PLN CDR draft: spec

Gleefre

July 7, 2024

1 Introduction

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

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.

1.2 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.

1.3 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.

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.

3 API

3.1 defpackage

3.1.1 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.

3.1.2 Arguments and Values:

`nickname` — a *string designator*.
`package` — a *package designator*.

3.1.3 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 `make-package` and `add-package-local-nickname`.

3.1.4 Implementation dependent

The consequences are undefined when a *local nickname* is specified for the 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.

3.2 make-package

3.2.1 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*.

3.2.2 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*.

3.2.3 Exceptional situations

An error of type *package-error* is signaled when a package designated by the `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 two or more local nicknames result in a name conflict, a *correctable* error of type *package-error* is signaled. A name conflict occurs when multiple local nicknames have same nicknames (equal by `string=`) but different packages.

A name conflict between multiple local nicknames may be resolved in favor of either nickname being defined.

3.2.4 Implementation dependent

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

3.3 add-package-local-nickname

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

3.3.1 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*.

3.3.2 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`. If a name conflict occurs, restarts `abort` and `continue` can be used to correct the error.

If the `continue` restart is invoked, the existing *local nickname* is removed and the new nickname is defined.

If the `abort` restart is invoked, the existing nickname is not removed, and the new nickname is not defined.

3.3.3 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 another package different from the `actual-package`, a *correctable* error of type *package-error* is signaled.

3.3.4 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.

3.4 remove-package-local-nickname

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

3.4.1 Arguments and Values

`old-nickname` — a *string designator.*

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

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

3.4.2 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.

3.4.3 Exceptional situations

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

3.5 package-local-nicknames

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

3.5.1 Arguments and Values

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

`local-nicknames-alist` — an *alist.*

`nickname` — a *string.*

`package` — a *package.*

3.5.2 Description

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

3.5.3 Exceptional situations

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

3.5.4 Notes

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

3.6 package-locally-nicknamed-by-list

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

3.6.1 Arguments and Values

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

3.6.2 Description

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

3.6.3 Exceptional situations

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

3.6.4 Notes

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

4 Affected symbols

4.1 defpackage

See defpackage.

4.2 make-package

See make-package.

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.

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.

4.4.1 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.

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`.

4.6 format

See Issue 8.

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*`.

5 Examples

[TODO]