Common Lisp package code and symbol conversions
Ron Fischer

[This document is for developers only. It is not intended to be part of any external software release.]

The package code can behave mighty strangely if not operated "just so." Please read this document carefully.

Loading

The package code is now loaded in the full.sysout. It lives in {Eris}<LispCore>Library> on
CMLSYMBOL.DCOM
CMLPACKAGE.DCOM

Initializing

Packages, as you might expect, must be bootstrapped carefully. At its simplest, initialization is accomplished by calling just one function:

```
(package-init t)
```

simple startup

(package-init &optional (convert? nil)) [Function]
Clear, make structures of, initialize & convert symbols (if convert? is t) to, and enable use of the symbol package system. On a DandeTiger this takes about 25 minutes.

The bootstrap actually takes place in three steps. The first one creates all the package structure needed. The second passes over all the atoms in the sysout to "tag" them with packages. The third enables the package world.

step 1: make system packages

(package-make) [Function]
Create, but do not fill with symbols, the base packages that need to exist. Also enables the package qualifier characters in the readtables and saves the old definitions of \READ.SYMBOL and \MKATOM.

step 2: convert existing symbols

(package-hierarchy-init &optional (convert? nil)) [Function]
Fill all the initial system packages with their proper symbols, moving litatoms into appropriate places and such. If convert? is non-nil then symbols whose pnames have fake package qualifiers, like cl:length, will be converted IN PLACE to remove the qualifier. If conversion takes place you cannot fully disable the package system.

step 3: enable package system, remove old hacks

The third part of the bootstrap enables the use of packages by the reader, printer and friends. This involves redefining \MKATOM and \READ.SYMBOL. Also, if *package* is set to nil things will not go well. Please use package-enable and package-disable to start and stop, as these fellows know what they're doing. These faithful functions are availible on a menu by calling package-menu.

(package-enable &optional (package *interlisp-package*)) [Function]
Turn on the package system, making package the current one and redefining \READ.SYMBOL and \MKATOM appropriatly.

(package-disable) [Function]
Turn off the package system and restore the old definitions of \READ.SYMBOL and \MKATOM. After disabling, symbols interned under the package system will not be eq to symbols of the same name reread.

(package-menu) [Function]
Make a menu that allows turning the package system on or off without using the reader.

(package-hacks-disable) [Function]
Eliminates package simulation hacks when loading over an old sysout. These hacks cannot be re-enabled.

`(pkgconvert-enable)` [Function]
Enables a change in the behavior of the function `\\READ.SYMBOL` (using the function `check-symbol-namestring`), which converts symbols being read based on their "looking like" a pacakge prefixed symbol name. "Looking like" is defined by a table in the global `litatom-package-conversion-table`.

`litatom-package-conversion-table` [Variable]
a global variable containing a list of clauses used by the functions package-hierarchy-init and check-symbol-namestring to determine if a symbol "looks like" is trying to be package qualified. The list contains clauses with the following structure:

(*prefix-string list-of-exception-strings package-name-string where*)

prefix-string - string containing the prefix of symbols which this clause converts, eg `"CL:"`

list-of-exception-strings - list of strings naming symbols which this clause should leave alone, eg `("CL:FLG")`

package-name-string - string containing the name of the package that symbols converted by this clause should wind up in, eg `"LISP"`

where - either `:INTERNAL` or `:EXTERNAL`, indicates whether symbols converted by this clause should be external or internal in their package.

The initial value of this variable is:

```
(     ("CL:"       ("CL:FLG" "CL:MAKE-SYMBOL" "CL:COPY-SYMBOL" "CL:INTERN"
                    "CL:MAKE-KEYWORD" "CL:GENTEMP" "CL:KEYWORDP")
                         "LISP"            :external)
      (":"         nil   "KEYWORD"   :external)
)
```

Notes & cautions

The read functions may return strings if you use package qualifier syntax and `*package*` does not contain a package. This is part of debugging code that Bill put into them. Beware especially of making `*package*` `NIL` without calling `package-disable`.

The list of conversion clauses is searched linearly, hence longer prefixes should come before shorter ones with the same chars in them, ie put a clause for `"CL::"` before one for `"CL:"`.

Missing features

Cannot be placed early into loadup due to dependancies on CL files.

Apropos (and other Interlisp litatom functions) are not redefined to operate with packages.

Performance

There have been some small improvements since these timings were taken.

```
FILEDATES : (("21-Jul-86 14:12:17" .
{ERIS}<LISPCORE>CML>LAB>CMLPACKAGES.;129))
```

Testing symbol / litatom creation. Old array package. There are 6 random symbols made (new ones). intern is a factor of 16 slower, conses heavily, makes 3 strings for each call.

```
85#(TIMEALL (TIME.MKATOM))
Elapsed Time =       .336 seconds
SWAP time =           .32 seconds
CPU Time =           .016 seconds
PAGEFAULTS = 8
LISTP
9

86#(TIMEALL (TIME.INTERN))
Elapsed Time =       .449 seconds
SWAP time =          .188 seconds
```

```
CPU Time =            .261 seconds
PAGEFAULTS = 5
LISTP    STRINGP
423      18
```

Breakdown spy graph follows: