File created: 26-Jan-90 10:28:55 **{DSK}/users/welch/migration/IL-CONVERT.;5**

changes to: (IL:VARS IL:IL-CONVERTCOMS)

previous date: 25-Jan-90 14:45:43 {DSK}/users/welch/migration/IL-CONVERT.;4

Read Table: XCL

Package: IL-CONVERT

Format: XCCS

```
(IL:RPAQQ IL:IL-CONVERTCOMS
          ((IL:FUNCTIONS IL-DEFCONV)

           ;; Used when an Interlisp function is the same as the Common Lisp function of the same name.

           (IL:FUNCTIONS IL-COPYDEF)

           ;; Used to define a run-time function (not a converter function).

           (IL:FUNCTIONS IL-DEFUN IL-DEFVAR)

           ;;
           ;; ; Creates an external symbol in the IL package.
           ;; (defmacro il-defsym (name)
           ;;     `(export (intern (symbol-name ',name) *il-package*) *il-package*))

           ;; (defmacro il-import (symbol)
           ;;   `(progn (import ,symbol 'il)
           ;;            (export (find-symbol (symbol-name ,symbol) 'il) 'il)))


           (IL:FUNCTIONS IL-COPYCONV)

           ;; Defines a "Non-conversion" form for use with things like \GETBASE.

           (IL:FUNCTIONS IL-WARNINGFORM)

           ;; Defines a function (e.g. PROGN-IF-NEEDED) that takes a list and sticks a PROGN (or whatever) at the beginning if the length is not 1.
           ;; Used to eliminate ugly redundant PROGNs. If the length is 0, returns whatever the form itself returns when given no arguments (e.g. T
           ;; for AND, NIL for OR).

           (IL:P (MACROLET ((DEF-*-IF-NEEDED
                             (NAME)
                             (LET ((NAME-STRING (SYMBOL-NAME NAME)))
                                  `(DEFUN ,(INTERN (CONCATENATE 'STRING NAME-STRING "-IF-NEEDED"))
                                          (ARGS)
                                          (CASE (LENGTH ARGS)
                                              (0 ,(EVAL `(,NAME)))
                                              (1 (FIRST ARGS))
                                              (T `(,',NAME ,@ARGS)))))))
                        (DEF-*-IF-NEEDED PROGN)
                        (DEF-*-IF-NEEDED AND)
                        (DEF-*-IF-NEEDED OR)))
           (IL:STRUCTURES FAKE-SYMBOL SHARP-DOT SHARP-COMMA)

           ;; Aux function to see whether or not to generate a symbolp check

           (IL:FUNCTIONS QUOTED-SYMBOL-P)
           (IL:VARIABLES *ORIGINAL-READTABLE*)
           (IL:FUNCTIONS OLD-CONVERT-FILE)
           (IL:P (EXPORT 'CONVERT-FILE))

           ;; (convert-file "~/medley/ADISPLAY" "adisplay") (convert-file "foo1" "foo2") (convert-file "foo3" "foo4")

           (IL:P (EXPORT '(READ-EXPORTS WRITE-EXPORTS READ-RECORD-TYPES WRITE-RECORD-TYPES)))
           (IL:FUNCTIONS READ-EXPORTS)
                                                                          ; Get the symbol list
           (IL:FUNCTIONS WRITE-EXPORTS READ-RECORD-TYPES WRITE-RECORD-TYPES READ-HASH-TABLE WRITE-HASH-TABLE)
           (IL:FUNCTIONS CONVERT-FILE CONVERT-FILECOMS CONVERT-ONE-FILECOM EXPURGATE-EXTRANEOUS-PROGNS
                  REORDER-FILECOMS MAKE-EXPORT-FORM)
           (IL:VARIABLES *WALKER-TEMPLATES*)
           (IL:FUNCTIONS GET-WALKER-TEMPLATE WALK-FORM-INTERNAL WALK-TEMPLATE WALK-TEMPLATE-HANDLE-REPEAT
                  WALK-TEMPLATE-HANDLE-REPEAT-1 WALK-REPEAT-EVAL RECONS RELIST RELIST* RELIST-INTERNAL)
           (IL:VARIABLES *GETVALUE-TRANSLATION* *CURRENT-DEFINITION* *CURRENT-DEFINITION-TYPE*
                  *CURRENT-EXPRESSION* *CURRENT-LOCALS* *FILE-CONTEXT* *WALKER-FIND-PARAMETER-LIST*
                  *WARNINGS-MADE* *PACKAGE-FOR-IL-SYMBOLS* *PACKAGE-FOR-RESULT-FILE*
                  *PARAMETERS-ALWAYS-OPTIONAL* *PROMPT-FOR-UNKNOWN-MACRO-TEMPLATE* *UNKNOWN-MACRO-ACTION*
                  *ALWAYS-INCLUDE-PROPS*)
           (IL:DECLARE\: IL:DONTCOPY (IL:PROP (IL:MAKEFILE-ENVIRONMENT IL:FILETYPE)
                                              IL:IL-CONVERT))))

(XCL:DEFDEFINER IL-DEFCONV IL:FUNCTIONS (NAME ARGLIST &REST REST)
   (CHECK-TYPE NAME SYMBOL)
   (LET ((FN-NAME (FIND-SYMBOL (SYMBOL-NAME NAME)
                         *IL-PACKAGE*)))
        (IF FN-NAME
            `(SETF (GET ',FN-NAME 'CONVERT-FORM)
                   #'(LAMBDA ,ARGLIST ,@REST))
            (PROGN (WARN "No symbol ~:@(~a~) found in IL package." NAME)
                   NIL))))
```

;; Used when an Interlisp function is the same as the Common Lisp function of the same name.

```
(DEFMACRO IL-COPYDEF (NAME &OPTIONAL (NEWNAME NAME))
   (LET ((SYM (FIND-SYMBOL (SYMBOL-NAME NEWNAME)
                     *IL-PACKAGE*)))
        (UNLESS SYM (ERROR "No symbol ~:@(~a~) found in IL package." SYM))
        `(SETF (GET ',SYM 'CONVERT-FORM)
               #'(LAMBDA (&REST ARGS)
                         (CONS ',NAME (MAPCONVERT ARGS))))))
```

;; Used to define a run-time function (not a converter function).

```
(XCL:DEFDEFINER IL-DEFUN IL:FUNCTIONS (NAME &REST REST)
   (CHECK-TYPE NAME SYMBOL)
   (LET* ((NAME-STRING (SYMBOL-NAME NAME))
          (IL-SYM (INTERN NAME-STRING 'IL))
          (IL-SYM1 (IF (CHAR/= (ELT NAME-STRING 0)
                               #\/)
                       (INTERN (CONCATENATE 'STRING "/" NAME-STRING)
                               'IL))))
         `(PROGN (EXPORT ',IL-SYM 'IL)
                 (DEFUN ,IL-SYM ,@REST)                          ; Also make a version starting with a /
                ,@(IF IL-SYM1
                      `((EXPORT ',IL-SYM1 'IL)
                        (SETF (SYMBOL-FUNCTION ',IL-SYM1)
                              (SYMBOL-FUNCTION ',IL-SYM)))))))

(XCL:DEFDEFINER IL-DEFVAR IL:FUNCTIONS (NAME &REST ARGS)
   (LET ((IL-SYM (INTERN (SYMBOL-NAME NAME)
                        *IL-PACKAGE*)))
        `(PROGN (EXPORT ',IL-SYM 'IL)
                (DEFVAR ,IL-SYM ,@(MAPCONVERT ARGS)))))
```

;;
;; ; Creates an external symbol in the IL package.
;; (defmacro il-defsym (name)
;;   `(export (intern (symbol-name ',name) *il-package*) *il-package*))

;; (defmacro il-import (symbol)
;;   `(progn (import ,symbol 'il)
;;   (export (find-symbol (symbol-name ,symbol) 'il) 'il)))

```
(DEFMACRO IL-COPYCONV (OLDNAME NEWNAME)
   (LET* ((OLD-SYM (FIND-SYMBOL (SYMBOL-NAME OLDNAME)
                          *IL-PACKAGE*))
          (NEW-SYM (FIND-SYMBOL (SYMBOL-NAME NEWNAME)
                          *IL-PACKAGE*)))
         (UNLESS OLD-SYM (ERROR "No symbol ~:@(~a~) found in IL package." OLD-SYM))
         (UNLESS NEW-SYM (ERROR "No symbol ~:@(~a~) found in IL package." NEW-SYM))
         `(SETF (GET ',NEW-SYM 'CONVERT-FORM)
                #'(LAMBDA (&REST ARGS)
                          (APPLY (GET ',OLD-SYM 'CONVERT-FORM)
                                 ARGS)))))
```

;; Defines a "Non-conversion" form for use with things like \GETBASE.

```
(XCL:DEFDEFINER IL-WARNINGFORM IL:FUNCTIONS (NAME &OPTIONAL (TEMPLATE '(NIL REPEAT (EVAL)))
                                                 (WARN-SWITCH '*WARN-ON-UNTRANSLATABLE-IL-FORM*))
   (LET ((FN-NAME (FIND-SYMBOL (SYMBOL-NAME NAME)
                        *IL-PACKAGE*)))
        (IF FN-NAME
            `(SETF (GET ',FN-NAME 'CONVERT-FORM)
                   #'(LAMBDA (&REST REST)
                             (DECLARE (SPECIAL ,WARN-SWITCH))
                             (WHEN ,WARN-SWITCH
                                 (WARN "Unable to translate a ~a form." ',FN-NAME))
                             (WALK-TEMPLATE (CONS ',FN-NAME REST)
                                    ',TEMPLATE)))
            (PROGN (WARN "No symbol ~:@(~a~) found in IL package." NAME)
                   NIL))))
```

;; Defines a function (e.g. PROGN-IF-NEEDED) that takes a list and sticks a PROGN (or whatever) at the beginning if the length is not 1. Used to
;; eliminate ugly redundant PROGNs. If the length is 0, returns whatever the form itself returns when given no arguments (e.g. T for AND, NIL for OR).

```
(MACROLET ((DEF-*-IF-NEEDED (NAME)
                  (LET ((NAME-STRING (SYMBOL-NAME NAME)))
                       `(DEFUN ,(INTERN (CONCATENATE 'STRING NAME-STRING "-IF-NEEDED")) (ARGS)
                           (CASE (LENGTH ARGS)
                               (0 ,(EVAL `(,NAME)))
                               (1 (FIRST ARGS))
                               (T `(,',NAME ,@ARGS)))))))
       (DEF-*-IF-NEEDED PROGN)
       (DEF-*-IF-NEEDED AND)
       (DEF-*-IF-NEEDED OR))


(DEFSTRUCT (FAKE-SYMBOL (:CONSTRUCTOR MAKE-FAKE-SYMBOL (NAME))
                        (:PRINT-FUNCTION (LAMBDA (OBJ STREAM DEPTH)
                                           (PRINC (FAKE-SYMBOL-NAME OBJ)
                                                  STREAM))))
   NAME)


(DEFSTRUCT (SHARP-DOT (:PRINT-FUNCTION (LAMBDA (SELF STREAM DEPTH)
                                         (WRITE-STRING "#." STREAM)
                                         (WRITE (SHARP-DOT-CONTENTS SELF)
                                                :STREAM STREAM))))
   CONTENTS)


(DEFSTRUCT (SHARP-COMMA (:PRINT-FUNCTION (LAMBDA (SELF STREAM DEPTH)
                                           (WRITE-STRING "#," STREAM)
                                           (WRITE (SHARP-COMMA-CONTENTS SELF)
                                                  :STREAM STREAM))))
   CONTENTS)
```

;; Aux function to see whether or not to generate a symbolp check

```
(DEFUN QUOTED-SYMBOL-P (X)
   (AND (CONSP X)
        (EQ (CAR X)
            'QUOTE)
        (SYMBOLP (CADR X))
        (NULL (CDDR X))))


(DEFVAR *ORIGINAL-READTABLE* (COPY-READTABLE NIL))


(DEFUN OLD-CONVERT-FILE (INFILE OUTFILE)
   (WITH-OPEN-FILE (INSTREAM INFILE)
          (IF OUTFILE
              (WITH-OPEN-STREAM (OUTSTREAM (COND
                                              ((EQ OUTFILE 'T)
                                               (MAKE-BROADCAST-STREAM *STANDARD-OUTPUT*))
                                              (T (OPEN OUTFILE :DIRECTION :OUTPUT :IF-EXISTS :SUPERSEDE
                                                        :IF-DOES-NOT-EXIST :CREATE))))
                     (CONVERT-FILE-INTERNAL INSTREAM OUTSTREAM))
              (CONVERT-FILE-INTERNAL INSTREAM NIL))))

(EXPORT 'CONVERT-FILE)
```

;; (convert-file "~/medley/ADISPLAY" "adisplay") (convert-file "foo1" "foo2") (convert-file "foo3" "foo4")

```
(EXPORT '(READ-EXPORTS WRITE-EXPORTS READ-RECORD-TYPES WRITE-RECORD-TYPES))


(DEFUN READ-EXPORTS (FILE)
   ;; Read the exported-symbols file if it exists
   (WITH-OPEN-FILE (STREAM FILE :IF-DOES-NOT-EXIST NIL)
          (WHEN STREAM
              (READ STREAM)                                     ; Read the "(in-package)" form
              (SETQ *EXPORTED-IL-SYMBOLS* (CADADR (READ STREAM))))))
```

;; Get the symbol list

```
(DEFUN WRITE-EXPORTS (FILE)
   (WITH-OPEN-FILE (STREAM FILE :DIRECTION :OUTPUT :IF-EXISTS :SUPERSEDE :IF-DOES-NOT-EXIST :CREATE)
          (SETQ *EXPORTED-IL-SYMBOLS* (SORT *EXPORTED-IL-SYMBOLS* #'STRING< :KEY #'SYMBOL-NAME))
          (LET ((*PACKAGE* *IL-PACKAGE*))
               (FORMAT STREAM "(lisp:in-package \"IL\")~%(lisp:export '(")
               (DOLIST (SYM *EXPORTED-IL-SYMBOLS*)
                   (FORMAT STREAM "~% ~s" SYM))
               (FORMAT STREAM ")~%"))))


(DEFUN READ-RECORD-TYPES (FILE)                                 ; Read the record-types file if it exists
   (WITH-OPEN-FILE (STREAM FILE :IF-DOES-NOT-EXIST NIL)
          (WHEN STREAM (READ-HASH-TABLE *RECORD-TYPES* STREAM))))


(DEFUN WRITE-RECORD-TYPES (FILE)
   (WITH-OPEN-FILE (STREAM FILE :DIRECTION :OUTPUT :IF-EXISTS :SUPERSEDE :IF-DOES-NOT-EXIST :CREATE)
          (WRITE-HASH-TABLE *RECORD-TYPES* STREAM)
          (TERPRI STREAM)))
```

```
(DEFUN READ-HASH-TABLE (HT STREAM &AUX ITEM)
   (LOOP (WHEN (EQ (SETQ ITEM (READ STREAM NIL 'STOP))
                   'STOP)
               (RETURN))
         (SETF (GETHASH (CAR ITEM)
                      HT)
               (CDR ITEM))))


(DEFUN WRITE-HASH-TABLE (HT STREAM)
   (LET* ((COUNT (HASH-TABLE-COUNT HT))
          (SORTED-TABLE (MAKE-ARRAY COUNT))
          (I 0))
         (MAPHASH #'(LAMBDA (KEY VALUE)
                            (SETF (SVREF SORTED-TABLE I)
                                  (CONS KEY VALUE))
                            (INCF I))
                HT)
         (SORT SORTED-TABLE #'STRING< :KEY #'(LAMBDA (X)
                                                    (SYMBOL-NAME (CAR X))))
         (DOTIMES (I COUNT)
             (PPRINT (SVREF SORTED-TABLE I)
                    STREAM))))


(DEFUN CONVERT-FILE (FILENAME OUTFILE)
   (LET* ((REAL-FILENAME (FIND-SYMBOL (STRING FILENAME)
                                (FIND-PACKAGE 'IL)))
          (COMS (SYMBOL-VALUE (OR (CAAR (GET REAL-FILENAME 'IL:FILE))
                                  (ERROR "~a has no FILES definition." FILENAME)))))
         (IF OUTFILE
             (WITH-OPEN-STREAM (OUTSTREAM (COND
                                             ((EQ OUTFILE 'T)
                                              (MAKE-BROADCAST-STREAM *STANDARD-OUTPUT*))
                                             (T (OPEN OUTFILE :DIRECTION :OUTPUT :IF-EXISTS :SUPERSEDE
                                                      :IF-DOES-NOT-EXIST :CREATE))))
                    (CONVERT-FILECOMS COMS REAL-FILENAME OUTSTREAM))
             (CONVERT-FILECOMS COMS REAL-FILENAME NIL))))


(DEFUN CONVERT-FILECOMS (COMS FILENAME &OPTIONAL OUTSTREAM)
   (LET ((*EXPORTED-IL-SYMBOLS* NIL)
         REORDERED-FILECOMS CONVERTED-FILE-LIST)
        (FORMAT T "~&Processing Forms...~%")
        (SETQ REORDERED-FILECOMS (REORDER-FILECOMS COMS)
              CONVERTED-FILE-LIST
              (EXPURGATE-EXTRANEOUS-PROGNS (MAPCAR 'CONVERT-ONE-FILECOM REORDERED-FILECOMS)))
        (WHEN OUTSTREAM
            (FORMAT T "~&Writing output...")
            (LET* ((MFE (GET FILENAME 'IL:MAKEFILE-ENVIRONMENT))
                   (*PACKAGE* (OR (FIND-PACKAGE (EVAL (GETF MFE :PACKAGE)))
                                  *IL-PACKAGE*))
                   (*PRINT-PRETTY* T)
                   (*PRINT-CASE* :DOWNCASE))
                  (WHEN MFE
                      (PRINT '(IN-PACKAGE "INTERLISP" :USE NIL :NICKNAMES '("IL"))
                             OUTSTREAM))
                  (PRINT (IF MFE
                             (LIST 'IN-PACKAGE (GETF MFE ':PACKAGE))
                             '(IN-PACKAGE "INTERLISP" :USE NIL :NICKNAMES '("IL")))
                         OUTSTREAM)
                  (TERPRI OUTSTREAM)
                  (WHEN *EXPORTED-IL-SYMBOLS*
                      (PRINT (MAKE-EXPORT-FORM *EXPORTED-IL-SYMBOLS*)
                             OUTSTREAM)
                      (TERPRI OUTSTREAM))
                  (DOLIST (FORM CONVERTED-FILE-LIST)
                      (WHEN FORM
                          (PRINT FORM OUTSTREAM)
                          (TERPRI OUTSTREAM))))))))


(DEFUN CONVERT-ONE-FILECOM (COM)
   (UNLESS (CONSP COM)
          (ERROR "Invalid filecom: ~s" COM))
   (LET (;; We bind these for the warnings mechanism in case the filecom type is unknown...  They'll be rebound lower down.

         (*CURRENT-EXPRESSION* COM)
         (*CURRENT-DEFINITION* (CAR COM))
         (*CURRENT-DEFINITION-TYPE* "Filecom")
         (*WARNINGS-MADE* NIL)
         (CONVERTER (GET (CAR COM)
                         'CONVERT-COM))

         ;; FILEVARS are handled at this level, except in PROP and IFPROP coms.
```

```
          (FILEVAR-P (AND (EQ (SECOND COM)
                              'IL:*)
                          (NOT (MEMBER (FIRST COM)
                                       '                                  (IL:* IL:PROP IL:IFPROP)))))
         (FUNCALL (OR CONVERTER 'CONVERT-UNKNOWN-COM)
                (IF CONVERTER
                    (IF FILEVAR-P
                        (IL:EVAL (THIRD COM))
                        (CDR COM))
                    COM))))


(DEFUN EXPURGATE-EXTRANEOUS-PROGNS (FORMS-LIST)
   (LET (RESULT)
        (DOLIST (FORM FORMS-LIST)
            (SETQ RESULT (NCONC RESULT (IF (AND (CONSP FORM)
                                                 (EQ (CAR FORM)
                                                     'PROGN))
                                            (EXPURGATE-EXTRANEOUS-PROGNS (CDR FORM))
                                            (CONS FORM NIL)))))
        RESULT))


(DEFUN REORDER-FILECOMS (COMS-LIST)
   (LET (EARLY-LIST LATE-LIST)
        (LABELS ((EARLY-P (COM)
                        (AND (CONSP COM)
                             (OR (MEMBER (CAR COM)
                                         '(IL:CONSTANTS IL:MACROS))
                                 (AND (MEMBER (CAR COM)
                                              '(IL:DECLARE\:))
                                      (SOME #'EARLY-P (CDR COM)))))))
               (DOLIST (COM COMS-LIST)
                   (IF (EARLY-P COM)
                       (PUSH COM EARLY-LIST)
                       (PUSH COM LATE-LIST)))
               (NCONC (NREVERSE EARLY-LIST)
                      (NREVERSE LATE-LIST)))))


(DEFUN MAKE-EXPORT-FORM (LIST-OF-SYMBOLS)
   (LET (SORTED)
        (DOLIST (S LIST-OF-SYMBOLS)
            (LET ((A (ASSOC (SYMBOL-PACKAGE S)
                            SORTED)))
                 (IF A
                     (PUSH S (CDR A))
                     (PUSH (CONS (SYMBOL-PACKAGE S)
                                 (LIST S))
                           SORTED))))
        (CONS 'PROGN (MAPCAR #'(LAMBDA (P)
                                      `(EXPORT (MAPCAR 'INTERN ',(MAPCAR 'STRING (CDR P))
                                                      ',(PACKAGE-NAME (CAR P)))))
                            SORTED))))


(DEFPARAMETER *WALKER-TEMPLATES*
   '(BLOCK (NIL NIL REPEAT (EVAL))
        CATCH
        (NIL EVAL REPEAT (EVAL))
        CHECK-TYPE
        (NIL EVAL REPEAT (NIL))
        COMPILER-LET
        (NIL (REPEAT (NIL EVAL))
             REPEAT
             (EVAL))
        DECLARE
        (REPEAT (NIL))
        EVAL-WHEN
        (NIL QUOTE REPEAT (EVAL))
        FLET
        (NIL (REPEAT ((NIL BINDING-CONTOUR PARAMETER-LIST REPEAT (EVAL))))
             REPEAT
             (EVAL))
        FUNCTION
        (NIL CALL)
        GO
        (NIL QUOTE)
        IF
        (NIL REPEAT (EVAL))
        LABELS
        (NIL (REPEAT ((NIL BINDING-CONTOUR PARAMETER-LIST REPEAT (EVAL))))
             REPEAT
             (EVAL))
        LAMBDA
        (NIL BINDING-CONTOUR PARAMETER-LIST REPEAT (EVAL))
        LET
        (NIL BINDING-CONTOUR (REPEAT ((NIL EVAL)))
             REPEAT
             (EVAL))
        LET*
        (NIL BINDING-CONTOUR (REPEAT ((NIL EVAL)))
             REPEAT
             (EVAL))
        LOCALLY
        (NIL REPEAT (EVAL))
        MACROLET
        (NIL (REPEAT ((NIL NIL REPEAT (EVAL))))
             REPEAT
             (EVAL))
        MULTIPLE-VALUE-CALL
        (NIL EVAL REPEAT (EVAL))
        MULTIPLE-VALUE-LIST
        (NIL EVAL)
        MULTIPLE-VALUE-PROG1
        (NIL RETURN REPEAT (EVAL))
        MULTIPLE-VALUE-SETQ
        (NIL (REPEAT (SET))
             EVAL)
        MULTIPLE-VALUE-BIND
        (NIL BINDING-CONTOUR (REPEAT (SET))
             REPEAT
             (EVAL))
        IL:NLSETQ
        (NIL REPEAT (EVAL))
        PROGN
        (NIL REPEAT (EVAL))
        PROGV
        (NIL EVAL EVAL REPEAT (EVAL))
        QUOTE
        (NIL QUOTE)
        RETURN-FROM
        (NIL QUOTE REPEAT (RETURN))
        SETQ
        (NIL REPEAT (SET EVAL))
        SETF
        (NIL REPEAT (SET EVAL))
        TAGBODY
        (NIL REPEAT (EVAL))
        THE
        (NIL QUOTE EVAL)
        THROW
        (NIL EVAL EVAL)
        UNLESS
        (NIL REPEAT (EVAL))
        UNWIND-PROTECT
        (NIL RETURN REPEAT (EVAL))
        WHEN
        (NIL REPEAT (EVAL))
        DO
        (NIL BINDING-CONTOUR (REPEAT ((BINDING REPEAT (EVAL))))
             (EVAL EVAL)
             REPEAT
             (EVAL))
        DO*
        (NIL BINDING-CONTOUR (REPEAT ((BINDING REPEAT (EVAL))))
             (EVAL EVAL)
             REPEAT
             (EVAL))
        DOLIST
        (NIL (NIL EVAL)
             REPEAT
             (EVAL))
        DOTIMES
        (NIL (NIL EVAL)
             REPEAT
             (EVAL))
        PROG
        (NIL BINDING-CONTOUR (REPEAT ((BINDING EVAL)))
             REPEAT
             (EVAL))
        PROG*
        (NIL BINDING-CONTOUR (REPEAT ((BINDING EVAL)))
             REPEAT
             (EVAL))
        COND
        (NIL REPEAT ((TEST REPEAT (EVAL))))
        DEFINE-SETF-METHOD
        (NIL BINDING-CONTOUR PARAMETER-LIST REPEAT (EVAL))
        DEFUN
        (NIL NAME BINDING-CONTOUR PARAMETER-LIST REPEAT (EVAL))
        DEFMACRO
        (NIL NAME BINDING-CONTOUR PARAMETER-LIST REPEAT (EVAL))
        CASE
        (NIL EVAL REPEAT ((NIL REPEAT (EVAL))))
        ECASE
        (NIL EVAL REPEAT ((NIL REPEAT (EVAL))))
        TYPECASE
        (NIL EVAL REPEAT ((NIL REPEAT (EVAL))))
        ETYPECASE
        (NIL EVAL REPEAT ((NIL REPEAT (EVAL))))
        XCL:DEFDEFINER
        (NIL NIL NIL NIL REPEAT (EVAL))
        INCF
        (NIL EVAL EVAL)
        DECF
        (NIL EVAL EVAL)
        WITH-INPUT-FROM-STRING
        (NIL (NIL EVAL REPEAT (EVAL))
             REPEAT
             (EVAL))
        WITH-OUTPUT-TO-STRING
        (NIL (NIL EVAL)
             REPEAT
             (EVAL))
        WITH-OPEN-FILE
        (NIL (NIL REPEAT (EVAL))
             REPEAT
             (EVAL))
        LOOP
        (NIL REPEAT (EVAL))
        POP
        (NIL EVAL)
        PUSH
        (NIL EVAL EVAL)
        PUSHNEW
        (NIL EVAL EVAL REPEAT EVAL)))


(DEFUN GET-WALKER-TEMPLATE (FN)
   (GETF *WALKER-TEMPLATES* FN NIL))


(DEFUN WALK-FORM-INTERNAL (FORM &AUX NEWFORM NEWNEWFORM WALK-NO-MORE-P MACROP FN TEMPLATE)
   (COND
      ((ATOM FORM)
       (WHEN (AND (SYMBOLP FORM)
                  (NOT (NULL *CURRENT-FREE-REFERENCES*))
                  (NOT (KEYWORDP FORM))
                  (NOT (MEMBER FORM '(T NIL)))
                  (NULL (ASSOC FORM *LOCALS*)))

           ;; Almost certainly a free ref.  Note for later analysis.

           (PUSHNEW FORM *CURRENT-FREE-REFERENCES*))
       FORM)
      ((SETQ TEMPLATE (GET-WALKER-TEMPLATE (SETQ FN (CAR FORM))))
       (IF (SYMBOLP TEMPLATE)
           (FUNCALL TEMPLATE FORM)
           (WALK-TEMPLATE FORM TEMPLATE)))
      ((AND (SYMBOLP FN)
            (OR (GET FN 'CONVERT-FORM)
                (EQ (CAR (GET FN 'IL:CLISPWORD))
                    'IL:FORWORD)))
       (CONVERT FORM))
      ((AND (SYMBOLP FN)
            (MACRO-FUNCTION FN))
       (LET ((*CURRENT-EXPRESSION* FORM))
            (WARN "Macro form ~s not translated" FN))
       FORM)
      ((AND (SYMBOLP FN)
            (NOT (FBOUNDP FN))
            (SPECIAL-FORM-P FN))
       (UNKNOWN-MACRO-FORM FORM))
      (T ;; Otherwise, walk the form as if its just a standard
         ;; functioncall using a template for standard function
         ;; call.
         (WALK-TEMPLATE FORM '(CALL REPEAT (EVAL))))))


(DEFUN WALK-TEMPLATE (FORM TEMPLATE)
   (IF (ATOM TEMPLATE)
       (ECASE TEMPLATE
           ((EVAL SET FUNCTION TEST EFFECT RETURN)
              (WHEN *WALKER-FIND-PARAMETER-LIST*
                  (THROW 'PARAMETER-LIST NIL))
              (WALK-FORM-INTERNAL FORM))
           ((NIL QUOTE) FORM)
           ((BINDING)

              ;; This should only appear inside (after)  a BINDING-CONTOUR...
              (WHEN (SYMBOLP FORM)
                  ;; Perhaps this should note if FORM is declared special somehow...
                  (PUSH (CONS FORM ':LOCAL)
                        *LOCALS*)
                  (PUSHNEW FORM *CURRENT-LOCALS*))
              FORM)
           ((LAMBDA CALL) (COND
                            ((SYMBOLP FORM)
                             (UNLESS (NULL *CURRENT-FUNCTION-CALLS*)
                                    (PUSHNEW FORM *CURRENT-FUNCTION-CALLS*))
                             FORM)
                            (T ;; Have we a "#'foo" here?
                               (WHEN (AND (CONSP FORM)
                                          (EQ (CAR FORM)
                                              'FUNCTION)
                                          (NULL (CDDR FORM))
                                          (SYMBOLP (SECOND FORM)))
                                   ;; Record it if we do...
                                   (PUSHNEW (SECOND FORM)
                                          *CURRENT-FUNCTION-CALLS*))
                               (WALK-FORM-INTERNAL FORM))))
           ((NAME)
              (WHEN (NULL *CURRENT-FUNCTION-CALLS*)
                  ;; Don't record name in a nested def, if we ever see one.
                  (SETQ *CURRENT-DEFINITION* FORM)
                  (PUSH FORM *CURRENT-FUNCTION-CALLS*)
                  (PUSH FORM *CURRENT-FREE-REFERENCES*))
              FORM)
           ((PARAMETER) (IF (SYMBOLP FORM)
                            (WALK-TEMPLATE FORM 'BINDING)
                            (WALK-TEMPLATE FORM '(BINDING EVAL REPEAT (BINDING)))))
           ((PARAMETER-LIST)
              (WHEN *WALKER-FIND-PARAMETER-LIST*
                  ;; Some code-analysis stuff uses this.
                  (THROW 'PARAMETER-LIST FORM))
              (WALK-TEMPLATE FORM '(REPEAT (PARAMETER)))))
       (CASE (CAR TEMPLATE)
           (REPEAT (WALK-TEMPLATE-HANDLE-REPEAT FORM (CDR TEMPLATE)
                          ;; For the case where nothing happens
                          ;; after the repeat optimize out the
                          ;; call to length.
                          (IF (NULL (CDDR TEMPLATE))
                              NIL
                              (NTHCDR (- (LENGTH FORM)
                                         (LENGTH (CDDR TEMPLATE)))
                                     FORM))))
           (IF (WALK-TEMPLATE FORM (IF (IF (LISTP (CADR TEMPLATE))
                                           (EVAL (CADR TEMPLATE))
                                           (FUNCALL (CADR TEMPLATE)
                                                  FORM))
                                       (CADDR TEMPLATE)
                                       (CADDDR TEMPLATE))))
           (BINDING-CONTOUR (LET ((*LOCALS* *LOCALS*))
                                 (WALK-TEMPLATE FORM (CDR TEMPLATE))))
           (REMOTE (WALK-TEMPLATE FORM (CADR TEMPLATE)))
           (WARN
              (WARN (SECOND TEMPLATE))
              (IF (NULL (CDDR TEMPLATE))
                  FORM
                  (WALK-TEMPLATE FORM (CDDR TEMPLATE))))
           (OTHERWISE (COND
                         ((ATOM FORM)
                          FORM)
                         (T (RECONS FORM (WALK-TEMPLATE (CAR FORM)
                                                (CAR TEMPLATE))
                                   (WALK-TEMPLATE (CDR FORM)
                                          (CDR TEMPLATE))))))))))


(DEFUN WALK-TEMPLATE-HANDLE-REPEAT (FORM TEMPLATE STOP-FORM)
   (IF (EQ FORM STOP-FORM)
       (WALK-TEMPLATE FORM (CDR TEMPLATE))
       (WALK-TEMPLATE-HANDLE-REPEAT-1 FORM TEMPLATE (CAR TEMPLATE)
              STOP-FORM)))


(DEFUN WALK-TEMPLATE-HANDLE-REPEAT-1 (FORM TEMPLATE REPEAT-TEMPLATE STOP-FORM)
   (COND
      ((NULL FORM)
       NIL)
      ((EQ FORM STOP-FORM)
       (IF (NULL REPEAT-TEMPLATE)
           (WALK-TEMPLATE STOP-FORM (CDR TEMPLATE))
           (ERROR "While handling repeat:
                                       ~%~Ran into stop while still in repeat template.")))
      ((NULL REPEAT-TEMPLATE)
       (WALK-TEMPLATE-HANDLE-REPEAT-1 FORM TEMPLATE (CAR TEMPLATE)
              STOP-FORM))
      (T (RECONS FORM (WALK-TEMPLATE (CAR FORM)
                                 (CAR REPEAT-TEMPLATE))
                (WALK-TEMPLATE-HANDLE-REPEAT-1 (CDR FORM)
                       TEMPLATE
                       (CDR REPEAT-TEMPLATE)
                       STOP-FORM)))))


(DEFUN WALK-REPEAT-EVAL (FORM ENV)
   (AND FORM (RECONS FORM (WALK-FORM-INTERNAL (CAR FORM))
                    (WALK-REPEAT-EVAL (CDR FORM)))))


(DEFUN RECONS (X CAR CDR)
   (IF (OR (NOT (EQ (CAR X)
                    CAR))
           (NOT (EQ (CDR X)
                    CDR)))
       (CONS CAR CDR)
       X))


(DEFUN RELIST (X &REST ARGS)
   (RELIST-INTERNAL X ARGS NIL))


(DEFUN RELIST* (X &REST ARGS)
   (RELIST-INTERNAL X ARGS 'T))


(DEFUN RELIST-INTERNAL (X ARGS *P)
   (IF (NULL (CDR ARGS))
       (IF *P
           (CAR ARGS)
           (LIST (CAR ARGS)))
       (RECONS X (CAR ARGS)
              (RELIST-INTERNAL (CDR X)
                     (CDR ARGS)
                     *P))))

(DEFVAR *GETVALUE-TRANSLATION* :SLOT-VALUE)

(DEFVAR *CURRENT-DEFINITION*)

(DEFVAR *CURRENT-DEFINITION-TYPE*)

(DEFVAR *CURRENT-EXPRESSION*)

(DEFVAR *CURRENT-LOCALS* NIL)

(DEFVAR *FILE-CONTEXT* NIL)

(DEFVAR *WALKER-FIND-PARAMETER-LIST* NIL)

(DEFVAR *WARNINGS-MADE* NIL)

(DEFVAR *PACKAGE-FOR-IL-SYMBOLS* NIL)

(DEFVAR *PACKAGE-FOR-RESULT-FILE* "CL")

(DEFVAR *PARAMETERS-ALWAYS-OPTIONAL* NIL)

(DEFVAR *PROMPT-FOR-UNKNOWN-MACRO-TEMPLATE* NIL)

(DEFVAR *UNKNOWN-MACRO-ACTION* :UM-WARN)
```

```
(DEFVAR *ALWAYS-INCLUDE-PROPS* NIL)

(IL:DECLARE\: IL:DONTCOPY

(IL:PUTPROPS IL:IL-CONVERT IL:MAKEFILE-ENVIRONMENT (:READTABLE "XCL" :PACKAGE "IL-CONVERT" :BASE 10))

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

(IL:PUTPROPS IL:IL-CONVERT IL:COPYRIGHT ("ENVOS Corporation" 1989 1990))
```

FUNCTION INDEX

VARIABLE INDEX

DEFINER INDEX

STRUCTURE INDEX

MACRO INDEX

PROPERTY INDEX