementation dependent **PROPOSAL**

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

---

5. `delete-package` When a package is deleted via `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 this package must not be available by calls to `package-locally-nicknamed-by-list` and `package-local-nicknames`.

---

#### 2.1.4 Edge cases

1. PRINT-READ consistency Lisp reader uses `find-package` to read a symbol, and is affected by *local nicknames* of the *current package*. So in order to maintain **print-read** consistency it is required to use a correct *package prefix* - such prefix that calling `find-package` on it in the *current package* will return the symbol's *home package*.

There are several situations to consider:

- (a) There **is** a *local nickname* defined in the *current package* for the symbol's *home package*.  
*In this case such local nickname can be used as the package prefix.*
- (b) 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 package name or global nickname can be used as the package prefix.*
- (c) Symbol's *home package name* and all its *global nicknames* are shadowed by one of the *local nicknames* of the *current package* and there **is no** *local nickname* defined (in the *current package*) for the symbol's *home package*.

#### PROPOSALS

- 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`.

- In this case the symbol must be printed using `:::` and `::::` syntax to lookup and intern ignoring *local nicknames* respectively:



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

- In this case the symbol must be printed using the `#'` syntax for reading an expression ignoring *local nicknames* in the *current package*:

```
#'foo:bar and #'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.

- In this case the symbol must be printed unreadably (specifics are implementation dependent):

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

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

If `*print-readably*` is *true* must signal an error of type `print-not-readable` without printing anything.

### 2.1.5 \*FEATURES\*

If an implementation supports package-local nicknames it should add symbols `:package-local-nicknames` and `:cdr-15` (per CDR 14) to `*features*`.

## 2.2 Examples

[TODO]

### **3 Links**

3b's notes on package-local nicknames.  
phoe's tests.  
SBCL's manual entry.

### **4 Copying and License**

[TODO]