Package-Local Nicknames

Gleefre

June 23, 2023

Contents

[THIS IS A DRAFT]

1 ISSUES

1.1 Issue 1.

Inconsistency between return values of `add-package-local-nickname` and `remove-package-local-nickname`. First always returns *designated package*, second one returns *true* if a nickname was removed (but it is not specified what exactly is returned). (Somewhat similar functions from the standard - `use-package` and `unuse-package` always return `T` - their api is consistent.)

1.1.1 Proposal

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

1.1.2 Currently

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

sbcl, ccl, ecl, acl, clasp: `remove-package-local-nickname` returns `T` on success and `NIL`

abcl: `remove-package-local-nickname` returns package nicknamed by the removed nickname on success and `NIL`. [That seems nice]

1.2 Issue 2 (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:

1. 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.

2. 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.

3. 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.

   It is not clear what should be done in this case (see proposals).

1.2.1 Example

```
(defpackage #:a (:use) (:export #:+))
(defpackage #:b (:local-nicknames (#:a #:cl)))
(let ((*package* (find-package '#:b)))
  (print 'a:+))
; => a:+ everywhere
;; but in #:b package a:+ would refer to cl:+

(defpackage #:a (:use) (:export #:+))
(defpackage #:b (:use) (:export #:+))
(defpackage #:c (:use) (:local-nicknames (#:a #:b) (#:b #:a)))
```

```
(let ((*package* (find-package '#:c)))
  (print 'a:+))
; => b:+  (sbcl, ccl, abcl)
; => a:+  (clasp, acl, ecl)
;; but in #:c package a:+ would refer to b:+
```

1.2.2 Currently

sbcl, ccl, abcl: print symbol with package name when all package's names and nicknames are shadowed by *current package*'s *local nicknames*.

ecl, acl, clasp: don't print with local nicknames at all.

1.2.3 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`.

- *Shinmera's idea.* In this case an extended `#:` syntax should be used:

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

- 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.

- 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
  ```

1.3 Issue 3

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

1.3.1 Example

```
(defpackage #:a (:use) (:export #:x))
(defpackage #:b (:use) (:export #:x))
(defpackage #:c
  (:use #:cl)
  (:local-nicknames (#:a #:b) (#:b #:a)))

(in-package #:c)
```

```
(defpackage #:d
  (:use #:a)
  (:export #:x))
(print (package-name (symbol-package 'd:x)))
; => "B"  (sbcl, ccl, acl, abcl)
; => "A"  (ecl, clasp)

(defpackage #:e
  (:use)
  (:local-nicknames (#:x #:a)))

(let ((*package* (find-package '#:e)))
  (print (package-name (find-package '#:x))))
; => "B"  (ccl, acl, abcl)
; => "A" (sbcl, ecl, clasp)
```

1.3.2 Proposal

They should affect all options like `:use`, `:local-nicknames`, `:shadowing-import-from` and `:import-from`.

1.3.3 Currently

ccl, acl, abcl: they do.
sbcl: partly (:use is affected, :local-nicknames are not)
ecl, clasp: they don't.

1.4 Issue 4

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

1.4.1 Example

```
(defpackage #:a (:use) (:export #:x))
(defpackage #:b (:use) (:export #:x))
(defpackage #:c
  (:use #:a)
  (:local-nicknames (#:a #:b) (#:b #:a)))

(print (package-name (symbol-package 'c::x)))
```

```
; => "A"  (sbcl, ccl, acl, abcl, clasp)
;; ecl errors... :/
```

1.4.2 Proposal

They should not.

1.4.3 Currently

sbcl, ccl, acl, abcl: they don't.

ecl: they do (or not? not sure).

clasp: they do in some weird way - also depends on order in which PLN are added.

1.5 Issue 5

It is not clear whether it is valid to have a *local nickname* in a package shadowing its own name or nickname.

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.

1.5.1 Example

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

1.5.2 Proposal

It should be allowed, but a warning might be signaled.

1.5.3 Currently

sbcl, ccl, abcl: not allowed.

ecl, acl, clasp: allowed.

1.6 Proposal 6

Add `:local-nicknames` option to `make-package` similar to `defpackage`. See make-package.

1.6.1 Currently

ecl: takes keyword parameter, but segfaults on incorrect usage + doesn't have it in docstring. Also API differs :/

acl: has it

abcl, sbcl, ccl, clasp: don't have it.

1.7 Issue 7

It is not clear whether functions `package-local-nicknames` and `package-locally-nicknamed-by-list` should be allowed to return lists with duplicate entries.

1.7.1 Example

```
(defpackage #:a (:use) (:local-nicknames (#:b #:cl) (#:c #:cl)))
(print (mapcar #'package-name (package-locally-nicknamed-by-list '#:cl)))
; => ("A")  (sbcl, acl, clasp)
; => ("A" "A")  (ccl, ecl, abcl)
```

1.7.2 Proposal

They should not.

1.7.3 Currently

sbcl, acl, clasp, ccl, abcl, ecl: don't return duplicate entries from `package-local-nicknames`.

sbcl, acl, clasp: don't return duplicate entries from `package-locally-nicknamed-by-list`.

ccl, abcl, ecl: return duplicate entries from `package-locally-nicknamed-by-list`.

1.8 Issue 8 *by /3b/*

How PLN affect `format`'s `\~//` directive? It seems tricky - compilers might want optimize it at compile time, but reffered symbol might change on re-binding the `*package*`.

1.8.1 Example

```
;;;; File test.lisp
(defpackage #:a (:use) (:export #:ff))
(defpackage #:b (:use) (:export #:ff))

(defun a:ff (stream &rest args)
```

```
  (declare (ignore args))
  (format stream "A:FF"))

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

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

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

(in-package #:foo)
(defun test ()
  (format t "Called ~/nick:ff/ & " nil)
  (let ((*package* (find-package '#:foo)))
    (format t "~/nick:ff/~%" nil)))

(test)
; => "Called A:FF & A:FF"  (sbcl, ccl, acl, abcl, ecl, clasp)

(let ((*package* (find-package '#:bar)))
  (test))
; => "Called A:FF & A:FF"  (sbcl, clasp)
; => "Called B:FF & A:FF"  (ccl, acl, abcl, ecl)
```

2 Introduction

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

2.1 Rationale

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

2.2 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.

2.3 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.

3 Specification

3.1 Description

3.1.1 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.

3.1.2 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.

(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 exists.
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 *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 begining of each redefinition of the package.
[TODO: What happens when a package is redefined with local nicknames in other packages that it is nicknamed by? It probably can't be strictly defined since redefining package is implementation dependent... But seems like they must be left intact.]

---

2. make-package Proposal#6

`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 a *list* of pairs `(nickname package)`.
`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 exists.

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 *local nickname* is specified for the package that is being defined.

---

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`.

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

(a) 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*.

(b) 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.

(c) Implementation dependent **PROPOSAL** (See issues#4.)

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

---

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.

(a) Arguments and Values `old-nickname` must be a *string designator*.
`designated-package` must be a *package designator*.
`nickname-removed-p` is a *generalized boolean*.

(b) Exceptional situations If a package designated by `designated-package` does not exists, 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 exists.

(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 exists.

(c) Notes The returned *list* must be safe to be modified by the user.

3.1.3 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** (See issues#4.)
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`.

3.1.4 `*FEATURES*`

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

3.2 Examples

[TODO]

4 Links

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

5 Copying and License

[TODO]