```
  File created:     14-Sep-89 10:01:13  {DSK}/python2/aria/migration/interlisp/IL-STARTUP.;2

   changes to:      (IL:FUNCTIONS NOTE-EXPORTED-SYMBOL CONVERT)

 previous date:      7-Jul-89 16:55:06  {DSK}/users/eweaver/convert/IL-STARTUP.;17

   Read Table:      XCL

      Package:      IL-CONVERT

         Format:      XCCS
```

```
(IL:RPAQQ IL:IL-STARTUPCOMS
          (

;;; This should be loaded before any other files.

            (EVAL-WHEN (LOAD COMPILE EVAL)
                   (IL:VARIABLES *IL-PACKAGE*))
            (IL:VARIABLES *IL-SIM-PACKAGE*)


;;; This funny stuff is for printing backquote forms.

            (IL:STRUCTURES BQ MACRO-ARG)


;;;

            (IL:VARIABLES *CURRENT-CONVERT-FORM* *CURRENT-CONVERT-FUNCTION* *GLOBALS* *LOCALS* *FUNCTION-CALLS*
                   *CURRENT-FUNCTION-CALLS* *CURRENT-FREE-REFERENCES* *EXPORTED-IL-SYMBOLS*)
            (IL:P (EXPORT 'CONVERT))
            (IL:FUNCTIONS CONVERT MAPCONVERT EXTERN NOTE-EXPORTED-SYMBOL)
            (IL:FUNCTIONS TRUE-LIST-P)
                                                                          ; true if this is nil or a true list

            ;; make a true list out of a pseudo-list  (make-true-list '(A B . C)) => (A B C)

            (IL:FUNCTIONS MAKE-TRUE-LIST)
            (IL:PROP (IL:MAKEFILE-ENVIRONMENT IL:FILETYPE)
                   IL:IL-STARTUP)))


;;; This should be loaded before any other files.

(EVAL-WHEN (LOAD COMPILE EVAL)


(DEFVAR *IL-PACKAGE* (FIND-PACKAGE "INTERLISP"))
)


(DEFVAR *IL-SIM-PACKAGE* (MAKE-PACKAGE "IL-SIM" :USE NIL))


;;; This funny stuff is for printing backquote forms.


(DEFSTRUCT (BQ (:TYPE LIST)
               (:CONSTRUCTOR MAKE-BQ (ELEMENT)))
   (BQFLAG 'IL:BQUOTE)
   ELEMENT)


(DEFSTRUCT (MACRO-ARG (:TYPE LIST)
                      (:CONSTRUCTOR MAKE-MACRO-ARG (&KEY ELEMENT APPEND-P (FLAG (IF APPEND-P
                                                                               'IL:\\\,@
                                                                               'IL:\\\,)))))
   FLAG
   ELEMENT)


;;;


(DEFVAR *CURRENT-CONVERT-FORM*)

(DEFVAR *CURRENT-CONVERT-FUNCTION*)

(DEFVAR *GLOBALS* NIL)

(DEFVAR *LOCALS* NIL)
```

```
(DEFVAR *FUNCTION-CALLS* NIL)

(DEFVAR *CURRENT-FUNCTION-CALLS* NIL)

(DEFVAR *CURRENT-FREE-REFERENCES* NIL)

(DEFVAR *EXPORTED-IL-SYMBOLS* NIL)

(EXPORT 'CONVERT)

(DEFUN CONVERT (FORM &AUX FN VAR)
   (IL:BLOCK)
   (LET ((*CURRENT-EXPRESSION* FORM))
        (COND
           ;; Forms in which the car is a symbol...

           ((AND (CONSP FORM)
                 (ATOM (FIRST FORM)))
            (COND
               ((NOT (TRUE-LIST-P FORM))
                (LET ((TAIL (CDR (LAST FORM))))

                     ;; dotted lists ending in a macro arg are okay.

                     (IF (AND (SYMBOLP TAIL)
                              (EQ (CDR (ASSOC TAIL *LOCALS*))
                                  :MACRO-ARG))
                         (LET ((MARG (MAKE-MACRO-ARG :ELEMENT TAIL))
                               (VAL (COPY-LIST FORM)))
                              (SETF (CDR (LAST VAL))
                                    MARG)
                              VAL)
                         (PROGN (WARN "~s not a list, left as is" FORM)
                                FORM))))
               ((LET ((FOO (GET (CAR FORM)
                                'IL:CLISPWORD)))
                     (AND (CONSP FOO)
                          (EQ (CAR FOO)
                              'IL:FORWORD)
                          (NOT (EQ (CAR FORM)
                                   'DECLARE))))
                (CONVERT-ITERATION-STATEMENT (CAR FORM)
                       (CDR FORM)))
               ((SETQ FN (GET (FIRST FORM)
                              'CONVERT-FORM))
                (SETQ *CURRENT-CONVERT-FORM* FORM *CURRENT-CONVERT-FUNCTION* FN)
                (APPLY FN (REST FORM)))
               ((OR (MACRO-FUNCTION (FIRST FORM))
                    (SPECIAL-FORM-P (FIRST FORM)))

                ;; Use CL code walker for this

                (WALK-FORM-INTERNAL FORM))
               ((EQ (CHAR (STRING (FIRST FORM))
                          0)
                    #\\)
                (WARN "Untranslatable function ~a" (STRING (FIRST FORM)))
                FORM)
               (T ;; (setq fn (first form) (extern (symbol-name (first form)) *il-package*))

                  (WHEN *CURRENT-FUNCTION-CALLS* (PUSHNEW FN *CURRENT-FUNCTION-CALLS*))
                  (NOTE-EXPORTED-SYMBOL (FIRST FORM))
                  (CONS (FIRST FORM)
                        (MAPCAR 'CONVERT (REST FORM))))))

           ;; Forms in which the car is a Lambda...

           ((AND (CONSP FORM)

                 ;; But car is cons

                 (SYMBOLP (CAAR FORM))
                 (STRING-EQUAL (CAAR FORM)
                        "LAMBDA"))
            (CONS (CONVERT (CAR FORM))
                  (MAPCONVERT (CDR FORM))))

           ;; Other non-atomic forms...

           ((CONSP FORM)
            (WARN "Unknown kind of form ~s, not converted." FORM)
            FORM)

           ;; Atomic forms...

           ((NULL FORM)
            NIL)
           ((EQ FORM T)
            T)
           ((KEYWORDP FORM)
            FORM)
           ((SYMBOLP FORM)
            (IF (SETQ VAR (ASSOC FORM *LOCALS*))
                (CASE (CDR VAR)
                    (:LOCAL (CAR VAR))
                    (:MACRO-ARG (MAKE-MACRO-ARG :ELEMENT (CAR VAR)))
                    (T (ERROR "unexpected value ~s in *LOCALS*" VAR)))
                (PROGN (NOTE-EXPORTED-SYMBOL FORM)
                       (WHEN *CURRENT-FREE-REFERENCES* (PUSHNEW FORM *CURRENT-FREE-REFERENCES*))
                       FORM)))
           (T FORM))))


(DEFUN MAPCONVERT (FORM-OR-FORMS)
   (IF (ATOM FORM-OR-FORMS)
       (CONVERT FORM-OR-FORMS)
       (DO* ((TAIL FORM-OR-FORMS (CDR TAIL))
             (SUBFORM (IF (CONSP TAIL)
                          (CAR TAIL)
                          TAIL)
                    (IF (CONSP TAIL)
                        (CAR TAIL)
                        TAIL))
             RESULT)
            ((ATOM TAIL)
             (IF (NULL TAIL)
                 (NREVERSE RESULT)
                 (PROGN (SETF (CDR (LAST (SETQ RESULT (NREVERSE RESULT))))
                              (CONVERT TAIL))
                        RESULT)))
          (PUSH (CONVERT SUBFORM)
                RESULT))))


(DEFUN EXTERN (STRING &OPTIONAL (PACKAGE *PACKAGE*))
                                                                       (IL:* (LET ((SYM (INTERN STRING PACKAGE)))
                                                                       (EXPORT SYM PACKAGE) (IF (EQ PACKAGE *IL-PACKAGE*)
                                                                       (PUSHNEW SYM *EXPORTED-IL-SYMBOLS*)) SYM))
   (ERROR "Old leftover call to EXTERN!"))


(DEFUN NOTE-EXPORTED-SYMBOL (SYM &AUX PKG PKGNM)
   ""
   (WHEN (NULL (SETQ PKG (SYMBOL-PACKAGE SYM)))
         (RETURN-FROM NOTE-EXPORTED-SYMBOL SYM))
   (WHEN (AND (EQ PKG IL:*INTERLISP-PACKAGE*)
              (NOT (EQ (FIND-SYMBOL (SYMBOL-NAME SYM)
                              IL:*LISP-PACKAGE*)
                       SYM))
              (OR *WARN-FOR-ALL-IL-SYMBOLS* (< (IL:\\LOLOC SYM)
                                               (IL:\\LOLOC *WARN-FOR-IL-SYMBOLS-LOWER-THAN-THIS*))))
       (LET ((*CURRENT-EXPRESSION* SYM))
            (WARN "Use of IL symbol ~a" SYM)))
   (WHEN (OR (EQ PKG IL:*INTERLISP-PACKAGE*)
             (AND (NOT (OR (EQ PKG IL:*KEYWORD-PACKAGE*)
                           (EQ PKG IL:*LISP-PACKAGE*)))
                  (MULTIPLE-VALUE-BIND (IGNORE TYPE)
                      (FIND-SYMBOL (SYMBOL-NAME SYM)
                             PKG)
                    (EQ TYPE :EXTERNAL))))
       (IF (NULL *FILE-CONTEXT*)
           (PUSHNEW SYM *EXPORTED-IL-SYMBOLS*)
           (PUSHNEW SYM (FILE-CONTEXT-EXPORTED-SYMS *FILE-CONTEXT*))))
   SYM)


(DEFUN TRUE-LIST-P (PSEUDO-LIST)
   (DO ((PL PSEUDO-LIST (CDR PL)))
       ((NULL PL)
        T)
     (IF (ATOM PL)
         (RETURN NIL))))

;; true if this is nil or a true list
;; make a true list out of a pseudo-list  (make-true-list '(A B . C)) => (A B C)


(DEFUN MAKE-TRUE-LIST (PSEUDO-LIST)
   (COND
      ((TRUE-LIST-P PSEUDO-LIST)
       PSEUDO-LIST)
      (T (DO ((TRUE-LIST NIL))
             ((ATOM PSEUDO-LIST)
              (NREVERSE (CONS PSEUDO-LIST TRUE-LIST)))
           (IF (ENDP PSEUDO-LIST)
                (RETURN (NREVERSE TRUE-LIST)))
            (PUSH (POP PSEUDO-LIST)
                  TRUE-LIST)))))

(IL:PUTPROPS IL:IL-STARTUP IL:MAKEFILE-ENVIRONMENT (:READTABLE "XCL" :PACKAGE (LET ((*PACKAGE* *PACKAGE*))
                                                                                   (IN-PACKAGE "IL-CONVERT")
                                                                                   *PACKAGE*)
                                                          :BASE 10))

(IL:PUTPROPS IL:IL-STARTUP IL:FILETYPE :COMPILE-FILE)

(IL:PUTPROPS IL:IL-STARTUP IL:COPYRIGHT ("ENVOS Corporation" 1989))
```

FUNCTION INDEX

VARIABLE INDEX

STRUCTURE INDEX

PROPERTY INDEX