File created: 24-Oct-2021 10:20:31 **{DSK}<home>larry>medley>sources>LLPACKAGE.;4**

changes to: (IL:FUNCTIONS IL:FIND-EXTERNAL-SYMBOL IL:FIND-SYMBOL*)

previous date: 22-Sep-92 11:47:31 {DSK}<home>larry>medley>sources>LLPACKAGE.;1

Read Table: XCL

Package: LISP

Format: XCCS


```
(IL:RPAQQ IL:LLPACKAGECOMS
          (;; The Xerox Lisp package system, based on CMU's Spice Lisp
           ;; Internal macros and definitions
           (IL:DECLARE\: IL:EVAL@COMPILE IL:DONTCOPY (IL:FUNCTIONS IL:PACKAGE-LISTIFY IL:\\SIMPLE-STRINGIFY
                                                                   IL:SYMBOL-LISTIFY IL:COPY-STRING
                                                                   IL:\\SYMBOL-EQUALBASE))
           (IL:FUNCTIONS IL:\\FATCHARSEENP IL:\\PACKAGIFY IL:\\STRING-EQUALBASE IL:NUMERIC-UPCASE
                  IL:\\UPCASEBASE IL:APROPOS-SEARCH)
           (IL:STRUCTURES PACKAGE-HASHTABLE PACKAGE)
           (IL:FUNCTIONS PACKAGE-NAME PACKAGE-NICKNAMES PACKAGE-SHADOWING-SYMBOLS PACKAGE-USE-LIST
                  PACKAGE-USED-BY-LIST)
           (IL:FUNCTIONS IL:MAKE-PACKAGE-HASHTABLE PRINT-PACKAGE PRINT-PACKAGE-HASHTABLE)
           (IL:VARIABLES *PACKAGE* XCL::*UNSAFE-TO-DELETE-PACKAGE-NAMES* IL:*LISP-PACKAGE* IL:*KEYWORD-PACKAGE*
                  IL:*INTERLISP-PACKAGE* IL:HASHTABLE-SIZE-LIMIT IL:PACKAGE-REHASH-THRESHOLD)
           (IL:VARIABLES IL:PRIME-HASHTABLE-SIZES)
           ;; The package system's version of symbol creation
           (IL:FUNCTIONS MAKE-SYMBOL)
           ;; Packages are currently implemented using a free byte in the litatom pnamecell.  The byte is used as an index into a table.
           (IL:VARIABLES IL:*PACKAGE-FROM-NAME* IL:*PACKAGE-FROM-INDEX* XCL:*TOTAL-PACKAGES-LIMIT*
                  IL:*UNINTERNED-PACKAGE-INDEX*)
           (IL:FUNCTIONS IL:\\PKG-FIND-FREE-PACKAGE-INDEX)
           ;; Symbol package cell handlers.
           (IL:FUNCTIONS IL:SETF-SYMBOL-PACKAGE SYMBOL-PACKAGE)
           ;; Symbol hashing
           (IL:FUNCTIONS IL:SYMBOL-HASH IL:REHASH-FACTOR IL:SYMBOL-HASH-REPROBE IL:ENTRY-HASH)
           ;; Constructing packages
           (IL:FUNCTIONS IL:COUNT-PACKAGE-HASHTABLE IL:INTERNAL-SYMBOL-COUNT IL:EXTERNAL-SYMBOL-COUNT)
           (IL:FUNCTIONS IL:ENTER-NEW-NICKNAMES IL:MAKE-PRIME-HASHTABLE-SIZE)
           (IL:FUNCTIONS MAKE-PACKAGE)
           (IL:FNS XCL:DEFPACKAGE)
           ;; Package manipulations
           (IL:FUNCTIONS FIND-PACKAGE USE-PACKAGE IN-PACKAGE XCL:PKG-GOTO RENAME-PACKAGE XCL:DELETE-PACKAGE
                  EXPORT UNEXPORT IMPORT SHADOWING-IMPORT SHADOW UNUSE-PACKAGE)
           ;; Knowing about the package name space
           (IL:FUNCTIONS LIST-ALL-PACKAGES)
           ;; Putting symbols into packages
           (IL:FUNCTIONS IL:ADD-SYMBOL IL:WITH-SYMBOL)
           (IL:FUNCTIONS IL:INTERN* IL:FIND-SYMBOL*)
           (IL:FUNCTIONS INTERN FIND-SYMBOL)
           ;; Removing symbols from packages
           (IL:FUNCTIONS IL:NUKE-SYMBOL)
           (IL:FUNCTIONS UNINTERN IL:MOBY-UNINTERN)
           ;; Iterations over package symbols
           (IL:FUNCTIONS IL:\\INDEXATOMPNAME)
                                                          ; Defined in EXPORTS.ALL and used by the DO-SYMBOLS
                                                          ; macro
           (IL:DECLARE\: IL:EVAL@COMPILE                  ; These are used in expanding the DO-SYMBOLS macro, which
                                                          ; is used in this file.
                  (IL:FUNCTIONS IL:MAKE-DO-SYMBOLS-VARS IL:MAKE-DO-SYMBOLS-CODE))
           (IL:FUNCTIONS DO-EXTERNAL-SYMBOLS XCL:DO-LOCAL-SYMBOLS XCL:DO-INTERNAL-SYMBOLS DO-SYMBOLS
                  DO-ALL-SYMBOLS)
           ;; Finding symbols in a package or packages
           (IL:FUNCTIONS FIND-ALL-SYMBOLS)
           (IL:FUNCTIONS IL:BRIEFLY-DESCRIBE-SYMBOL APROPOS APROPOS-LIST)
           ;; Reader and printer's interface to packages (plus *PACKAGE-FROM-INDEX* above)
           (IL:FUNCTIONS IL:FIND-EXTERNAL-SYMBOL)
           (IL:FUNCTIONS IL:FIND-EXACT-SYMBOL IL:PACKAGE-NAME-AS-SYMBOL IL:\\FIND.PACKAGE.INTERNAL)
           ;; Proper compiler, readtable and package environment
           (IL:PROP (IL:FILETYPE IL:MAKEFILE-ENVIRONMENT)
```

```
                IL:LLPACKAGE)
          (IL:DECLARE\: IL:DONTEVAL@LOAD IL:DOEVAL@COMPILE IL:DONTCOPY IL:COMPILERVARS (IL:ADDVARS (IL:NLAMA
                                                                                                XCL:DEFPACKAGE
                                                                                                   )
                                                                                             (IL:NLAML)
                                                                                             (IL:LAMA)))))
```

;; The Xerox Lisp package system, based on CMU's Spice Lisp

;; Internal macros and definitions

```
(IL:DECLARE\: IL:EVAL@COMPILE IL:DONTCOPY

(DEFMACRO IL:PACKAGE-LISTIFY (IL:OBJ)
   "Return NIL or a list of packages given NIL or a package-or-string-or-symbol or list thereof, or die
   trying."
   `(LET ((IL:THING ,IL:OBJ))
         (COND
            ((NULL IL:THING)
             NIL)
            ((IL:LISTP IL:THING)
             (LET ((IL:RESULT NIL))
                  (DOLIST (PACKAGE IL:THING IL:RESULT)
                      (PUSH (IL:\\PACKAGIFY PACKAGE)
                            IL:RESULT))))
            (T (LIST (IL:\\PACKAGIFY IL:THING))))))

(DEFMACRO IL:\\SIMPLE-STRINGIFY (IL:OBJ)
   "If OBJ is a non-stringp-string or symbol, make it a stringp."
   `(LET ((IL:|obj| ,IL:OBJ))
         (COND
            ((IL:STRINGP IL:|obj|)
             IL:|obj|)
            ((OR (STRINGP IL:|obj|)
                 (SYMBOLP IL:|obj|))
             (IL:MKSTRING IL:|obj|))
            (T (IL:ERROR "Not a string or symbol " IL:|obj|)))))

(DEFMACRO IL:SYMBOL-LISTIFY (IL:OBJ)
   "Take a symbol-or-list-of-symbols and return a list, checking types."
   `(LET ((IL:THING ,IL:OBJ))
         (COND
            ((SYMBOLP IL:THING)
             (LIST IL:THING))
            ((IL:LISTP IL:THING)
             (DOLIST (IL:S IL:THING)
                 (UNLESS (SYMBOLP IL:S)
                         (IL:ERROR "Not a symbol." IL:S)))
             IL:THING)
            (T (IL:ERROR "Neither a symbol nor a list of symbols." IL:THING)))))

(DEFMACRO IL:COPY-STRING (STRING)
   `(IL:CONCAT ,STRING))

(DEFMACRO IL:\\SYMBOL-EQUALBASE (SYMBOL IL:BASE IL:OFFSET IL:LENGTH IL:FATP)
   "Compare a string, given in base offset length form, to a symbol's pname string"
   `(AND (EQL ,IL:LENGTH (IL:|ffetch| (SYMBOL IL:PNAMELENGTH) IL:|of| ,SYMBOL))
         (DO ((IL:I 0 (IL:ADD1 IL:I))
              (IL:SYMBOL-BASE (IL:|ffetch| (SYMBOL IL:PNAMEBASE) IL:|of| ,SYMBOL))
              (IL:SYMBOL-FATP (IL:|ffetch| (SYMBOL IL:FATPNAMEP) IL:|of| ,SYMBOL)))
             ((EQL IL:I ,IL:LENGTH)
              T)
           (IF (NOT (EQL (IL:\\GETBASECHAR IL:SYMBOL-FATP IL:SYMBOL-BASE (IL:ADD1 IL:I))
                         (IL:\\GETBASECHAR ,IL:FATP ,IL:BASE (IL:IPLUS ,IL:OFFSET IL:I))))
               (RETURN NIL)))))
)

(DEFMACRO IL:\\FATCHARSEENP (IL:BASE IL:OFFSET IL:LEN IL:FATP)
   `(AND ,IL:FATP (NOT (NULL (IL:FOR IL:I IL:FROM ,IL:OFFSET IL:TO (IL:SUB1 (IL:IPLUS ,IL:OFFSET ,IL:LEN))
                                  IL:SUCHTHAT (IL:IGREATERP (IL:\\GETBASEFAT ,IL:BASE IL:I)
                                                            IL:\\MAXTHINCHAR))))))

(DEFMACRO IL:\\PACKAGIFY (IL:OBJ)
   "If OBJ isn't already a package, turn the symbol or string into the package of that name."
   `(LET ((IL:|obj| ,IL:OBJ))
         (OR (COND
                ((PACKAGEP IL:|obj|)
                 IL:|obj|)
                ((STRINGP IL:|obj|)
```

```
                (FIND-PACKAGE IL:|obj|))
               ((SYMBOLP IL:|obj|)
                (FIND-PACKAGE (SYMBOL-NAME IL:|obj|)))
               (T NIL))
             (IL:ERROR "Not an existing package, string or symbol " IL:|obj|))))


(DEFMACRO IL:\\STRING-EQUALBASE (STRING IL:BASE IL:OFFSET IL:LENGTH IL:FATP)
   "Compare a string to another string, with the second given in base offset length form."
   `(AND (EQL ,IL:LENGTH (IL:|ffetch| (IL:STRINGP IL:LENGTH)
                                ,STRING))
         (DO ((IL:I 0 (IL:ADD1 IL:I))
              (IL:STRING-BASE (IL:|ffetch| (IL:STRINGP IL:BASE)
                                           ,STRING))
              (IL:STRING-OFFSET (IL:|ffetch| (IL:STRINGP IL:OFFST)
                                             ,STRING))
              (IL:STRING-FATP (IL:|ffetch| (IL:STRINGP IL:FATSTRINGP)
                                           ,STRING)))
             ((EQL IL:I ,IL:LENGTH)
              T)
           (IF (NOT (EQL (IL:\\GETBASECHAR IL:STRING-FATP IL:STRING-BASE (IL:IPLUS IL:STRING-OFFSET IL:I))
                         (IL:\\GETBASECHAR ,IL:FATP ,IL:BASE (IL:IPLUS ,IL:OFFSET IL:I))))
               (RETURN NIL)))))


(DEFMACRO IL:NUMERIC-UPCASE (IL:A)
   `(LET ((IL:N ,IL:A))
         (IF (AND (IL:IGEQ IL:N (IL:CHARCODE "a"))
                  (IL:ILEQ IL:N (IL:CHARCODE "z")))
             (IL:IDIFFERENCE IL:N 32)
             IL:N)))


(DEFUN IL:\\UPCASEBASE (IL:BASE IL:OFFSET IL:LENGTH IL:FATP)
   (IL:|for| IL:I IL:|from| IL:OFFSET IL:|to| (IL:IPLUS IL:OFFSET IL:LENGTH)
      IL:|do| (IL:\\PUTBASECHAR IL:FATP IL:BASE IL:I (IL:NUMERIC-UPCASE (IL:\\GETBASECHAR IL:FATP IL:BASE IL:I))))
)


(DEFUN IL:APROPOS-SEARCH (SYMBOL IL:BASE IL:OFFSET IL:LENGTH IL:FATP)
   "The symbol to substring comparison macro for APROPOS and APROPOS-LIST.  The string is assumed to already be
   uppercase."
   (DO ((IL:INDEX 0 (IL:ADD1 IL:INDEX))
        (IL:SYMBOL-BASE (IL:|ffetch| (SYMBOL IL:PNAMEBASE) IL:|of| SYMBOL))
        (IL:SYMBOL-FATP (IL:|ffetch| (SYMBOL IL:FATPNAMEP) IL:|of| SYMBOL))
        (IL:TERMINUS (IL:IDIFFERENCE (IL:|ffetch| (SYMBOL IL:PNAMELENGTH) IL:|of| SYMBOL)
                                     IL:LENGTH)))
       ((IL:IGREATERP IL:INDEX IL:TERMINUS)
        NIL)
     (IF (DO ((IL:JNDEX IL:OFFSET (IL:ADD1 IL:JNDEX))
              (IL:KNDEX IL:INDEX (IL:ADD1 IL:KNDEX))
              (IL:TERMINUS (IL:IPLUS IL:LENGTH IL:OFFSET)))
             ((EQL IL:JNDEX IL:TERMINUS)
              T)
           (UNLESS (EQL (IL:\\GETBASECHAR IL:FATP IL:BASE IL:JNDEX)
                        (IL:NUMERIC-UPCASE (IL:\\GETBASECHAR IL:SYMBOL-FATP IL:SYMBOL-BASE (IL:ADD1 IL:KNDEX)))
                        )
                   (RETURN NIL)))
         (RETURN T))))


(DEFSTRUCT (PACKAGE-HASHTABLE (:CONSTRUCTOR %MAKE-PACKAGE-HASHTABLE)
                              (:COPIER NIL)
                              (:PRINT-FUNCTION PRINT-PACKAGE-HASHTABLE))
   "Packages are implemented using a special kind of hashtable (this one).  It is an open hashtable with a
   parallel 8-bit I-vector of hash-codes.  The primary purpose of the hash for each entry is to reduce paging by
   allowing collisions and misses to be detected without paging in the symbol and pname for an entry.  If the
   hash for an entry doesn't match that for the symbol that we are looking for, then we can go on without
   touching the symbol, pname, or even hastable vector.  It turns out that, contrary to my expectations, paging
   is a very important consideration the design of the package representation.  Using a similar scheme without
   the entry hash, the fasloader was spending more than half its time paging in INTERN.  The hash code also
   indicates the status of an entry.  If it zero, the the entry is unused.  If it is one, then it is deleted.
   Double-hashing is used for collision resolution."
   TABLE
   HASH
   SIZE
   FREE
   DELETED)


(DEFSTRUCT (PACKAGE (:CONC-NAME %PACKAGE-)
                    (:CONSTRUCTOR %MAKE-PACKAGE)
                    (:PREDICATE PACKAGEP)
                    (:PRINT-FUNCTION PRINT-PACKAGE))
   INDEX
   (TABLES (LIST NIL))
   NAME NAMESYMBOL NICKNAMES (USE-LIST NIL)
```

```
   (USED-BY-LIST NIL)
   (EXTERNAL-ONLY NIL)
   INTERNAL-SYMBOLS EXTERNAL-SYMBOLS (SHADOWING-SYMBOLS NIL))


(DEFUN PACKAGE-NAME (PACKAGE)
   (%PACKAGE-NAME (IL:\\PACKAGIFY PACKAGE)))


(DEFUN PACKAGE-NICKNAMES (PACKAGE)
   (%PACKAGE-NICKNAMES (IL:\\PACKAGIFY PACKAGE)))


(DEFUN PACKAGE-SHADOWING-SYMBOLS (PACKAGE)
   (%PACKAGE-SHADOWING-SYMBOLS (IL:\\PACKAGIFY PACKAGE)))


(DEFUN PACKAGE-USE-LIST (PACKAGE)
   (%PACKAGE-USE-LIST (IL:\\PACKAGIFY PACKAGE)))


(DEFUN PACKAGE-USED-BY-LIST (PACKAGE)
   (%PACKAGE-USED-BY-LIST (IL:\\PACKAGIFY PACKAGE)))


(DEFUN IL:MAKE-PACKAGE-HASHTABLE (IL:SIZE &OPTIONAL (IL:RES (%MAKE-PACKAGE-HASHTABLE)))
   "Make a package hashtable having a prime number of entries at least as great as (/ size
   package-rehash-threshold).  If Res is supplied, then it is destructively modified to produce the result. This
   is useful when changing the size, since there are many pointers to the hashtable."
   (LET ((IL:N (IL:MAKE-PRIME-HASHTABLE-SIZE IL:SIZE)))
        (DECLARE (TYPE FIXNUM IL:N))
        (SETF (PACKAGE-HASHTABLE-TABLE IL:RES)
              (LIST (MAKE-ARRAY IL:N :ELEMENT-TYPE '(UNSIGNED-BYTE 32))))
        (SETF (PACKAGE-HASHTABLE-HASH IL:RES)
              (LIST (MAKE-ARRAY IL:N :ELEMENT-TYPE '(UNSIGNED-BYTE 8))))
        (LET ((IL:SIZE (IF (EQL IL:N IL:HASHTABLE-SIZE-LIMIT)
                           IL:HASHTABLE-SIZE-LIMIT
                           (IL:FIX (IL:FTIMES IL:N IL:PACKAGE-REHASH-THRESHOLD)))))
             (SETF (PACKAGE-HASHTABLE-SIZE IL:RES)
                   IL:SIZE)
             (SETF (PACKAGE-HASHTABLE-FREE IL:RES)
                   IL:SIZE))
        (SETF (PACKAGE-HASHTABLE-DELETED IL:RES)
              0)
        IL:RES))


(DEFUN PRINT-PACKAGE (PACKAGE STREAM DEPTH)
   (IL:PRIN3 "#<Package " STREAM)
   (IL:PRIN3 (%PACKAGE-NAME PACKAGE)
          STREAM)
   (IL:PRIN3 ">" STREAM))


(DEFUN PRINT-PACKAGE-HASHTABLE (TABLE STREAM DEPTH)
   (IL:PRIN3 "#<Package-hashtable: Size = " STREAM)
   (IL:PRIN3 (PACKAGE-HASHTABLE-SIZE TABLE)
          STREAM)
   (IL:PRIN3 ", Free = " STREAM)
   (IL:PRIN3 (PACKAGE-HASHTABLE-FREE TABLE)
          STREAM)
   (IL:PRIN3 ", Deleted = " STREAM)
   (IL:PRIN3 (PACKAGE-HASHTABLE-DELETED TABLE)
          STREAM)
   (IL:PRIN3 ">" STREAM))


(DEFVAR *PACKAGE* NIL
   "The current package, in which read symbols are intern'ed.")


(DEFVAR XCL::*UNSAFE-TO-DELETE-PACKAGE-NAMES* '("LISP" "INTERLISP" "XEROX-COMMON-LISP")
                                              "Packages whose deletion requires confirmation.")


(XCL:DEFGLOBALVAR IL:*LISP-PACKAGE* NIL
   "Global for internal references to the lisp package.")


(XCL:DEFGLOBALVAR IL:*KEYWORD-PACKAGE* NIL
   "Global for internal references to the keyword package.")


(XCL:DEFGLOBALVAR IL:*INTERLISP-PACKAGE* NIL
   "Global for internal references to the interlisp package.")
```

```
(DEFCONSTANT IL:HASHTABLE-SIZE-LIMIT 65521
   "The maximum (inclusive, prime) limit to the size of a hashtable.")


(DEFPARAMETER IL:PACKAGE-REHASH-THRESHOLD 0.5
   "The maximum density allowed in a package hashtable")


(DEFCONSTANT IL:PRIME-HASHTABLE-SIZES
   '(7 19 67 113 199 293 397 887 1373 2347 4297 8191 15991 40763 65521)
   "Some valid (prime) hashtable sizes.")
```

;; The package system's version of symbol creation

```
(DEFUN MAKE-SYMBOL (IL:PRINT-NAME)
   "Make an uninterned symbol."
   (IF (NOT (STRINGP IL:PRINT-NAME))
       (IL:ERROR "Not a string " IL:PRINT-NAME))
   (IL:SETQ IL:PRINT-NAME (IL:MKSTRING IL:PRINT-NAME))
   (LET ((IL:FATP (IL:|ffetch| (IL:STRINGP IL:FATSTRINGP) IL:|of| IL:PRINT-NAME))
         (IL:BASE (IL:|ffetch| (IL:STRINGP IL:BASE) IL:|of| IL:PRINT-NAME))
         (IL:LEN (IL:|ffetch| (IL:STRINGP IL:LENGTH) IL:|of| IL:PRINT-NAME))
         (IL:OFFST (IL:|ffetch| (IL:STRINGP IL:OFFST) IL:|of| IL:PRINT-NAME)))
        (IL:UNINTERRUPTABLY
            (IL:\\CREATE.SYMBOL IL:BASE IL:OFFST IL:LEN IL:FATP (IL:\\FATCHARSEENP IL:BASE IL:OFFST IL:LEN
                                                                  IL:FATP)))))
```

;; Packages are currently implemented using a free byte in the litatom pnamecell. The byte is used as an index into a table.

```
(XCL:DEFGLOBALVAR IL:*PACKAGE-FROM-NAME* (IL:HASHARRAY 255 'IL:ERROR 'IL:STRINGHASHBITS 'IL:STREQUAL)
                                         "An equal hashtable from package names to packages.")


(XCL:DEFGLOBALVAR IL:*PACKAGE-FROM-INDEX* (MAKE-ARRAY 256 ':INITIAL-ELEMENT NIL)
                                          "Index to package converter.")


(DEFCONSTANT XCL:*TOTAL-PACKAGES-LIMIT* 255
   "The total number of packages that the system may have (excluding the 'uninterned' package).")


(DEFCONSTANT IL:*UNINTERNED-PACKAGE-INDEX* 0
   "Package index value for uninterned symbols.  The function \\PKG-FIND-FREE-PACKAGE-INDEX and the constant
   *UNINTERNED-PACKAGE-INDEX* are arranged so that SYMBOL-PACKAGE can find NIL in the index vector and NIL can
   also be the free slot marker.  *UNINTERNED-PACKAGE-INDEX* must be zero, otherwise change
   \\PKG-FIND-FREE-PACKAGE-INDEX .")


(DEFUN IL:\\PKG-FIND-FREE-PACKAGE-INDEX ()
   "Return the next free table index for a package.  Starts counting at 1 because 0 is for uninterned symbols."
   (DO ((IL:I 1 (IL:ADD1 IL:I)))
       ((EQL IL:I XCL:*TOTAL-PACKAGES-LIMIT*)
        (ERROR "Package space full" NIL))
     (DECLARE (SPECIAL IL:*PACKAGE-FROM-INDEX*))
     (IF (NULL (AREF IL:*PACKAGE-FROM-INDEX* IL:I))
         (RETURN IL:I))))
```

;; Symbol package cell handlers.

```
(DEFUN IL:SETF-SYMBOL-PACKAGE (IL:OBJ IL:VALUE)
   (IL:|freplace| (SYMBOL PACKAGE) IL:|of| IL:OBJ IL:|with| IL:VALUE)
   IL:VALUE)


(DEFUN SYMBOL-PACKAGE (SYMBOL)
   (IL:|ffetch| (SYMBOL PACKAGE) IL:|of| SYMBOL))
```

;; Symbol hashing

```
(DEFMACRO IL:SYMBOL-HASH (IL:BASE IL:OFFST IL:LEN IL:FATP)
   "Returns the atom hash of the given string"
   `(IF (EQL 0 ,IL:LEN)
        0
        (DO* ((IL:TERMINUS (IL:IPLUS ,IL:OFFST ,IL:LEN))
              (IL:HASH (IL:LLSH (IL:UNLESSRDSYS (COND
                                                  (,IL:FATP (LOGAND (IL:\\GETBASEFAT ,IL:BASE ,IL:OFFST)
                                                                    255))
                                                  (T (IL:\\GETBASETHIN ,IL:BASE ,IL:OFFST)))
                                                (IL:NTHCHARCODE ,IL:BASE ,IL:OFFST))
                                8)
                       (IL:IPLUS16 (IL:IPLUS16 (IL:SETQ IL:HASH (IL:IPLUS16 IL:HASH (IL:LLSH (LOGAND IL:HASH 4095)
```

```
                                                                                2)))
                                          (IL:LLSH (LOGAND IL:HASH 255)
                                                8))
                                   (IL:UNLESSRDSYS (COND
                                                   (,IL:FATP (LOGAND (IL:\\GETBASEFAT ,IL:BASE IL:CHAR#)
                                                                     255))
                                                   (T (IL:\\GETBASETHIN ,IL:BASE IL:CHAR#)))
                                                (IL:NTHCHARCODE ,IL:BASE IL:CHAR#))))
                    (IL:CHAR# (IL:ADD1 ,IL:OFFST)
                              (IL:ADD1 IL:CHAR#)))
                   ((IL:IGEQ IL:CHAR# IL:TERMINUS)
                    IL:HASH))))

(DEFMACRO IL:REHASH-FACTOR (IL:HASH IL:TABLE-LENGTH)
   `(IL:ADD1 (IL:IREMAINDER ,IL:HASH (IL:IDIFFERENCE ,IL:TABLE-LENGTH 2))))

(DEFMACRO IL:SYMBOL-HASH-REPROBE (IL:HASH IL:REHASH-FACTOR IL:TABLE-LENGTH)
   `(IL:IREMAINDER (IL:IPLUS ,IL:HASH ,IL:REHASH-FACTOR)
           ,IL:TABLE-LENGTH))

(DEFMACRO IL:ENTRY-HASH (IL:STRING-LENGTH SXHASH)
   "Compute a number from the sxhash of the pname and the length which must be between 2 and 255."
   `(IL:IPLUS (IL:IREMAINDER (LOGXOR ,IL:STRING-LENGTH ,SXHASH (IL:LRSH ,SXHASH 8)
                                     (IL:LRSH ,SXHASH 16)
                                     (IL:LRSH ,SXHASH 19))
                             254)
           2))

;; Constructing packages

(DEFMACRO IL:COUNT-PACKAGE-HASHTABLE (IL:TABLE)
   "Return two values: free elements and total size."
   `(LET ((IL:SIZE (IL:IDIFFERENCE (PACKAGE-HASHTABLE-SIZE ,IL:TABLE)
                           (PACKAGE-HASHTABLE-DELETED ,IL:TABLE))))
         (VALUES (IL:IDIFFERENCE IL:SIZE (PACKAGE-HASHTABLE-FREE ,IL:TABLE))
                 IL:SIZE)))

(DEFUN IL:INTERNAL-SYMBOL-COUNT (PACKAGE)
   (IF (%PACKAGE-EXTERNAL-ONLY PACKAGE)
       0
       (IL:COUNT-PACKAGE-HASHTABLE (%PACKAGE-INTERNAL-SYMBOLS PACKAGE))))

(DEFUN IL:EXTERNAL-SYMBOL-COUNT (PACKAGE)
   (IL:COUNT-PACKAGE-HASHTABLE (%PACKAGE-EXTERNAL-SYMBOLS PACKAGE)))

(DEFUN IL:ENTER-NEW-NICKNAMES (PACKAGE IL:NICKNAMES)
   "Enter any new Nicknames for Package into *package-names*.  If there is a conflict then give the user a
   chance to do something about it."
   (DECLARE (SPECIAL IL:*PACKAGE-FROM-NAME*))
   (CHECK-TYPE IL:NICKNAMES LIST)
   (DOLIST (IL:N IL:NICKNAMES)
       (IL:SETQ IL:N (IL:\\SIMPLE-STRINGIFY IL:N))
       (LET ((IL:FOUND (IL:GETHASH IL:N IL:*PACKAGE-FROM-NAME*)))
            (COND
               ((NOT IL:FOUND)
                (IL:PUTHASH IL:N PACKAGE IL:*PACKAGE-FROM-NAME*)
                (PUSH IL:N (%PACKAGE-NICKNAMES PACKAGE)))
               ((EQ IL:FOUND PACKAGE))
               ((IL:STREQUAL (%PACKAGE-NAME IL:FOUND)
                             IL:N)
                (IL:ERROR (IL:CONCAT IL:N "is already a package name, so it cannot be a nickname for "
                                     (%PACKAGE-NAME PACKAGE))))
               (T (IL:ERROR (IL:CONCAT IL:N " is already a nickname for " (%PACKAGE-NAME IL:FOUND)))
                  (IL:PUTHASH IL:N PACKAGE IL:*PACKAGE-FROM-NAME*)
                  (PUSH IL:N (%PACKAGE-NICKNAMES PACKAGE)))))))

(DEFUN IL:MAKE-PRIME-HASHTABLE-SIZE (IL:N)
   "Find an appropriate size based on the expected number of elements, N, the rehash threshold and the limit on
   array size."
   (LET ((IL:N (IL:LOGOR (IL:FIX (IL:FQUOTIENT IL:N IL:PACKAGE-REHASH-THRESHOLD))
                         1)))
        (DOLIST (IL:X IL:PRIME-HASHTABLE-SIZES IL:HASHTABLE-SIZE-LIMIT)
            (WHEN (IL:IGEQ IL:X IL:N)
                  (RETURN IL:X)))))

(DEFUN MAKE-PACKAGE (NAME &KEY (USE '("LISP"))
                               NICKNAMES PREFIX-NAME (EXTERNAL-ONLY NIL)
                               (INTERNAL-SYMBOLS 10)
```

```
                              (EXTERNAL-SYMBOLS 10))
   "Check for package name conflicts in name and nicknames, then make the package.  Do a use-package for each
   thing in the use list so that checking for conflicting exports among used packages is done."
   (DECLARE (SPECIAL IL:*PACKAGE-FROM-INDEX* IL:*PACKAGE-FROM-NAME*))
   (WHEN (FIND-PACKAGE NAME)
       (IL:ERROR (IL:CONCAT "Package " NAME " already exists.")))
   (SETF NAME (IL:MKSTRING NAME))
   (SETF PREFIX-NAME (MAKE-SYMBOL (OR PREFIX-NAME NAME)))
   (LET* ((%PACKAGE-INDEX (IL:\\PKG-FIND-FREE-PACKAGE-INDEX))
          (PACKAGE (%MAKE-PACKAGE :NAME NAME :NAMESYMBOL PREFIX-NAME :EXTERNAL-ONLY EXTERNAL-ONLY
                                  :INTERNAL-SYMBOLS (IF (NOT EXTERNAL-ONLY)
                                                        (IL:MAKE-PACKAGE-HASHTABLE INTERNAL-SYMBOLS)
                                                      NIL)
                                  :EXTERNAL-SYMBOLS
                                  (IL:MAKE-PACKAGE-HASHTABLE EXTERNAL-SYMBOLS)
                                  :INDEX %PACKAGE-INDEX)))
          (USE-PACKAGE USE PACKAGE)
          (IL:ENTER-NEW-NICKNAMES PACKAGE (IF (IL:STREQUAL NAME (SYMBOL-NAME PREFIX-NAME))
                                              NICKNAMES
                                              (CONS PREFIX-NAME NICKNAMES)))
          (IL:PUTHASH NAME PACKAGE IL:*PACKAGE-FROM-NAME*)
          (SETF (AREF IL:*PACKAGE-FROM-INDEX* %PACKAGE-INDEX)
                PACKAGE)))

(IL:DEFINEQ

(XCL:DEFPACKAGE
  (IL:NLAMBDA IL:ARGS                                          ; Edited  2-Dec-87 10:39 by raf
    (IL:SETQ IL:ARGS (XCL:REMOVE-COMMENTS IL:ARGS))
    (LET ((PACKAGE (FIND-PACKAGE (CAR IL:ARGS))))
         (COND
            ((PACKAGEP PACKAGE)                                ; If one already exists, test compatability of package definitions
             (IL:|for| IL:OPTION IL:|in| (CDR IL:ARGS)
                IL:|do| (LET* ((IL:KEY (COND
                                          ((KEYWORDP IL:OPTION)
                                           IL:OPTION)
                                          ((IL:LISTP IL:OPTION)
                                           (CAR IL:OPTION))
                                          (T (IL:ERROR "Bad option for defpackage " IL:OPTION))))
                               (VALUES (COND
                                          ((KEYWORDP IL:OPTION)
                                           (LIST T))
                                          ((IL:LISTP IL:OPTION)
                                           (CDR IL:OPTION))
                                          (T (IL:ERROR "Bad option for defpackage " IL:OPTION)))))
                          (IL:SELECTQ IL:KEY
                                ((:INTERNAL-SYMBOLS :EXTERNAL-SYMBOLS)
                                     NIL)
                                (:EXTERNAL-ONLY (IF (NOT (%PACKAGE-EXTERNAL-ONLY PACKAGE))
                                                    (IL:ERROR "Package NOT :external-only as asserted by
                                                           defpackage: " PACKAGE)))
                                (:PREFIX-NAME (SETF (%PACKAGE-NAMESYMBOL PACKAGE)
                                                    (MAKE-SYMBOL (CAR VALUES))))
                                (:USE (USE-PACKAGE VALUES PACKAGE))
                                (:NICKNAMES (IL:ENTER-NEW-NICKNAMES PACKAGE VALUES))
                                (:EXPORT (EXPORT (IL:FOR IL:SYMBOL IL:IN VALUES
                                                         IL:COLLECT (IL:IF (IL:LITATOM IL:SYMBOL)
                                                                           IL:THEN IL:SYMBOL
                                                                        IL:ELSEIF (IL:STRINGP IL:SYMBOL)
                                                                           IL:THEN (INTERN IL:SYMBOL PACKAGE)
                                                                        IL:ELSE (IL:ERROR "Bad object in :export
                                                                                          option of defpackage "
                                                                                          IL:SYMBOL)))
                                                 PACKAGE))
                                (:IMPORT (IMPORT VALUES PACKAGE))
                                ((:SHADOW :SHADOWING-IMPORT)
                                     (LET ((IL:SYMBOLS-TO-SHADOW (IL:MAPCONC
                                                                  VALUES
                                                                  (IL:FUNCTION (IL:LAMBDA (SYMBOL)
                                                                                 (COND
                                                                                    ((NOT (IL:MEMB
                                                                                           SYMBOL
                                                                                           (
                                                                                    %PACKAGE-SHADOWING-SYMBOLS
                                                                                           PACKAGE)))
                                                                                     (LIST SYMBOL))))))))
                                          (IL:SELECTQ IL:KEY
                                                (:SHADOW (SHADOW IL:SYMBOLS-TO-SHADOW PACKAGE))
                                                (:SHADOWING-IMPORT
                                                     (SHADOWING-IMPORT IL:SYMBOLS-TO-SHADOW PACKAGE))
                                                NIL)))
                                (IL:ERROR "Bad keyword for defpackage " IL:KEY)))))
            (T                                                 ; Otherwise, make a new package to spec
               (LET ((IL:POST-MAKE-FORMS NIL))
                    (IL:SETQ PACKAGE (IL:APPLY
                                      'MAKE-PACKAGE
                                      (CONS (CAR IL:ARGS)
```

```
                                        (IL:|for| IL:OPTION IL:|in| (CDR IL:ARGS)
                                           IL:|join| (LET ((IL:KEY (COND
                                                                     ((KEYWORDP IL:OPTION)
                                                                      IL:OPTION)
                                                                     ((IL:LISTP IL:OPTION)
                                                                      (CAR IL:OPTION))
                                                                     (T (IL:ERROR "Bad option for defpackage "
                                                                                  IL:OPTION))))
                                                           (VALUES (COND
                                                                     ((KEYWORDP IL:OPTION)
                                                                      (LIST T))
                                                                     ((IL:LISTP IL:OPTION)
                                                                      (CDR IL:OPTION))
                                                                     (T (IL:ERROR "Bad option for defpackage "
                                                                                  IL:OPTION)))))
                                                          (IL:SELECTQ IL:KEY
                                                                ((:USE :NICKNAMES)
                                                                      (LIST IL:KEY (IL:|if| (CAR VALUES)
                                                                                            IL:|then| VALUES
                                                                                           IL:|else|
                                                                ; Handles case where NIL is being used to explicitly say the
                                                                ; package's :USE list is empty, since the default is to use LISP.
                                                                                                 NIL)))
                                                                ((:PREFIX-NAME :INTERNAL-SYMBOLS
                                                                        :EXTERNAL-SYMBOLS :EXTERNAL-ONLY)
                                                                      (LIST IL:KEY (CAR VALUES)))
                                                                ((:SHADOW :EXPORT :IMPORT :SHADOWING-IMPORT)
                                                                      (IL:SETQ IL:POST-MAKE-FORMS
                                                                       (CONS (CONS IL:KEY VALUES)
                                                                             IL:POST-MAKE-FORMS))
                                                                      NIL)
                                                                (IL:ERROR "Bad keyword for defpackage " IL:KEY)
                                                           ))))))
                    (IL:MAPC IL:POST-MAKE-FORMS
                           (IL:FUNCTION (IL:LAMBDA (IL:FORM)
                                        (IL:SELECTQ (CAR IL:FORM)
                                              (:SHADOW (SHADOW (CDR IL:FORM)
                                                               PACKAGE))
                                              (:EXPORT (EXPORT (IL:FOR IL:SYMBOL IL:IN (CDR IL:FORM)
                                                                       IL:COLLECT (IL:IF (IL:LITATOM IL:SYMBOL)
                                                                                         IL:THEN IL:SYMBOL
                                                                                      IL:ELSEIF (IL:STRINGP IL:SYMBOL
                                                                                                )
                                                                                         IL:THEN (INTERN IL:SYMBOL
                                                                                                  PACKAGE)
                                                                                      IL:ELSE (IL:ERROR "Bad object
                                                                                                        in :export
                                                                                                        option of
                                                                                                        defpackage "
                                                                                                        IL:SYMBOL)))
                                                               PACKAGE))
                                              (:IMPORT (IMPORT (CDR IL:FORM)
                                                               PACKAGE))
                                              (:SHADOWING-IMPORT
                                                    (SHADOWING-IMPORT (CDR IL:FORM)
                                                                      PACKAGE))
                                              (IL:SHOULDNT "Bogus form on post-make-forms")))))))
          (PACKAGE-NAME PACKAGE))))

)

;; Package manipulations


(DEFUN FIND-PACKAGE (IL:NAME)
   "Given a name, find the package with that name or nickname"
   (DECLARE (SPECIAL IL:*PACKAGE-FROM-NAME*))
   (IL:GETHASH (IL:MKSTRING IL:NAME)
          IL:*PACKAGE-FROM-NAME* NIL))


(DEFUN USE-PACKAGE (IL:PACKAGES-TO-USE &OPTIONAL (PACKAGE *PACKAGE*))
   "Make a package use (inherit) symbols from others.  Checks for name-conflicts."
   (DECLARE (SPECIAL *PACKAGE*))
   (IL:SETQ PACKAGE (IL:\\PACKAGIFY PACKAGE))

   ;; Loop over each package, use'ing one at a time...

   (DOLIST (IL:PKG (IL:PACKAGE-LISTIFY IL:PACKAGES-TO-USE))
       (UNLESS (IL:FMEMB IL:PKG (%PACKAGE-USE-LIST PACKAGE))
           (LET ((IL:CSET NIL)
                 (IL:SHADOWING-SYMBOLS (%PACKAGE-SHADOWING-SYMBOLS PACKAGE))
                 (IL:USE-LIST (%PACKAGE-USE-LIST PACKAGE)))

                ;; If the number of symbols already available is less than the number to be inherited then it is faster to run the test the other way.
                ;; This is particularly valuable in the case of a new package use'ing Lisp.

                (COND
                   ((IL:ILESSP (IL:IPLUS (IL:INTERNAL-SYMBOL-COUNT PACKAGE)
```

```
                                    (IL:EXTERNAL-SYMBOL-COUNT PACKAGE)
                                    (LET ((IL:RES 0))
                                         (DOLIST (IL:P IL:USE-LIST IL:RES)
                                                 (INCF IL:RES (IL:EXTERNAL-SYMBOL-COUNT IL:P)))))
                              (IL:EXTERNAL-SYMBOL-COUNT IL:PKG))
                        (DO-SYMBOLS (IL:SYM PACKAGE)
                              (MULTIPLE-VALUE-BIND (IL:S IL:W)
                                  (IL:FIND-EXTERNAL-SYMBOL (SYMBOL-NAME IL:SYM)
                                          IL:PKG)
                                (WHEN (AND IL:W (NOT (EQ IL:S IL:SYM))
                                           (NOT (IL:FMEMB IL:SYM IL:SHADOWING-SYMBOLS)))
                                   (PUSHNEW IL:SYM IL:CSET :TEST 'EQ))))
                        (DOLIST (IL:P IL:USE-LIST)
                            (DO-EXTERNAL-SYMBOLS (IL:SYM IL:P)
                                  (MULTIPLE-VALUE-BIND (IL:S IL:W)
                                      (IL:FIND-EXTERNAL-SYMBOL (SYMBOL-NAME IL:SYM)
                                             IL:PKG)
                                    (WHEN (AND IL:W (NOT (EQ IL:S IL:SYM))
                                                (NOT (IL:FMEMB (INTERN (SYMBOL-NAME IL:SYM)
                                                                       PACKAGE)
                                                               IL:SHADOWING-SYMBOLS)))
                                       (PUSHNEW IL:SYM IL:CSET :TEST 'EQ))))))
                       (T (DO-EXTERNAL-SYMBOLS (IL:SYM IL:PKG)
                                 (MULTIPLE-VALUE-BIND (IL:S IL:W)
                                     (FIND-SYMBOL (SYMBOL-NAME IL:SYM)
                                            PACKAGE)
                                   (WHEN (AND IL:W (NOT (EQ IL:S IL:SYM))
                                              (NOT (IL:FMEMB IL:S IL:SHADOWING-SYMBOLS)))
                                      (PUSHNEW IL:S IL:CSET :TEST 'EQ))))))
                 (WHEN IL:CSET (IL:RESOLVE-USE-PACKAGE-CONFLICT IL:PKG IL:CSET PACKAGE)))
              (PUSH IL:PKG (%PACKAGE-USE-LIST PACKAGE))
              (PUSH (%PACKAGE-EXTERNAL-SYMBOLS IL:PKG)
                    (CDR (%PACKAGE-TABLES PACKAGE)))
              (PUSH PACKAGE (%PACKAGE-USED-BY-LIST IL:PKG))))
   T)

(DEFUN IN-PACKAGE (IL:NAME &REST IL:KEYS &KEY IL:NICKNAMES IL:USE)
   "Like Make-Package, but also makes the created package current."
   (DECLARE (SPECIAL *PACKAGE*))
   (LET ((PACKAGE (FIND-PACKAGE IL:NAME)))
        (COND
           (PACKAGE (USE-PACKAGE IL:USE PACKAGE)
                  (IL:ENTER-NEW-NICKNAMES PACKAGE IL:NICKNAMES)
                  (IL:SETQ *PACKAGE* PACKAGE))
           (T (IL:SETQ *PACKAGE* (APPLY 'MAKE-PACKAGE IL:NAME IL:KEYS))))))


(DEFUN XCL:PKG-GOTO (XCL::NAME &REST XCL::KEYS)
   "Like in-package, but confirms creation of new packages."
   (WHEN (OR (PACKAGEP (FIND-PACKAGE XCL::NAME))
             (Y-OR-N-P "Create new package ~a?" XCL::NAME))
       (APPLY 'IN-PACKAGE XCL::NAME XCL::KEYS)))


(DEFUN RENAME-PACKAGE (PACKAGE IL:NAME &OPTIONAL IL:NICKNAMES IL:PREFIX-NAME)
   "Change the name if we can, blast any old nicknames and then add in any new ones."
   (DECLARE (SPECIAL IL:*PACKAGE-FROM-NAME*))
   (SETF PACKAGE (IL:\\PACKAGIFY PACKAGE))
   (SETF IL:NAME (IL:\\SIMPLE-STRINGIFY IL:NAME))
   (SETF IL:PREFIX-NAME (MAKE-SYMBOL (OR IL:PREFIX-NAME IL:NAME)))
   (LET ((IL:FOUND (FIND-PACKAGE IL:NAME)))
        (UNLESS (OR (NOT IL:FOUND)
                    (EQ IL:FOUND PACKAGE))
               (ERROR "A package named ~S already exists." IL:NAME))
        (REMHASH (%PACKAGE-NAME PACKAGE)
                 IL:*PACKAGE-FROM-NAME*)
        (SETF (%PACKAGE-NAME PACKAGE)
              IL:NAME)
        (SETF (%PACKAGE-NAMESYMBOL PACKAGE)
              IL:PREFIX-NAME)
        (IL:PUTHASH IL:NAME PACKAGE IL:*PACKAGE-FROM-NAME*)
        (DOLIST (IL:N (%PACKAGE-NICKNAMES PACKAGE))
            (REMHASH IL:N IL:*PACKAGE-FROM-NAME*))
        (SETF (%PACKAGE-NICKNAMES PACKAGE)
              NIL)
        (IL:ENTER-NEW-NICKNAMES PACKAGE IL:NICKNAMES)
        PACKAGE))

(DEFUN XCL:DELETE-PACKAGE (PACKAGE)

;;; All other packages unuse this one, all the package's symbols are uninterned and then its name is removed.

   (DECLARE (SPECIAL IL:*PACKAGE-FROM-NAME*))
   (SETF PACKAGE (IL:\\PACKAGIFY PACKAGE))
   (WHEN (OR (AND (EQ PACKAGE *PACKAGE*)
```

```
                (NOT (YES-OR-NO-P "About to delete the current package; this is dangerous, are you sure?")))
           (AND (MEMBER (%PACKAGE-NAME PACKAGE)
                        XCL::*UNSAFE-TO-DELETE-PACKAGE-NAMES* :TEST 'STRING=)
                (NOT (YES-OR-NO-P "About to delete the ~a package; this is dangerous, are you sure?"
                                  (%PACKAGE-NAME PACKAGE)))))
      (RETURN-FROM XCL:DELETE-PACKAGE NIL))
  (DOLIST (XCL::USER (%PACKAGE-USED-BY-LIST PACKAGE))
    (UNUSE-PACKAGE PACKAGE XCL::USER))
  (DOLIST (XCL::USED (%PACKAGE-USE-LIST PACKAGE))
    (UNUSE-PACKAGE XCL::USED PACKAGE))
  (XCL:DO-LOCAL-SYMBOLS (SYMBOL PACKAGE)
    (WHEN (EQ PACKAGE (SYMBOL-PACKAGE SYMBOL))
      (UNINTERN SYMBOL PACKAGE)))
  (REMHASH (%PACKAGE-NAME PACKAGE)
    IL:*PACKAGE-FROM-NAME*)
  (DOLIST (IL:NAME (%PACKAGE-NICKNAMES PACKAGE))
    (REMHASH IL:NAME IL:*PACKAGE-FROM-NAME*))
  (SETF (AREF IL:*PACKAGE-FROM-INDEX* (%PACKAGE-INDEX PACKAGE))
    NIL)
  T)


(DEFUN EXPORT (IL:SYMBOLS &OPTIONAL (PACKAGE *PACKAGE*))
  "Make the symbols external in the package."
  (DECLARE (SPECIAL *PACKAGE*))
  (SETF PACKAGE (IL:\\PACKAGIFY PACKAGE))
  (LET ((IL:SYMS NIL))
       ;; Punt any symbols that are already external.
       (DOLIST (IL:SYM (IL:SYMBOL-LISTIFY IL:SYMBOLS))
           (MULTIPLE-VALUE-BIND (IL:S IL:W)
               (IL:FIND-EXTERNAL-SYMBOL (SYMBOL-NAME IL:SYM)
                   PACKAGE)
             (UNLESS (OR IL:W (IL:FMEMB IL:SYM IL:SYMS))
                 (PUSH IL:SYM IL:SYMS))))
       ;; Find symbols and packages with conflicts.
       (LET ((IL:USED-BY (%PACKAGE-USED-BY-LIST PACKAGE))
             (IL:CPACKAGES NIL)
             (IL:CSET NIL))
            (DOLIST (IL:SYM IL:SYMS)
                (LET ((IL:NAME (SYMBOL-NAME IL:SYM)))
                     (DOLIST (IL:P IL:USED-BY)
                         (MULTIPLE-VALUE-BIND (IL:S IL:W)
                             (FIND-SYMBOL IL:NAME IL:P)
                           (WHEN (AND IL:W (NOT (EQ IL:S IL:SYM))
                                      (NOT (IL:FMEMB IL:S (%PACKAGE-SHADOWING-SYMBOLS IL:P))))
                               (PUSHNEW IL:SYM IL:CSET)
                               (PUSHNEW IL:P IL:CPACKAGES))))))
            (WHEN IL:CSET                                          ; Resolve conflict
                (IL:SETQ IL:SYMS (IL:RESOLVE-EXPORT-CONFLICT PACKAGE IL:CSET IL:CPACKAGES IL:SYMS))))
       ;; Check that all symbols are available.  If not, ask to import them.
       (LET ((IL:MISSING NIL)
             (IL:IMPORTS NIL))
            (DOLIST (IL:SYM IL:SYMS)
                (MULTIPLE-VALUE-BIND (IL:S IL:W)
                    (FIND-SYMBOL (SYMBOL-NAME IL:SYM)
                        PACKAGE)
                  (COND
                     ((NOT (AND IL:W (EQ IL:S IL:SYM)))
                      (PUSH IL:SYM IL:MISSING))
                     ((EQ IL:W :INHERITED)
                      (PUSH IL:SYM IL:IMPORTS)))))
            (WHEN IL:MISSING                                       ; Get missing symbols
                (IL:RESOLVE-EXPORT-MISSING PACKAGE IL:MISSING))
            (WHEN IL:IMPORTS                                       ; Get inherited symbols
                (IMPORT IL:IMPORTS PACKAGE)))
       ;; And now we export the symbols.
       (LET ((IL:INTERNAL (%PACKAGE-INTERNAL-SYMBOLS PACKAGE))
             (IL:EXTERNAL (%PACKAGE-EXTERNAL-SYMBOLS PACKAGE)))
            (DOLIST (IL:SYM IL:SYMS)
                (IF (NOT (%PACKAGE-EXTERNAL-ONLY PACKAGE))
                    (IL:NUKE-SYMBOL IL:INTERNAL (SYMBOL-NAME IL:SYM)))
                (IL:ADD-SYMBOL IL:EXTERNAL IL:SYM)))
       T))


(DEFUN UNEXPORT (IL:SYMBOLS &OPTIONAL (PACKAGE *PACKAGE*))
  "Check that all symbols are available, then move from external to internal."
  (DECLARE (SPECIAL *PACKAGE*))
  (SETF PACKAGE (IL:\\PACKAGIFY PACKAGE))
  (WHEN (%PACKAGE-EXTERNAL-ONLY PACKAGE)
      (IL:ERROR (IL:CONCAT "Can't unexport symbols " IL:SYMBOLS " from an external-only package " PACKAGE)))
  (LET ((IL:SYMS NIL))
       (DOLIST (IL:SYM (IL:SYMBOL-LISTIFY IL:SYMBOLS))
```

```
            (MULTIPLE-VALUE-BIND (IL:S IL:W)
                (FIND-SYMBOL (SYMBOL-NAME IL:SYM)
                      PACKAGE)
              (COND
                 ((OR (NOT IL:W)
                      (NOT (EQ IL:S IL:SYM)))
                  (ERROR "~S is not available in the ~A package." IL:SYM (SYMBOL-NAME PACKAGE)))
                 ((EQ IL:W :EXTERNAL)
                  (PUSHNEW IL:SYM IL:SYMS)))))
        (LET ((IL:INTERNAL (%PACKAGE-INTERNAL-SYMBOLS PACKAGE))
              (IL:EXTERNAL (%PACKAGE-EXTERNAL-SYMBOLS PACKAGE)))
             (DOLIST (IL:SYM IL:SYMS)
                 (IL:ADD-SYMBOL IL:INTERNAL IL:SYM)
                 (IL:NUKE-SYMBOL IL:EXTERNAL (SYMBOL-NAME IL:SYM))))
        T))


(DEFUN IMPORT (SYMBOLS &OPTIONAL (PACKAGE *PACKAGE*))
   "Make the symbol internal in the package, noting name conflicts."
   (DECLARE (SPECIAL *PACKAGE*))
   (SETF PACKAGE (IL:\\PACKAGIFY PACKAGE))
   (LET ((SYMS NIL)
         (CSET NIL))
        (DOLIST (SYM (IL:SYMBOL-LISTIFY SYMBOLS))
            (MULTIPLE-VALUE-BIND (S W)
                (FIND-SYMBOL (SYMBOL-NAME SYM)
                      PACKAGE)
              (COND
                 ((NOT W)
                  (LET ((FOUND (MEMBER SYM SYMS :TEST 'IL:STREQUAL)))
                       (IF FOUND
                           (WHEN (NOT (EQ (CAR FOUND)
                                          SYM))
                                 (PUSH SYM CSET))
                           (PUSH SYM SYMS))))
                 ((NOT (EQ S SYM))
                  (PUSH SYM CSET))
                 ((EQ W :INHERITED)
                  (PUSH SYM SYMS)))))
        (WHEN CSET                                              ; Display the conflict
            (IL:RESOLVE-IMPORT-CONFLICT PACKAGE CSET))
        (LET ((HASHTABLE (IF (%PACKAGE-EXTERNAL-ONLY PACKAGE)
                             (%PACKAGE-EXTERNAL-SYMBOLS PACKAGE)
                             (%PACKAGE-INTERNAL-SYMBOLS PACKAGE))))
             (DOLIST (SYM SYMS)
                 (IL:ADD-SYMBOL HASHTABLE SYM)
                 (IF (NULL (SYMBOL-PACKAGE SYM))
                     (SETF (SYMBOL-PACKAGE SYM)
                           PACKAGE))))
        (IF CSET
            (SHADOWING-IMPORT CSET PACKAGE)
            T)))


(DEFUN SHADOWING-IMPORT (IL:SYMBOLS &OPTIONAL (PACKAGE *PACKAGE*))
   "If a conflicting symbol is present, unintern it, otherwise just stick the symbol in."
   (DECLARE (SPECIAL *PACKAGE*))
   (SETF PACKAGE (IL:\\PACKAGIFY PACKAGE))
   (LET ((IL:HASHTABLE (IF (%PACKAGE-EXTERNAL-ONLY PACKAGE)
                           (%PACKAGE-EXTERNAL-SYMBOLS PACKAGE)
                           (%PACKAGE-INTERNAL-SYMBOLS PACKAGE))))
        (DOLIST (IL:SYM (IL:SYMBOL-LISTIFY IL:SYMBOLS))
            (MULTIPLE-VALUE-BIND (IL:S IL:W)
                (FIND-SYMBOL (SYMBOL-NAME IL:SYM)
                      PACKAGE)
              (UNLESS (AND IL:W (EQ IL:S IL:SYM))
                  (WHEN (OR (EQ IL:W :INTERNAL)
                            (EQ IL:W :EXTERNAL))                ; If it was shadowed, we don't want Unintern to fail
                      (SETF (%PACKAGE-SHADOWING-SYMBOLS PACKAGE)
                            (DELETE IL:S (%PACKAGE-SHADOWING-SYMBOLS PACKAGE)))
                      (UNINTERN IL:S PACKAGE))
                  (IL:ADD-SYMBOL IL:HASHTABLE IL:SYM))
              (PUSHNEW IL:SYM (%PACKAGE-SHADOWING-SYMBOLS PACKAGE)))))
   T)


(DEFUN SHADOW (IL:SYMBOLS &OPTIONAL (PACKAGE *PACKAGE*))
   "Hide the existing symbols with new ones in the package."
   (DECLARE (SPECIAL *PACKAGE*))
   (SETF PACKAGE (IL:\\PACKAGIFY PACKAGE))
   (LET ((IL:HASHTABLE (IF (%PACKAGE-EXTERNAL-ONLY PACKAGE)
                           (%PACKAGE-EXTERNAL-SYMBOLS PACKAGE)
                           (%PACKAGE-INTERNAL-SYMBOLS PACKAGE))))
        (DOLIST (IL:SYM (IL:SYMBOL-LISTIFY IL:SYMBOLS))
            (LET ((IL:NAME (SYMBOL-NAME IL:SYM)))
                 (MULTIPLE-VALUE-BIND (IL:S IL:W)
                     (FIND-SYMBOL IL:NAME PACKAGE)
```

```
                    (UNLESS (OR (EQ IL:W :INTERNAL)
                                (EQ IL:W :EXTERNAL))
                        (IL:SETQ IL:S (MAKE-SYMBOL IL:NAME))
                        (SETF (SYMBOL-PACKAGE IL:S)
                              PACKAGE)
                        (IL:ADD-SYMBOL IL:HASHTABLE IL:S)
                        (PUSHNEW IL:S (%PACKAGE-SHADOWING-SYMBOLS PACKAGE)))))))
   T)

(DEFUN UNUSE-PACKAGE (IL:PACKAGES-TO-UNUSE &OPTIONAL (PACKAGE *PACKAGE*))
   "Remove some packages from the use (inherit) list of another package."
   (DECLARE (SPECIAL *PACKAGE*))
   (SETF PACKAGE (IL:\\PACKAGIFY PACKAGE))
   (DOLIST (IL:P (IL:PACKAGE-LISTIFY IL:PACKAGES-TO-UNUSE))
       (SETF (%PACKAGE-USE-LIST PACKAGE)
             (IL:REMOVE IL:P (%PACKAGE-USE-LIST PACKAGE)))
       (SETF (%PACKAGE-TABLES PACKAGE)
             (IL:REMOVE (%PACKAGE-EXTERNAL-SYMBOLS IL:P)
                    (%PACKAGE-TABLES PACKAGE)))
       (SETF (%PACKAGE-USED-BY-LIST IL:P)
             (IL:REMOVE PACKAGE (%PACKAGE-USED-BY-LIST IL:P))))
   T)
```

;; Knowing about the package name space

```
(DEFUN LIST-ALL-PACKAGES ()
   "Return a list of the names of all existing packages."
   (DECLARE (SPECIAL IL:*PACKAGE-FROM-NAME*))
   (LET ((IL:RES NIL))
        (MAPHASH #'(LAMBDA (IL:K IL:V)
                       (PUSHNEW IL:V IL:RES))
                 IL:*PACKAGE-FROM-NAME*)
        IL:RES))
```

;; Putting symbols into packages

```
(DEFUN IL:ADD-SYMBOL (IL:TABLE SYMBOL)
   "Add a symbol to a package hashtable.  The symbol is assumed not to be present."
   (LET* ((IL:VEC (PACKAGE-HASHTABLE-TABLE IL:TABLE))
          (IL:HASH (PACKAGE-HASHTABLE-HASH IL:TABLE))
          (IL:LEN (ARRAY-TOTAL-SIZE (CAR IL:VEC)))
          (IL:SIZE (PACKAGE-HASHTABLE-SIZE IL:TABLE))
          (IL:SYMBOL-BASE (IL:|ffetch| (SYMBOL IL:PNAMEBASE) IL:|of| SYMBOL))
          (IL:SYMBOL-LENGTH (IL:|ffetch| (SYMBOL IL:PNAMELENGTH) IL:|of| SYMBOL))
          (IL:SYMBOL-FATP (IL:|ffetch| (SYMBOL IL:FATPNAMEP) IL:|of| SYMBOL))
          (SXHASH (IL:SYMBOL-HASH IL:SYMBOL-BASE 1 IL:SYMBOL-LENGTH IL:SYMBOL-FATP))
          (IL:H2 (IL:REHASH-FACTOR SXHASH IL:LEN)))
         (DECLARE (TYPE (SIMPLE-ARRAY (UNSIGNED-BYTE 32))
                        IL:VEC)
                  (TYPE (SIMPLE-ARRAY (UNSIGNED-BYTE 8))
                        IL:HASH))
         (COND
            ((<= (PACKAGE-HASHTABLE-FREE IL:TABLE)
                 (IL:LRSH IL:SIZE 2))
               ;; Let each hash table get at most 75% full, so we have a reasonable chance of makeing a clear hash miss in few reprobes.  Formerly,
               ;; there was a BIG performance hit after the initial table overflowed.
               (COND
                  ((>= IL:SIZE IL:HASHTABLE-SIZE-LIMIT)
                     ;; We've spilled over into needing the list-of-tables feature, so add to the list.
                     (IL:SETQ IL:VEC (IL:NCONC1 IL:VEC (MAKE-ARRAY IL:LEN :ELEMENT-TYPE '(UNSIGNED-BYTE 32))))
                     (IL:SETQ IL:HASH (IL:NCONC1 IL:HASH (MAKE-ARRAY IL:LEN :ELEMENT-TYPE '(UNSIGNED-BYTE 8))))
                     (SETF (PACKAGE-HASHTABLE-FREE IL:TABLE)
                           (IL:FIX (IL:FTIMES (PACKAGE-HASHTABLE-SIZE IL:TABLE)
                                              IL:PACKAGE-REHASH-THRESHOLD)))
                     (IL:ADD-SYMBOL IL:TABLE SYMBOL))
                  (T ;; The initial table is still smaller than the limit.  Increase its size.
                     (LET ((IL:SIZE (PACKAGE-HASHTABLE-SIZE IL:TABLE))
                           (IL:VEC1 (CAR IL:VEC))
                           (IL:HASH1 (CAR IL:HASH)))
                          (IL:MAKE-PACKAGE-HASHTABLE (IL:ITIMES IL:SIZE 2)
                                 IL:TABLE)
                          (IL:ADD-SYMBOL IL:TABLE SYMBOL)
                          (DOTIMES (IL:I IL:LEN)
                              (WHEN (IL:IGREATERP (AREF IL:HASH1 IL:I)
                                                  1)
                                    (IL:ADD-SYMBOL IL:TABLE (IL:\\INDEXATOMPNAME (AREF IL:VEC1 IL:I))))))))))
            (T (LET ((IL:THIS-HASH (CAR (IL:FLAST IL:HASH)))
                     (IL:THIS-VEC (CAR (IL:FLAST IL:VEC))))
                    (DO ((IL:I (IL:IREMAINDER SXHASH IL:LEN)
                               (IL:SYMBOL-HASH-REPROBE IL:I IL:H2 IL:LEN)))
```

```
                        ((IL:ILESSP (AREF IL:THIS-HASH IL:I)
                                    2)
                         (IF (EQL 0 (AREF IL:THIS-HASH IL:I))
                             (DECF (PACKAGE-HASHTABLE-FREE IL:TABLE))
                             (DECF (PACKAGE-HASHTABLE-DELETED IL:TABLE)))
                         (SETF (AREF IL:THIS-VEC IL:I)
                               (IL:\\ATOMPNAMEINDEX SYMBOL))
                         (SETF (AREF IL:THIS-HASH IL:I)
                               (IL:ENTRY-HASH IL:SYMBOL-LENGTH SXHASH)))))))))

(DEFMACRO IL:WITH-SYMBOL ((IL:INDEX-VAR IL:SYMBOL-VAR IL:TABLE IL:BASE IL:OFFSET IL:LENGTH IL:FATP SXHASH
                                IL:ENTRY-HASH IL:HASH-TABLE-TABLE IL:HASH-TABLE-HASH)
                          &BODY IL:FORMS)
   "Find where the symbol named String is stored in Table.  Index-Var is bound to the index, or NIL if it is not
   present.  Symbol-Var is bound to the symbol.  Length and Hash are the length and sxhash of String.
   Entry-Hash is the entry-hash of the string and length."
   (LET ((IL:VEC (OR IL:HASH-TABLE-TABLE (IL:GENSYM)))
         (IL:HASH (OR IL:HASH-TABLE-HASH (IL:GENSYM)))
         (IL:LEN (IL:GENSYM))
         (IL:H2 (IL:GENSYM))
         (IL:EHASH (IL:GENSYM))
         (IL:VECS (IL:GENSYM))
         (IL:HASHS (IL:GENSYM))
         (IL:LIMIT (IL:GENSYM)))
        `(LET* ((,IL:VECS (PACKAGE-HASHTABLE-TABLE ,IL:TABLE))
                (,IL:HASHS (PACKAGE-HASHTABLE-HASH ,IL:TABLE))
                (,IL:LEN (ARRAY-TOTAL-SIZE (CAR ,IL:VECS)))
                (,IL:H2 (IL:REHASH-FACTOR ,SXHASH ,IL:LEN))
                ,IL:VEC
                ,IL:HASH
                ,IL:LIMIT)
               (DECLARE (TYPE (SIMPLE-ARRAY (UNSIGNED-BYTE 8))
                              ,IL:HASH)
                        (TYPE (SIMPLE-ARRAY (UNSIGNED-BYTE 32))
                              ,IL:VEC))
               (PROG (,IL:INDEX-VAR ,IL:SYMBOL-VAR ,IL:EHASH)
                     ;; Loop thru all the hash tables looking for the symbol.
                     IL:OUTER-LOOP
                          (IL:SETQ ,IL:HASH (IL:POP ,IL:HASHS))          ; Hashvalues
                          (IL:SETQ ,IL:VEC (IL:POP ,IL:VECS))            ; The symbol vector
                          (IL:SETQ ,IL:INDEX-VAR (IL:IREMAINDER ,SXHASH ,IL:LEN))
                                                                         ; Starting probe.
                          (IL:SETQ ,IL:LIMIT ,IL:LEN)
                     LOOP

                     ;; Loop thru the entries in a single hash table.
                          (IL:SETQ ,IL:EHASH (AREF ,IL:HASH ,IL:INDEX-VAR))
                          (COND
                             ((EQL ,IL:EHASH ,IL:ENTRY-HASH)
                              ;; SIngle-byte hash matches; try the whole name.
                              (IL:SETQ ,IL:SYMBOL-VAR (IL:\\INDEXATOMPNAME (AREF ,IL:VEC ,IL:INDEX-VAR)))
                              (WHEN (IL:\\SYMBOL-EQUALBASE ,IL:SYMBOL-VAR ,IL:BASE ,IL:OFFSET ,IL:LENGTH ,IL:FATP)
                                    (GO IL:DOIT)))
                             ((EQL 0 ,IL:EHASH)                          ; Found an empty hash slot, so it's not in this table.
                              (COND
                                 ((NULL ,IL:HASHS)
                                  ;; we've run out of sub-tables to look in.  Give the we-couldn't-find-it signal.
                                  (IL:SETQ ,IL:INDEX-VAR NIL)
                                  (GO IL:DOIT))
                                 (T (GO IL:OUTER-LOOP))))
                             ((EQL 0 (IL:SETQ ,IL:LIMIT (IL:SUB1 ,IL:LIMIT)))
                                                                         ; We.ve been thru the whole table, so it's not in this table.
                              (COND
                                 ((NULL ,IL:HASHS)
                                  ;; we've run out of sub-tables to look in.  Give the we-couldn't-find-it signal.
                                  (IL:SETQ ,IL:INDEX-VAR NIL)
                                  (GO IL:DOIT))
                                 (T (GO IL:OUTER-LOOP)))))
                          (IL:SETQ ,IL:INDEX-VAR (IL:SYMBOL-HASH-REPROBE ,IL:INDEX-VAR ,IL:H2 ,IL:LEN))
                          (GO LOOP)
                     IL:DOIT
                          (RETURN (PROGN ,@IL:FORMS))))))

(DEFUN IL:INTERN* (IL:BASE IL:OFFSET IL:LENGTH IL:FATP IL:FATCHARSEENP PACKAGE IL:EXTERNALP)
   "If the symbol doesn't exist then create it, special-casing the keyword package."
   (DECLARE (SPECIAL IL:*KEYWORD-PACKAGE*))
   (MULTIPLE-VALUE-BIND (SYMBOL IL:WHERE)
       (IL:FIND-SYMBOL* IL:BASE IL:OFFSET IL:LENGTH IL:FATP PACKAGE)
     (IF IL:WHERE
         (VALUES SYMBOL IL:WHERE)
```

```
        (LET ((SYMBOL (IL:UNINTERRUPTABLY
                        (IL:\\CREATE.SYMBOL IL:BASE IL:OFFSET IL:LENGTH IL:FATP IL:FATCHARSEENP))))
             (SETF (SYMBOL-PACKAGE SYMBOL)
                   PACKAGE)
             (COND
                ((EQ PACKAGE IL:*KEYWORD-PACKAGE*)
                 (IL:ADD-SYMBOL (%PACKAGE-EXTERNAL-SYMBOLS IL:*KEYWORD-PACKAGE*)
                        SYMBOL)
                 (SET SYMBOL SYMBOL))
                ((OR IL:EXTERNALP (%PACKAGE-EXTERNAL-ONLY PACKAGE))
                 (IL:ADD-SYMBOL (%PACKAGE-EXTERNAL-SYMBOLS PACKAGE)
                        SYMBOL))
                (T (IL:ADD-SYMBOL (%PACKAGE-INTERNAL-SYMBOLS PACKAGE)
                          SYMBOL)))
             (VALUES SYMBOL NIL)))))
```

```
(DEFUN IL:FIND-SYMBOL* (IL:BASE IL:OFFSET IL:LENGTH IL:FATP PACKAGE)
                                                                ; Check internal and external symbols, then scan down the list of
                                                                ; hashtables for inherited symbols. When an inherited symbol is
                                                                ; found pull that table to the beginning of the list.
```

;; Find a symbol in the package given, if it eexists.

```
  (LET* ((IL:HASH (IL:SYMBOL-HASH IL:BASE IL:OFFSET IL:LENGTH IL:FATP))
         (IL:EHASH (IL:ENTRY-HASH IL:LENGTH IL:HASH))
         (IL:RESULT (IL:\\CREATECELL IL:\\FIXP))
         IL:SYM IL:WHERE (IL:DONE))
        (UNLESS (%PACKAGE-EXTERNAL-ONLY PACKAGE)
            (IL:NEW-SYMBOL-CODE (PROGN (IL:SETQ IL:SYM (IL:SUBRCALL IL:WITH-SYMBOL IL:BASE IL:OFFSET IL:LENGTH
                                                                    IL:FATP (%PACKAGE-INTERNAL-SYMBOLS PACKAGE)
                                                                    IL:RESULT))
                                       (COND
                                          ((NOT (IL:IEQP IL:RESULT -1))
                                           (IL:SETQ IL:WHERE :INTERNAL)
                                           (IL:SETQ IL:DONE T))))
                    (IL:WITH-SYMBOL (IL:FOUND SYMBOL (%PACKAGE-INTERNAL-SYMBOLS PACKAGE)
                                              IL:BASE IL:OFFSET IL:LENGTH IL:FATP IL:HASH IL:EHASH NIL NIL)
                           (WHEN IL:FOUND
                               ;; Was (cl:return-from find-symbol* (cl:values cl:symbol :internal))
                               (IL:SETQ IL:WHERE :INTERNAL)
                               (IL:SETQ IL:DONE T)))))
        (UNLESS IL:DONE
            (IL:NEW-SYMBOL-CODE (PROGN (IL:SETQ IL:SYM (IL:SUBRCALL IL:WITH-SYMBOL IL:BASE IL:OFFSET IL:LENGTH
                                                                    IL:FATP (%PACKAGE-EXTERNAL-SYMBOLS PACKAGE)
                                                                    IL:RESULT))
                                       (COND
                                          ((NOT (IL:IEQP IL:RESULT -1))
                                           (IL:SETQ IL:WHERE :EXTERNAL)
                                           (IL:SETQ IL:DONE T))))
                    (IL:WITH-SYMBOL (IL:FOUND SYMBOL (%PACKAGE-EXTERNAL-SYMBOLS PACKAGE)
                                              IL:BASE IL:OFFSET IL:LENGTH IL:FATP IL:HASH IL:EHASH NIL NIL)
                           (WHEN IL:FOUND
                               ;; Was (cl:return-from find-symbol* (cl:values cl:symbol :external))
                               (IL:SETQ IL:SYM SYMBOL)
                               (IL:SETQ IL:WHERE :EXTERNAL)
                               (IL:SETQ IL:DONE T)))))
        (UNLESS IL:DONE
            (LET ((IL:HEAD (%PACKAGE-TABLES PACKAGE)))
                 (DO ((IL:PREV IL:HEAD IL:TABLE)
                      (IL:TABLE (CDR IL:HEAD)
                                (CDR IL:TABLE)))
                     ((OR IL:DONE (NULL IL:TABLE))
                      (VALUES NIL NIL))
                   (IL:NEW-SYMBOL-CODE (PROGN (IL:SETQ IL:SYM (IL:SUBRCALL IL:WITH-SYMBOL IL:BASE IL:OFFSET
                                                                           IL:LENGTH IL:FATP (CAR IL:TABLE)
                                                                           IL:RESULT))
                                              (COND
                                                 ((NOT (IL:IEQP IL:RESULT -1))
                                                  (UNLESS (EQ IL:PREV IL:HEAD)
                                                      (SHIFTF (CDR IL:PREV)
                                                              (CDR IL:TABLE)
                                                              (CDR IL:HEAD)
                                                              IL:TABLE))
                                                  ;; Was (cl:return-from find-symbol* (cl:values cl:symbol :inherited))
                                                  (IL:SETQ IL:WHERE :INHERITED)
                                                  (IL:SETQ IL:DONE T))))
                                       (IL:WITH-SYMBOL (IL:FOUND SYMBOL (CAR IL:TABLE)
                                                                 IL:BASE IL:OFFSET IL:LENGTH IL:FATP IL:HASH IL:EHASH NIL NIL)
                                              (WHEN IL:FOUND
                                                  (UNLESS (EQ IL:PREV IL:HEAD)
                                                      (SHIFTF (CDR IL:PREV)
                                                              (CDR IL:TABLE)
                                                              (CDR IL:HEAD)
                                                              IL:TABLE))))))))))
```

```
                                        ;; Was (cl:return-from find-symbol* (cl:values cl:symbol :inherited))
                                        (IL:SETQ IL:SYM SYMBOL)
                                        (IL:SETQ IL:WHERE :INHERITED)
                                        (IL:SETQ IL:DONE T)))))))
        (VALUES IL:SYM IL:WHERE)))


(DEFUN INTERN (IL:NAME &OPTIONAL (PACKAGE *PACKAGE*))
   "Intern the name in the package, returning a symbol."
   (DECLARE (SPECIAL *PACKAGE*))
   (IL:SETQ IL:NAME (COND
                       ((IL:STRINGP IL:NAME)
                        IL:NAME)
                       ((STRINGP IL:NAME)
                        (IL:MKSTRING IL:NAME))
                       (T (IL:ERROR "Not a string " IL:NAME))))
   (COND
      ((NULL PACKAGE)                                          ; XCL extension, makes uninterned symbols
       (MAKE-SYMBOL IL:NAME))
      (T                                                       ; Package is at least non-null
         (IL:SETQ PACKAGE (IL:\\PACKAGIFY PACKAGE))
         (LET ((IL:BASE (IL:|ffetch| (IL:STRINGP IL:BASE) IL:|of| IL:NAME))
               (IL:OFFSET (IL:|ffetch| (IL:STRINGP IL:OFFST) IL:|of| IL:NAME))
               (IL:LENGTH (IL:|ffetch| (IL:STRINGP IL:LENGTH) IL:|of| IL:NAME))
               (IL:FATP (IL:|ffetch| (IL:STRINGP IL:FATSTRINGP) IL:|of| IL:NAME)))
              (IL:INTERN* IL:BASE IL:OFFSET IL:LENGTH IL:FATP (IL:\\FATCHARSEENP IL:BASE IL:OFFSET IL:LENGTH
                                                                                  IL:FATP)
                     PACKAGE NIL)))))


(DEFUN FIND-SYMBOL (IL:NAME &OPTIONAL (PACKAGE *PACKAGE*))
   "Find a symbol with the given name in a package."
   (DECLARE (SPECIAL *PACKAGE*))
   (IL:SETQ IL:NAME (IL:\\SIMPLE-STRINGIFY IL:NAME))
   (IL:SETQ PACKAGE (IL:\\PACKAGIFY PACKAGE))
   (IL:FIND-SYMBOL* (IL:|ffetch| (IL:STRINGP IL:BASE) IL:|of| IL:NAME)
          (IL:|ffetch| (IL:STRINGP IL:OFFST) IL:|of| IL:NAME)
          (IL:|ffetch| (IL:STRINGP IL:LENGTH) IL:|of| IL:NAME)
          (IL:|ffetch| (IL:STRINGP IL:FATSTRINGP) IL:|of| IL:NAME)
          PACKAGE))

;; Removing symbols from packages


(DEFUN IL:NUKE-SYMBOL (IL:TABLE STRING)
   "Mark a symbol in a package-hashtable deleted"
   (IL:SETQ STRING (IL:MKSTRING STRING))
   (LET* ((IL:BASE (IL:|ffetch| (IL:STRINGP IL:BASE) IL:|of| STRING))
          (IL:OFFSET (IL:|ffetch| (IL:STRINGP IL:OFFST) IL:|of| STRING))
          (IL:LENGTH (IL:|ffetch| (IL:STRINGP IL:LENGTH) IL:|of| STRING))
          (IL:FATP (IL:|ffetch| (IL:STRINGP IL:FATSTRINGP) IL:|of| STRING))
          (IL:HASH (IL:SYMBOL-HASH IL:BASE IL:OFFSET IL:LENGTH IL:FATP))
          (IL:EHASH (IL:ENTRY-HASH IL:LENGTH IL:HASH)))
         (IL:WITH-SYMBOL (IL:INDEX SYMBOL IL:TABLE IL:BASE IL:OFFSET IL:LENGTH IL:FATP IL:HASH IL:EHASH NIL
                                   IL:TABLE-HASH)
                (SETF (AREF IL:TABLE-HASH IL:INDEX)
                      1)
                (INCF (PACKAGE-HASHTABLE-DELETED IL:TABLE)))))


(DEFUN UNINTERN (SYMBOL &OPTIONAL (PACKAGE *PACKAGE*))
   "Remove a symbol from a package.  If uninterning a shadowing symbol, then a name conflict can result,
   otherwise just nuke the symbol."
   (DECLARE (SPECIAL *PACKAGE*))
   (IL:SETQ PACKAGE (IL:\\PACKAGIFY PACKAGE))
   (LET* ((IL:NAME (SYMBOL-NAME SYMBOL))
          (IL:SHADOWING-SYMBOLS (%PACKAGE-SHADOWING-SYMBOLS PACKAGE)))
         (DECLARE (TYPE LIST IL:SHADOWING-SYMBOLS)
                (SPECIAL *QUERY-IO*))
         (WHEN (IL:FMEMB SYMBOL IL:SHADOWING-SYMBOLS)
             (LET ((IL:CSET NIL))
                  ;; If a name conflict is revealed, give the user a chance to shadowing-import one of the available symbols.
                  (DOLIST (IL:P (%PACKAGE-USE-LIST PACKAGE))
                      (MULTIPLE-VALUE-BIND (IL:S IL:W)
                          (IL:FIND-EXTERNAL-SYMBOL IL:NAME IL:P)
                        (WHEN IL:W (PUSHNEW IL:S IL:CSET))))
                  (WHEN (CDR IL:CSET)                          ; If there is more than one, handle the conflict
                      (IL:RESOLVE-UNINTERN-CONFLICT SYMBOL IL:CSET PACKAGE)))
             (SETF (%PACKAGE-SHADOWING-SYMBOLS PACKAGE)
                   (DELETE SYMBOL IL:SHADOWING-SYMBOLS :TEST #'EQ)))
         (MULTIPLE-VALUE-BIND (IL:S IL:W)
             (FIND-SYMBOL IL:NAME PACKAGE)
           (COND
              ((AND (EQ IL:S SYMBOL)
                    (OR (EQ IL:W :INTERNAL)
```

```
                    (EQ IL:W :EXTERNAL)))
               (IL:NUKE-SYMBOL (IF (EQ IL:W :INTERNAL)
                                   (%PACKAGE-INTERNAL-SYMBOLS PACKAGE)
                                   (%PACKAGE-EXTERNAL-SYMBOLS PACKAGE))
                      IL:NAME)
               (IF (EQ (SYMBOL-PACKAGE SYMBOL)
                       PACKAGE)
                   (SETF (SYMBOL-PACKAGE SYMBOL)
                         NIL))
               T)
              (T NIL)))))

(DEFUN IL:MOBY-UNINTERN (SYMBOL PACKAGE)
   "Like Unintern, but if symbol is inherited chases down the package it is inherited from and uninterns it
   there.  Used for name-conflict resolution.  Shadowing symbols are not uninterned since they do not cause
   conflicts."
   (UNLESS (IL:FMEMB SYMBOL (%PACKAGE-SHADOWING-SYMBOLS PACKAGE))
       (OR (UNINTERN SYMBOL PACKAGE)
           (LET ((IL:NAME (SYMBOL-NAME SYMBOL)))
                (MULTIPLE-VALUE-BIND (IL:S IL:W)
                    (FIND-SYMBOL IL:NAME PACKAGE)
                  (WHEN (EQ IL:W :INHERITED)
                      (DOLIST (IL:Q (%PACKAGE-USE-LIST PACKAGE))
                          (MULTIPLE-VALUE-BIND (IL:U IL:X)
                              (IL:FIND-EXTERNAL-SYMBOL IL:NAME IL:Q)
                            (WHEN IL:X
                                (UNINTERN SYMBOL IL:Q)
                                (IL:RETFROM 'IL:MOBY-UNINTERN T)
                                ;; Was (cl:return-from moby-unintern t)

                                )))))))))

;; Iterations over package symbols

(DEFUN IL:\\INDEXATOMPNAME (IL:X)
   (IL:\\INDEXATOMPNAME IL:X))

;; Defined in EXPORTS.ALL and used by the DO-SYMBOLS macro

(IL:DECLARE\: IL:EVAL@COMPILE

(DEFUN IL:MAKE-DO-SYMBOLS-VARS ()
   `(,(IL:GENSYM)
     ,(IL:GENSYM)
     ,(IL:GENSYM)
     ,(IL:GENSYM)
     ,(IL:GENSYM)
     ,(IL:GENSYM)))

(DEFUN IL:MAKE-DO-SYMBOLS-CODE (IL:VARS IL:VAR HASH-TABLE IL:EXIT-FORM IL:FORMS)
   (LET ((IL:INDEX (FIRST IL:VARS))
         (IL:HASH-VECTOR (SECOND IL:VARS))
         (IL:HASH (THIRD IL:VARS))
         (IL:TERMINUS (FOURTH IL:VARS))
         (IL:HASH-VECTOR-LIST (FIFTH IL:VARS))
         (IL:TABLE-VECTOR-LIST (SIXTH IL:VARS))
         (IL:TOP (IL:GENSYM))
         (IL:REAL-TOP (IL:GENSYM)))
        `((IL:SETQ ,IL:TABLE-VECTOR-LIST (PACKAGE-HASHTABLE-TABLE ,HASH-TABLE))
          (IL:SETQ ,IL:HASH-VECTOR-LIST (PACKAGE-HASHTABLE-HASH ,HASH-TABLE))
          ,IL:REAL-TOP
          (IL:SETQ ,IL:INDEX 0)
          (IL:SETQ ,IL:HASH-VECTOR (IL:POP ,IL:TABLE-VECTOR-LIST))
          (IL:SETQ ,IL:HASH (IL:POP ,IL:HASH-VECTOR-LIST))
          (IL:SETQ ,IL:TERMINUS (ARRAY-TOTAL-SIZE (THE (SIMPLE-ARRAY (UNSIGNED-BYTE 32))
                                                       ,IL:HASH-VECTOR)))
          ,IL:TOP
          (IF (EQL ,IL:INDEX ,IL:TERMINUS)
              (IF (NULL ,IL:TABLE-VECTOR-LIST)
                  ,IL:EXIT-FORM
                  (GO ,IL:REAL-TOP)))
          (WHEN (IL:IGREATERP (AREF (THE (SIMPLE-ARRAY (UNSIGNED-BYTE 8))
                                         ,IL:HASH)
                                    ,IL:INDEX)
                              1)
              (IL:SETQ ,IL:VAR (IL:\\INDEXATOMPNAME (AREF ,IL:HASH-VECTOR ,IL:INDEX)))
              ,@IL:FORMS)
          (INCF ,IL:INDEX)
          (GO ,IL:TOP))))
)
```

```
(DEFMACRO DO-EXTERNAL-SYMBOLS ((IL:VAR &OPTIONAL (PACKAGE '*PACKAGE*)
                                        IL:RESULT-FORM)
                               &BODY
                               (IL:CODE IL:DECLS))
   "Do-External-Symbols (Var [Package [Result-Form]]) {Declaration}* {Tag | Statement}* Executes the Forms once
   for each external symbol in the given Package with Var bound to the current symbol."
   (LET ((IL:VARS (IL:MAKE-DO-SYMBOLS-VARS)))
        `(PROG (,IL:VAR ,@IL:VARS)
               ,@IL:DECLS
               ,@(IL:MAKE-DO-SYMBOLS-CODE IL:VARS IL:VAR `(%PACKAGE-EXTERNAL-SYMBOLS ,PACKAGE)
                        `(RETURN (PROGN (IL:SETQ ,IL:VAR NIL)
                                        ,IL:RESULT-FORM))
                        IL:CODE))))

(DEFMACRO XCL:DO-LOCAL-SYMBOLS ((IL:VAR &OPTIONAL (PACKAGE '*PACKAGE*)
                                         IL:RESULT-FORM)
                                &BODY
                                (IL:CODE IL:DECLS))
   "Do-Local-Symbols (Var [Package [Result-Form]]) {Declaration}* {Tag | Statement}* Executes the Forms at least
   once for each symbol actually in the given Package with Var bound to the current symbol."
   (LET* ((IL:DONE-INTERNAL (IL:GENSYM))
          (IL:DONE-EXTERNAL (IL:GENSYM))
          (IL:VARS (IL:MAKE-DO-SYMBOLS-VARS))
          (IL:N-PACKAGE (IL:GENSYM)))
         `(PROG* ((,IL:N-PACKAGE ,PACKAGE)
                  ,IL:VAR
                  ,@IL:VARS)
                 ,@IL:DECLS
                 (WHEN (%PACKAGE-EXTERNAL-ONLY ,PACKAGE)
                     (GO ,IL:DONE-INTERNAL))
                 ,@(IL:MAKE-DO-SYMBOLS-CODE IL:VARS IL:VAR `(%PACKAGE-INTERNAL-SYMBOLS ,PACKAGE)
                           `(GO ,IL:DONE-INTERNAL)
                           IL:CODE)
                 ,IL:DONE-INTERNAL
                 ,@(IL:MAKE-DO-SYMBOLS-CODE IL:VARS IL:VAR `(%PACKAGE-EXTERNAL-SYMBOLS ,PACKAGE)
                           `(GO ,IL:DONE-EXTERNAL)
                           IL:CODE)
                 ,IL:DONE-EXTERNAL
                 (IL:SETQ ,IL:VAR NIL)
                 (RETURN ,IL:RESULT-FORM))))

(DEFMACRO XCL:DO-INTERNAL-SYMBOLS ((IL:VAR &OPTIONAL (PACKAGE '*PACKAGE*)
                                           IL:RESULT-FORM)
                                   &BODY
                                   (IL:CODE IL:DECLS))
   "Do-Internal-Symbols (Var [Package [Result-Form]]) {Declaration}* {Tag | Statement}* Executes the Forms at
   least once for each symbol actually in the given Package and not exported with Var bound to the current
   symbol."
   (LET* ((IL:DONE-INTERNAL (IL:GENSYM))
          (IL:VARS (IL:MAKE-DO-SYMBOLS-VARS))
          (IL:N-PACKAGE (IL:GENSYM)))
         `(PROG* ((,IL:N-PACKAGE ,PACKAGE)
                  ,IL:VAR
                  ,@IL:VARS)
                 ,@IL:DECLS
                 (WHEN (%PACKAGE-EXTERNAL-ONLY ,PACKAGE)
                     (GO ,IL:DONE-INTERNAL))
                 ,@(IL:MAKE-DO-SYMBOLS-CODE IL:VARS IL:VAR `(%PACKAGE-INTERNAL-SYMBOLS ,PACKAGE)
                           `(GO ,IL:DONE-INTERNAL)
                           IL:CODE)
                 ,IL:DONE-INTERNAL
                 (IL:SETQ ,IL:VAR NIL)
                 (RETURN ,IL:RESULT-FORM))))

(DEFMACRO DO-SYMBOLS ((IL:VAR &OPTIONAL (PACKAGE '*PACKAGE*)
                              IL:RESULT-FORM)
                      &BODY
                      (IL:CODE IL:DECLS))
   "Do-Symbols (Var [Package [Result-Form]]) {Declaration}* {Tag | Statement}* Executes the Forms at least once
   for each symbol accessible in the given Package with Var bound to the current symbol."
   (LET* ((IL:DONE-INTERNAL (IL:GENSYM))
          (IL:DONE-EXTERNAL (IL:GENSYM))
          (IL:NEXT-INHERIT (IL:GENSYM))
          (IL:VARS (IL:MAKE-DO-SYMBOLS-VARS))
          (IL:N-PACKAGE (IL:GENSYM))
          (IL:SHADOWED (IL:GENSYM))
          (IL:INHERITS (IL:GENSYM))
          (IL:THIS-INHERIT (IL:GENSYM)))
         `(PROG* ((,IL:N-PACKAGE ,PACKAGE)
                  (,IL:SHADOWED (%PACKAGE-SHADOWING-SYMBOLS ,IL:N-PACKAGE))
                  (,IL:INHERITS (CDR (%PACKAGE-TABLES ,IL:N-PACKAGE)))
                  ,IL:VAR
                  ,@IL:VARS
                  ,IL:THIS-INHERIT)
```

```
                ,@IL:DECLS
                (WHEN (%PACKAGE-EXTERNAL-ONLY ,PACKAGE)
                    (GO ,IL:DONE-INTERNAL))
                ,@(IL:MAKE-DO-SYMBOLS-CODE IL:VARS IL:VAR `(%PACKAGE-INTERNAL-SYMBOLS ,PACKAGE)
                          `(GO ,IL:DONE-INTERNAL)
                          IL:CODE)
                ,IL:DONE-INTERNAL
                ,@(IL:MAKE-DO-SYMBOLS-CODE IL:VARS IL:VAR `(%PACKAGE-EXTERNAL-SYMBOLS ,PACKAGE)
                          `(GO ,IL:DONE-EXTERNAL)
                          IL:CODE)
                ,IL:DONE-EXTERNAL
                ,IL:NEXT-INHERIT
                (WHEN (NULL ,IL:INHERITS)
                    (IL:SETQ ,IL:VAR NIL)
                    (RETURN ,IL:RESULT-FORM))
                (IL:SETQ ,IL:THIS-INHERIT (CAR ,IL:INHERITS))
                ,@(IL:MAKE-DO-SYMBOLS-CODE IL:VARS IL:VAR IL:THIS-INHERIT `(PROGN (IL:SETQ ,IL:INHERITS
                                                                                    (CDR ,IL:INHERITS))
                                                                                   (GO ,IL:NEXT-INHERIT))
                          `((WHEN (OR (NOT ,IL:SHADOWED)
                                      (EQ (FIND-SYMBOL (SYMBOL-NAME ,IL:VAR)
                                                       ,IL:N-PACKAGE)
                                          ,IL:VAR))
                                  ,@IL:CODE))))))

(DEFMACRO DO-ALL-SYMBOLS ((IL:VAR &OPTIONAL IL:RESULT-FORM)
                          &BODY
                          (IL:CODE IL:DECLS))
   "Do-All-Symbols (Var [Package [Result-Form]]) {Declaration}* {Tag | Statement}* Executes the Forms once for
   each symbol in each package with Var bound to the current symbol."
   (LET* ((IL:PACKAGE-LOOP (IL:GENSYM))
          (IL:TAG (IL:GENSYM))
          (IL:PACKAGE-LIST (IL:GENSYM))
          (IL:VARS (IL:MAKE-DO-SYMBOLS-VARS))
          (IL:INTERNAL-CODE (IL:MAKE-DO-SYMBOLS-CODE IL:VARS IL:VAR `(%PACKAGE-INTERNAL-SYMBOLS
                                                                      (CAR ,IL:PACKAGE-LIST))
                                                     `(GO ,IL:TAG)
                                                     IL:CODE))
          (IL:EXTERNAL-CODE (IL:MAKE-DO-SYMBOLS-CODE IL:VARS IL:VAR `(%PACKAGE-EXTERNAL-SYMBOLS
                                                                      (CAR ,IL:PACKAGE-LIST))
                                                     `(PROGN (IL:SETQ ,IL:PACKAGE-LIST (CDR ,IL:PACKAGE-LIST))
                                                             (GO ,IL:PACKAGE-LOOP))
                                                     IL:CODE)))
         `(PROG (,IL:PACKAGE-LIST ,IL:VAR ,@IL:VARS)
                ,@IL:DECLS
                (IL:SETQ ,IL:PACKAGE-LIST (LIST-ALL-PACKAGES))
                ,IL:PACKAGE-LOOP
                (WHEN (NULL ,IL:PACKAGE-LIST)
                    (IL:SETQ ,IL:VAR NIL)
                    (RETURN ,IL:RESULT-FORM))
                (WHEN (%PACKAGE-EXTERNAL-ONLY (CAR ,IL:PACKAGE-LIST))
                    (GO ,IL:TAG))
                ,@IL:INTERNAL-CODE
                ,IL:TAG
                ,@IL:EXTERNAL-CODE)))
```

;; Finding symbols in a package or packages

```
(DEFUN FIND-ALL-SYMBOLS (IL:STRING-OR-SYMBOL)
   "Find every symbol in all packages with the given name."
   (LET ((STRING (IL:MKSTRING IL:STRING-OR-SYMBOL))
         (IL:RES NIL))
        (DECLARE (SPECIAL IL:*PACKAGE-FROM-NAME*))
        (MAPHASH #'(LAMBDA (IL:K IL:V)
                           (MULTIPLE-VALUE-BIND (IL:S IL:W)
                               (FIND-SYMBOL STRING IL:V)
                             (WHEN IL:W (PUSHNEW IL:S IL:RES))))
                 IL:*PACKAGE-FROM-NAME*)
        IL:RES))

(DEFUN IL:BRIEFLY-DESCRIBE-SYMBOL (SYMBOL)
   "Short form description of a symbol."
   (FRESH-LINE)
   (PRIN1 SYMBOL)
   (WHEN (BOUNDP SYMBOL)
       (WRITE-STRING ", value: ")
       (PRIN1 (SYMBOL-VALUE SYMBOL)))
   (IF (FBOUNDP SYMBOL)
       (WRITE-STRING " (defined)")))

(DEFUN APROPOS (STRING &OPTIONAL PACKAGE IL:EXTERNAL-ONLY)
   "Find all symbols matching the string pattern in the given (or current) package.  The search can be limited
   to external symbols only.  Prints a short description of each found symbols."
```

```
   (IL:SETQ STRING (IL:COPY-STRING (IL:\\SIMPLE-STRINGIFY STRING)))
   (LET ((IL:BASE (IL:|ffetch| (IL:STRINGP IL:BASE) IL:|of| STRING))
         (IL:OFFSET (IL:|ffetch| (IL:STRINGP IL:OFFST) IL:|of| STRING))
         (IL:LENGTH (IL:|ffetch| (IL:STRINGP IL:LENGTH) IL:|of| STRING))
         (IL:FATP (IL:|ffetch| (IL:STRINGP IL:FATSTRINGP) IL:|of| STRING)))
        (IL:\\UPCASEBASE IL:BASE IL:OFFSET IL:LENGTH IL:FATP)
        (IF (NULL PACKAGE)
            (DO-ALL-SYMBOLS (SYMBOL)
                   (IF (IL:APROPOS-SEARCH SYMBOL IL:BASE IL:OFFSET IL:LENGTH IL:FATP)
                       (IL:BRIEFLY-DESCRIBE-SYMBOL SYMBOL)))
            (LET ((PACKAGE (IL:\\PACKAGIFY PACKAGE)))
                 (IF IL:EXTERNAL-ONLY
                     (DO-EXTERNAL-SYMBOLS (SYMBOL PACKAGE)
                            (IF (IL:APROPOS-SEARCH SYMBOL IL:BASE IL:OFFSET IL:LENGTH IL:FATP)
                                (IL:BRIEFLY-DESCRIBE-SYMBOL SYMBOL)))
                     (DO-SYMBOLS (SYMBOL PACKAGE)
                            (IF (IL:APROPOS-SEARCH SYMBOL IL:BASE IL:OFFSET IL:LENGTH IL:FATP)
                                (IL:BRIEFLY-DESCRIBE-SYMBOL SYMBOL)))))))
   (VALUES))


(DEFUN APROPOS-LIST (STRING &OPTIONAL PACKAGE IL:EXTERNAL-ONLY)
   "Find all symbols matching the string pattern in the given (or current) package.  The search can be limited
   to external symbols only.  Returns a list of the matching symbols."
   (LET ((STRING (IL:COPY-STRING (IL:\\SIMPLE-STRINGIFY (IL:MKSTRING STRING))))
         (LIST 'NIL))
        (LET ((IL:BASE (IL:|ffetch| (IL:STRINGP IL:BASE) IL:|of| STRING))
              (IL:OFFSET (IL:|ffetch| (IL:STRINGP IL:OFFST) IL:|of| STRING))
              (IL:LENGTH (IL:|ffetch| (IL:STRINGP IL:LENGTH) IL:|of| STRING))
              (IL:FATP (IL:|ffetch| (IL:STRINGP IL:FATSTRINGP) IL:|of| STRING)))
             (IL:\\UPCASEBASE IL:BASE IL:OFFSET IL:LENGTH IL:FATP)
             (IF (NULL PACKAGE)
                 (DO-ALL-SYMBOLS (SYMBOL)
                        (IF (IL:APROPOS-SEARCH SYMBOL IL:BASE IL:OFFSET IL:LENGTH IL:FATP)
                            (PUSH SYMBOL LIST)))
                 (LET ((PACKAGE (IL:\\PACKAGIFY PACKAGE)))
                      (IF IL:EXTERNAL-ONLY
                          (DO-EXTERNAL-SYMBOLS (SYMBOL PACKAGE)
                                 (IF (IL:APROPOS-SEARCH SYMBOL IL:BASE IL:OFFSET IL:LENGTH IL:FATP)
                                     (PUSH SYMBOL LIST)))
                          (DO-SYMBOLS (SYMBOL PACKAGE)
                                 (IF (IL:APROPOS-SEARCH SYMBOL IL:BASE IL:OFFSET IL:LENGTH IL:FATP)
                                     (PUSH SYMBOL LIST)))))))
        LIST))
```

;; Reader and printer's interface to packages (plus *PACKAGE-FROM-INDEX* above)

```
(DEFUN IL:FIND-EXTERNAL-SYMBOL (STRING PACKAGE)
   (IL:SETQ STRING (IL:MKSTRING STRING))                          ; Convert symbols to strings (for the reader)
   (LET* ((IL:BASE (IL:|ffetch| (IL:STRINGP IL:BASE) IL:|of| STRING))
          (IL:OFFSET (IL:|ffetch| (IL:STRINGP IL:OFFST) IL:|of| STRING))
          (IL:LENGTH (IL:|ffetch| (IL:STRINGP IL:LENGTH) IL:|of| STRING))
          (IL:FATP (IL:|ffetch| (IL:STRINGP IL:FATSTRINGP) IL:|of| STRING))
          (IL:HASH (IL:SYMBOL-HASH IL:BASE IL:OFFSET IL:LENGTH IL:FATP))
          (IL:EHASH (IL:ENTRY-HASH IL:LENGTH IL:HASH))
          (IL:RESULT (IL:\\CREATECELL IL:\\FIXP))
          IL:SYM)
         (IL:NEW-SYMBOL-CODE (PROGN (IL:SETQ IL:SYM (IL:SUBRCALL IL:WITH-SYMBOL IL:BASE IL:OFFSET IL:LENGTH
                                                              IL:FATP (%PACKAGE-EXTERNAL-SYMBOLS PACKAGE)
                                                              IL:RESULT))
                                    (VALUES IL:SYM (NOT (IL:IEQP IL:RESULT -1))))
                (IL:WITH-SYMBOL (IL:FOUND SYMBOL (%PACKAGE-EXTERNAL-SYMBOLS PACKAGE)
                                      IL:BASE IL:OFFSET IL:LENGTH IL:FATP IL:HASH IL:EHASH NIL NIL)
                      (VALUES SYMBOL IL:FOUND)))))


(DEFUN IL:FIND-EXACT-SYMBOL (SYMBOL PACKAGE)
   "True if name of SYMBOL when looked up in PACKAGE is found and is exactly SYMBOL"
   (MULTIPLE-VALUE-BIND (IL:FOUNDSYM IL:WHERE)
       (IL:FIND-SYMBOL* (IL:|ffetch| (SYMBOL IL:PNAMEBASE) IL:|of| SYMBOL)
              1
              (IL:|ffetch| (SYMBOL IL:PNAMELENGTH) IL:|of| SYMBOL)
              (IL:|ffetch| (SYMBOL IL:FATPNAMEP) IL:|of| SYMBOL)
              PACKAGE)
     (AND IL:WHERE (EQ IL:FOUNDSYM SYMBOL))))


(DEFUN IL:PACKAGE-NAME-AS-SYMBOL (PACKAGE)
   (%PACKAGE-NAMESYMBOL PACKAGE))


(DEFUN IL:\\FIND.PACKAGE.INTERNAL (IL:BASE IL:OFFSET IL:LEN IL:FATP)
   (FIND-PACKAGE (IL:\\GETBASESTRING IL:BASE IL:OFFSET IL:LEN IL:FATP)))
```

;; Proper compiler, readtable and package environment

```
(IL:PUTPROPS IL:LLPACKAGE IL:FILETYPE COMPILE-FILE)

(IL:PUTPROPS IL:LLPACKAGE IL:MAKEFILE-ENVIRONMENT (:READTABLE "XCL" :PACKAGE "LISP"))

(IL:DECLARE\: IL:DONTEVAL@LOAD IL:DOEVAL@COMPILE IL:DONTCOPY IL:COMPILERVARS

(IL:ADDTOVAR IL:NLAMA XCL:DEFPACKAGE)

(IL:ADDTOVAR IL:NLAML )

(IL:ADDTOVAR IL:LAMA )
)

(IL:PUTPROPS IL:LLPACKAGE IL:COPYRIGHT ("Venue & Xerox Corporation" 1986 1987 1990 1991 1992))
```

FUNCTION INDEX

MACRO INDEX

VARIABLE INDEX

CONSTANT INDEX

STRUCTURE INDEX

PROPERTY INDEX