File created: 19-Sep-2020 22:02:59 **{DSK}<Users>kaplan>Local>medley3.5>lispcore>sources>XCLC-TOP-LEVEL.;10**

changes to: (IL:FUNCTIONS START-COMPILATION)

previous date: 19-Sep-2020 21:33:34 {DSK}<Users>kaplan>Local>medley3.5>lispcore>sources>XCLC-TOP-LEVEL.;9

Read Table: XCL

Package: COMPILER

Format: XCCS

```
(IL:RPAQQ IL:XCLC-TOP-LEVELCOMS
          (;; Top-level entry points

           (IL:STRUCTURES COMPILER-CONTEXT)
           (IL:VARIABLES *COMPILE-FILE-CONTEXT* *COMPILE-SCAN-CONTEXT* *COMPILE-DEFINER-CONTEXT*)
           (IL:FUNCTIONS COMPILER-ERROR)
           (IL:FUNCTIONS COMPILER-APPLY)
           (IL:VARIABLES *EVAL-WHEN-COMPILE* *FASL-HANDLE* *INPUT-FILENAME* *INPUT-STREAM* *LAP-STREAM*
                  *LOAD-COMPILED-CODE* *NEW-COMPILER-IS-EXPANDING* *OUTSTANDING-LOOSE-FORMS* *COMPILING-DEFINER*
                  *LOOSE-NAME*)
           (IL:FUNCTIONS COMPILE-FILE)
           (IL:FUNCTIONS START-COMPILATION FINISH-COMPILATION)
           (IL:FUNCTIONS SCAN-ONE-FORM FUNCTION-P)
           (IL:FUNCTIONS COMPILER-MESSAGE COMPILING-MESSAGE DONE-MESSAGE)
           (IL:COMS (IL:STRUCTURES UNKNOWN-FUNCTION-WARNING)
                  (IL:FUNCTIONS CHECK-FOR-UNKNOWN-FUNCTION WARN-ABOUT-UNKNOWN-FUNCTIONS))
           (IL:VARIABLES *PROCESSED-FUNCTIONS* *UNKNOWN-FUNCTIONS* *CURRENT-FUNCTION*)
           (IL:COMS (IL:STRUCTURES ASSEMBLER-ERROR)
                  (IL:FUNCTIONS ASSEMBLER-ERROR))

           ;; Reading the #, macro

           (IL:VARIABLES *COMPILER-IS-READING*)
           (IL:STRUCTURES EVAL-WHEN-LOAD)

           ;; Support for Block Compilation

           (IL:VARIABLES *BLOCK-HASH-TABLE* *BLOCKS* *CURRENT-BLOCK*)
           (IL:STRUCTURES BLOCK-DECL)
           (IL:FUNCTIONS SET-UP-BLOCK-DECLS)

           ;; Processing of top-level forms in a file

           (IL:VARIABLES PASS)
           (IL:FUNCTIONS CONSTANT-EXPRESSION-P)
           (IL:FUNCTIONS COMPILE-AND-DUMP COMPILE-AND-DUMP-1 COMPILE-ONE-LAMBDA)
           (IL:FUNCTIONS OPTIMIZE-AND-MACROEXPAND OPTIMIZE-AND-MACROEXPAND-1 EXPAND-DEFINER PROCESS-FORMS)
           (IL:FUNCTIONS MAYBE-REMOVE-COMMENTS)
           (IL:FUNCTIONS COMPILE-FILE-SETF-SYMBOL-FUNCTION COMPILE-FILE-DEFINEQ COMPILE-FILE-DEFCONSTANT
                  COMPILE-FILE-DECLARE\: COMPILE-FILE-DEFINE-FILE-INFO COMPILE-FILE-PACKAGE-FORM
                  COMPILE-FILE-PROCLAMATION COMPILE-FILE-COMPILER-LET COMPILE-FILE-MACROLET COMPILE-FILE-DEFINER
                  COMPILE-FILE-NAMED-PROGN COMPILE-FILE-OUTSTANDING-LOOSE-FORMS COMPILE-FILE-LOOSE-FORM
                  COMPILE-FILE-PROCESS-FUNCTION)
           (IL:FUNCTIONS CRACK-DEFMACRO ESTABLISH-MACRO-IN-COMPILER)

           ;; Support for :Process-Entire-File

           (IL:VARIABLES *DEFERRED-FORMS* *MAKING-SECOND-PASS* *PREPROCESSING-PHASE*)
           (IL:FUNCTIONS COMPILE-SCAN-DECLARE\: COMPILE-SCAN-DEFINE-FILE-INFO COMPILE-SCAN-MACROLET
                  COMPILE-SCAN-DEFINER COMPILE-SCAN-LOOSE-FORM COMPILE-SCAN-OUTSTANDING-LOOSE-FORMS)
           (IL:FUNCTIONS MERGE-FIRST-FORMS)

           ;; for compiling definers

           (IL:VARIABLES *LAP-FLG* *AUTOMATIC-SPECIAL-DECLARATIONS*)
           (IL:FUNCTIONS COMPILE COMPILE-DEFINER)
           (IL:FUNCTIONS COMPILE-FORM RAW-COMPILE)
           (IL:FUNCTIONS COMPILE-DEFINER-DEFINER COMPILE-DEFINER-NAMED-PROGN COMPILE-DEFINER-PROCESS-FUNCTION
                  COMPILE-DEFINER-OUTSTANDING-LOOSE-FORMS)

           ;; Arrange for correct compiler to be used.

           (IL:PROP IL:FILETYPE IL:XCLC-TOP-LEVEL)

           ;; Arrange for the correct makefile environment

           (IL:PROP IL:MAKEFILE-ENVIRONMENT IL:XCLC-TOP-LEVEL)))
```

;; Top-level entry points

```
(DEFSTRUCT (COMPILER-CONTEXT (:FAST-ACCESSORS T)
                             (:CONC-NAME NIL)
                             (:COPIER NIL)
                             (:PREDICATE NIL))
   SETF-SYMBOL-FUNCTION-FN
   DEFINEQ-FN
   DEFCONSTANT-FN
   DECLARE\:-FN
   DEFINE-FILE-INFO-FN
   PACKAGE-FORM-FN
   PROCLAIM-FN
   COMPILER-LET-FN
   MACROLET-FN
   DEFINER-FN
   NAMED-PROGN-FN
   PROCESS-FUNCTION-FN
   PROCESS-LOOSE-FORM-FN
   PROCESS-OUTSTANDING-LOOSE-FORMS-FN)


(DEFPARAMETER *COMPILE-FILE-CONTEXT*
   (MAKE-COMPILER-CONTEXT :SETF-SYMBOL-FUNCTION-FN 'COMPILE-FILE-SETF-SYMBOL-FUNCTION :DEFINEQ-FN
          'COMPILE-FILE-DEFINEQ :DEFCONSTANT-FN 'COMPILE-FILE-DEFCONSTANT :DECLARE\:-FN 'COMPILE-FILE-DECLARE\:
          :DEFINE-FILE-INFO-FN 'COMPILE-FILE-DEFINE-FILE-INFO :PACKAGE-FORM-FN 'COMPILE-FILE-PACKAGE-FORM
          :PROCLAIM-FN 'COMPILE-FILE-PROCLAMATION :COMPILER-LET-FN 'COMPILE-FILE-COMPILER-LET :MACROLET-FN
          'COMPILE-FILE-MACROLET :DEFINER-FN 'COMPILE-FILE-DEFINER :NAMED-PROGN-FN 'COMPILE-FILE-NAMED-PROGN
          :PROCESS-FUNCTION-FN 'COMPILE-FILE-PROCESS-FUNCTION :PROCESS-LOOSE-FORM-FN 'COMPILE-FILE-LOOSE-FORM
          :PROCESS-OUTSTANDING-LOOSE-FORMS-FN 'COMPILE-FILE-OUTSTANDING-LOOSE-FORMS))


(DEFPARAMETER *COMPILE-SCAN-CONTEXT*
   (MAKE-COMPILER-CONTEXT :SETF-SYMBOL-FUNCTION-FN 'COMPILE-SCAN-LOOSE-FORM :DEFINEQ-FN
          'COMPILE-SCAN-LOOSE-FORM :DEFCONSTANT-FN 'COMPILE-SCAN-LOOSE-FORM :DECLARE\:-FN
          'COMPILE-SCAN-DECLARE\: :DEFINE-FILE-INFO-FN 'COMPILE-SCAN-DEFINE-FILE-INFO :PACKAGE-FORM-FN
          'COMPILE-FILE-PACKAGE-FORM :PROCLAIM-FN 'COMPILE-FILE-PROCLAMATION :COMPILER-LET-FN
          'COMPILE-SCAN-LOOSE-FORM :MACROLET-FN 'COMPILE-SCAN-MACROLET :DEFINER-FN 'COMPILE-SCAN-DEFINER
          :NAMED-PROGN-FN 'COMPILE-SCAN-LOOSE-FORM :PROCESS-FUNCTION-FN 'COMPILER-ERROR :PROCESS-LOOSE-FORM-FN
          'COMPILE-SCAN-LOOSE-FORM :PROCESS-OUTSTANDING-LOOSE-FORMS-FN 'COMPILE-SCAN-OUTSTANDING-LOOSE-FORMS))


(DEFPARAMETER *COMPILE-DEFINER-CONTEXT*
   (MAKE-COMPILER-CONTEXT :SETF-SYMBOL-FUNCTION-FN 'COMPILE-FILE-SETF-SYMBOL-FUNCTION :DEFINEQ-FN
          'COMPILE-FILE-DEFINEQ :DEFCONSTANT-FN 'COMPILE-FILE-DEFCONSTANT :DECLARE\:-FN 'COMPILER-ERROR
          :DEFINE-FILE-INFO-FN 'COMPILER-ERROR :PACKAGE-FORM-FN 'COMPILE-FILE-PACKAGE-FORM :PROCLAIM-FN
          'COMPILE-FILE-PROCLAMATION :COMPILER-LET-FN 'COMPILE-FILE-COMPILER-LET :MACROLET-FN
          'COMPILE-FILE-MACROLET :DEFINER-FN 'COMPILE-DEFINER-DEFINER :NAMED-PROGN-FN
          'COMPILE-DEFINER-NAMED-PROGN :PROCESS-FUNCTION-FN 'COMPILE-DEFINER-PROCESS-FUNCTION
          :PROCESS-LOOSE-FORM-FN 'COMPILE-FILE-LOOSE-FORM :PROCESS-OUTSTANDING-LOOSE-FORMS-FN
          'COMPILE-DEFINER-OUTSTANDING-LOOSE-FORMS))


(DEFUN COMPILER-ERROR (COMPILER-CONTEXT &REST ARGS)
   (ERROR "Unexpected compiler error. Context is ~s args are ~s" COMPILER-CONTEXT ARGS))


(DEFMACRO COMPILER-APPLY (KEY COMPILER-CONTEXT &OPTIONAL FORM &REST OTHER-ARGS)
   (LET ((ACCESSOR (INTERN (CONCATENATE 'STRING (STRING KEY)
                                  "-FN")
                          (FIND-PACKAGE "COMPILER"))))
        (IF FORM
            `(FUNCALL (,ACCESSOR ,COMPILER-CONTEXT)
                    ,COMPILER-CONTEXT
                    ,FORM
                    ,@OTHER-ARGS)
            `(FUNCALL (,ACCESSOR ,COMPILER-CONTEXT)
                    ,COMPILER-CONTEXT))))


(DEFVAR *EVAL-WHEN-COMPILE* NIL
   "Bound to T during processing of forms to be evaluated at compile-time.")


(DEFVAR *FASL-HANDLE* NIL
   "Handle used for writing out the code to a FASL file.")


(DEFVAR *INPUT-FILENAME* NIL
   "Full name of the file being compiled.")


(DEFVAR *INPUT-STREAM* NIL
   "Stream from which compile-file reads forms.")


(DEFVAR *LAP-STREAM* NIL
   "Stream to which compile-file writes LAP output.")


(DEFVAR *LOAD-COMPILED-CODE* NIL
   "Non-nil if new compiled code should be installed in running Lisp.
 :save   if old versions should be saved on
   the property list before installing")


(DEFVAR *NEW-COMPILER-IS-EXPANDING* NIL
   "Bound to T whenever the new compiler might be expanding macros.  Used in some optimizers to let them only
   take effect in the new compiler.")

(DEFVAR *OUTSTANDING-LOOSE-FORMS* NIL
   "A list of the random top-level forms to be gathered together into a single lambda for compilation.")

(DEFVAR *COMPILING-DEFINER* NIL)

(DEFVAR *LOOSE-NAME* NIL)

(DEFUN COMPILE-FILE (INPUT-FILE &KEY (OUTPUT-FILE NIL)
                                (LAP-FILE NIL)
                                (ERROR-FILE NIL)
                                (ERRORS-TO-TERMINAL T)
                                (FILE-MANAGER-FORMAT NIL F-M-F-GIVEN)
                                (PROCESS-ENTIRE-FILE NIL P-E-F-GIVEN)
                                (LOAD NIL))

;;; Compiles the forms on Input-File, producing a FASL file.

;;;  :Output-File
   ;; The name of a file to which binary code should be written.
   ;;  Defaults to Input-File with the extension '.dfasl'

;;; :Lap-File
   ;; The name of a file to which LAP assemble code should be written.
   ;;  If T, defulats to Input-File with the extension '.dlap', if NIL, no LAP file is produced.

;;; :Error-FIle
   ;; The name of a file to which compiler error messages should be written. Defaults like :Lap-File, but with the extension '.log'

;;; :Errors-To-Terminal
   ;; True if error messages should be sent to *ERROR-OUTPUT* as well as any :Error-File.

;;; :File-Manager-Format
   ;; True if the file should be assumed to have been produced by the MAKEFILE function.
   ;; If not specified, we check the first non-blank character in the file. If that character is a left-paren, we assume that MAKEFILE made the file.

;;; :Process-Entire-File
   ;; If true, the whole file is read in, evaluating those forms which are explicitly or implicitly EVAL-WHEN (OMPILE), before any code is generated. This
   ;; allows macros to be defined after use, for example. This defaults to T if the file is declared or discovered to be in Interlisp format.

;;; :Load
   ;; If true, definitions will be installed in the environment after they are compiled. If this is :SAVE, the any previous definitions are saved on the
   ;; property list before the new ones are installed.
   (LET ((*ERROR-OUTPUT* *ERROR-OUTPUT*)
         (*INPUT-STREAM* NIL)
         (*INPUT-FILENAME* NIL)
         (*FASL-HANDLE* NIL)
         (FASL-PATHNAME NIL)
         (*LAP-STREAM* NIL)
         (ERROR-FILE-STREAM NIL)
         (COMPILATION-SUCCEEDED NIL)
         (*LOAD-COMPILED-CODE* LOAD)
         (*CONTEXT* (MAKE-CONTEXT :TOP-LEVEL-P T :VALUES-USED 0))
         (*ENVIRONMENT* (MAKE-ENV :PARENT T))
         (*CONSTANTS-HASH-TABLE* (MAKE-HASH-TABLE))
         (*OUTSTANDING-LOOSE-FORMS* NIL)
         (*PROCESSED-FUNCTIONS* NIL)
         (*UNKNOWN-FUNCTIONS* NIL)
         (*INPUT-FILECOMS-VARIABLE* NIL)                        ; Bound for the convenience of the optimizers on RPAQQ and
                                                                ; PRETTYCOMPRINT.

         ;; Rebind all of these both to set up a canonical environment inside the compiler and to protect the outside environment from anything that
         ;; might happen during this file.

         (IL:SPECVARS T)
         (IL:LOCALVARS IL:SYSLOCALVARS)
         (IL:LOCALFREEVARS NIL)
         (IL:GLOBALVARS IL:GLOBALVARS)
         (IL:NLAMA IL:NLAMA)
         (IL:NLAML IL:NLAML)
         (IL:LAMA IL:LAMA)
         (IL:DONTCOMPILEFNS IL:DONTCOMPILEFNS))
        (DECLARE (SPECIAL IL:SPECVARS IL:LOCALVARS IL:LOCALFREEVARS IL:GLOBALVARS IL:NLAMA IL:NLAML IL:LAMA
                        IL:DONTCOMPILEFNS))
        (UNWIND-PROTECT
            (PROGN ;; Set up the input stream.
                (LET ((PATH (OR (PROBE-FILE INPUT-FILE)
                                (PROBE-FILE (MERGE-PATHNAMES INPUT-FILE ".lisp")))))
                     (COND
                        (PATH (SETQ *INPUT-FILENAME* PATH)
                              (SETQ *INPUT-STREAM* (OPEN PATH :DIRECTION :INPUT))
                              (WHEN (AND (FBOUNDP 'IL:OPENTEXTSTREAM)
                                         (FBOUNDP 'IL:\\TEDIT.FORMATTEDP1)
                                         (IF (IL:RANDACCESSP *INPUT-STREAM*)
                                             (IL:\\TEDIT.FORMATTEDP1 *INPUT-STREAM*)
                                             (WITH-OPEN-FILE (TEMP-STREAM *INPUT-STREAM*)
                                                    (IL:\\TEDIT.FORMATTEDP1 TEMP-STREAM))))
                                  (SETQ *INPUT-STREAM* (IL:OPENTEXTSTREAM *INPUT-STREAM* NIL NIL NIL
                                                              '(IL:READONLY T)))))
                        (T (ERROR "The file \"~A\" is nonexistent or cannot be read.~%" INPUT-FILE))))
```

;; Set up the FASL output stream.

```
                (SETQ FASL-PATHNAME (COND
                                       (OUTPUT-FILE (PATHNAME OUTPUT-FILE))
                                       (T (MAKE-PATHNAME :TYPE (STRING (LOCALLY (DECLARE (SPECIAL
                                                                                          IL:FASL.EXT)
                                                                                         )
                                                                               IL:FASL.EXT))
                                                 :VERSION :NEWEST :DEFAULTS *INPUT-FILENAME*))))
                (SETQ *FASL-HANDLE* (FASL:OPEN-FASL-HANDLE FASL-PATHNAME))
```

;; Set up the LAP stream.

```
                (WHEN LAP-FILE
                    (SETQ *LAP-STREAM* (OPEN (IF (EQ LAP-FILE T)
                                                 (MAKE-PATHNAME :TYPE "dlap" :VERSION :NEWEST :DEFAULTS
                                                        *INPUT-FILENAME*)
                                                 LAP-FILE)
                                             :DIRECTION :OUTPUT)))
```

;; Set up the error output stream.

```
                (WHEN ERROR-FILE
                    (SETQ ERROR-FILE-STREAM (OPEN (IF (EQ ERROR-FILE T)
                                                      (MAKE-PATHNAME :TYPE "log" :VERSION :NEWEST :DEFAULTS
                                                             *INPUT-FILENAME*)
                                                      ERROR-FILE)
                                                  :DIRECTION :OUTPUT)))
                (SETQ *ERROR-OUTPUT* (IF ERRORS-TO-TERMINAL
                                         (IF ERROR-FILE-STREAM
                                             (MAKE-BROADCAST-STREAM ERROR-FILE-STREAM *ERROR-OUTPUT*)
                                             *ERROR-OUTPUT*)
                                         ERROR-FILE-STREAM))
```

;; Fix up the default values of FILE-MANAGER-FORMAT and PROCESS-ENTIRE-FILE.

```
                (IF (NOT F-M-F-GIVEN)
                    (SETQ FILE-MANAGER-FORMAT (EQ (IL:SKIPSEPRCODES *INPUT-STREAM* IL:FILERDTBL)
                                                  (IL:CHARCODE "("))))
                (IF (NOT P-E-F-GIVEN)
                    (SETQ PROCESS-ENTIRE-FILE FILE-MANAGER-FORMAT))
```

;; Pick the right readtable and do the compilation.

```
                (IL:WITH-READER-ENVIRONMENT (IF FILE-MANAGER-FORMAT
                                                IL:*OLD-INTERLISP-READ-ENVIRONMENT*
                                                IL:*COMMON-LISP-READ-ENVIRONMENT*)
                    (START-COMPILATION)
                    (PROCESS-FORMS PROCESS-ENTIRE-FILE)
                    (FINISH-COMPILATION)
                    (SETQ COMPILATION-SUCCEEDED T)
```

;; Return the DFASL pathname so that people can say, for example, (LOAD (COMPILE-FILE ...))

```
                    FASL-PATHNAME))
```

;; The compilation is over. Close all of the streams. If the compilations did not succeed (that is, we have aborted it), then delete the
;; FASL file as well rather than leave garbage around.

```
         (IF (STREAMP *INPUT-STREAM*)
             (CLOSE *INPUT-STREAM*))
         (IF (NOT (NULL *FASL-HANDLE*))
             (FASL:CLOSE-FASL-HANDLE *FASL-HANDLE* :ABORT (NOT COMPILATION-SUCCEEDED)))
         (IF (STREAMP ERROR-FILE-STREAM)
             (CLOSE ERROR-FILE-STREAM))
         (IF (STREAMP *LAP-STREAM*)
             (CLOSE *LAP-STREAM*)))))
```

(DEFUN **START-COMPILATION** ()   ; Edited 19-Sep-2020 22:02 by rmk:

;;; Write out banners on the various output files.

```
   (FLET ((DATE-STRING (UNIV-TIME)
                 (MULTIPLE-VALUE-BIND (SECONDS MINUTES HOUR DATE MONTH YEAR DAY-OF-WEEK)
                     (DECODE-UNIVERSAL-TIME UNIV-TIME)
                   (FORMAT NIL "~A, ~D ~A ~D, ~D:~2,'0D:~2,'0D"
                          (NTH DAY-OF-WEEK '("Monday" "Tuesday" "Wednesday" "Thursday" "Friday" "Saturday"
                                             "Sunday"))
```

```
                              DATE
                              (NTH (1- MONTH)
                                   '("January" "February" "March" "April" "May" "June" "July" "August" "September"
                                           "October" "November" "December"))
                              YEAR HOUR MINUTES SECONDS))))
         (LET ((FASL-STREAM (FASL:BEGIN-TEXT *FASL-HANDLE*))
               (FILECREATED (IL:READ-FILECREATED *INPUT-STREAM*)))
```

;; RMK: This had a complicated format, didn't work, so I reverted to printout. PRIN3 to stop wrap around

```
              (IL:PRINTOUT FASL-STREAM "XCL Compiler output for source file " IL:\# (IL:PRIN3
                                                                                      (OR (CADDR FILECREATED)
                                                                                          (NAMESTRING
                                                                                                 *INPUT-FILENAME*
                                                                                                 ))
                                                                                      FASL-STREAM)
                     T "Source file created " IL:\# (IL:PRIN3 (OR (CADR FILECREATED)
                                                                   (DATE-STRING (FILE-WRITE-DATE *INPUT-FILENAME*
                                                                                       )))
                                                              FASL-STREAM)
                     T "FASL file created " IL:\# (IL:PRIN3 (DATE-STRING (GET-UNIVERSAL-TIME))
                                                            FASL-STREAM)
                     T))
         (FASL:BEGIN-BLOCK *FASL-HANDLE*)
         (WHEN (STREAMP *LAP-STREAM*)
             (FORMAT *LAP-STREAM* "XCL Compiler output for source file ~A~%~
Source file created ~A.~%~
LAP file
                    created ~A.~%~%" (NAMESTRING *INPUT-FILENAME*)
                    (DATE-STRING (FILE-WRITE-DATE *INPUT-FILENAME*))
                    (DATE-STRING (GET-UNIVERSAL-TIME))))))
```

(DEFUN **FINISH-COMPILATION** ()

;;; Clean up after the compilation.

;; Remove this file from IL:NOTCOMPILEDFILES for CLEANUP.

```
   (LOCALLY (DECLARE (IL:GLOBALVARS IL:NOTCOMPILEDFILES))
          (SETQ IL:NOTCOMPILEDFILES (REMOVE (INTERN (LET ((TYPE (PATHNAME-TYPE *INPUT-FILENAME*)))
                                                         (STRING-UPCASE (IF (ZEROP (LENGTH TYPE))
                                                                                (PATHNAME-NAME *INPUT-FILENAME*)
                                                                                (FORMAT NIL "~A.~A" (PATHNAME-NAME

                                                                                                     *INPUT-FILENAME*
                                                                                                     )
                                                                                       TYPE))))
                                                    "INTERLISP")
                                            IL:NOTCOMPILEDFILES)))
```

;; Possibly warn about unknown functions encountered during compilation.

(**WARN-ABOUT-UNKNOWN-FUNCTIONS**))

(DEFUN **SCAN-ONE-FORM** (FORM COMPILER-CONTEXT)

;; Assumes sedit like comments have already been stripped

```
   (IF (ATOM FORM)
       FORM
       (CASE (CAR FORM)
           ((IL:FUNCTION FUNCTION QUOTE) (EVAL FORM))
           ((PROGN) (LET ((VALUE NIL))
                         (DOLIST (SUB-FORM (CDR FORM))
                             (SETQ VALUE (SCAN-ONE-FORM SUB-FORM COMPILER-CONTEXT)))
                         VALUE))
           ((DEFMACRO) (LET ((NAME (SECOND FORM)))
                            (COND
                               ((NOT (SYMBOLP NAME))
                                (CERROR "Ignore this DEFMACRO." "~S is not a legal macro name." NAME))
                               (T (UNLESS *MAKING-SECOND-PASS*
                                         (ESTABLISH-MACRO-IN-COMPILER NAME (CRACK-DEFMACRO FORM)))
                                  (SCAN-ONE-FORM (OPTIMIZE-AND-MACROEXPAND-1 FORM)
                                         COMPILER-CONTEXT)))))
           ((EVAL-WHEN) (IF (NOT (AND (LISTP (SECOND FORM))
                                      (NOT (EQ 'QUOTE (CAR (SECOND FORM))))))
                            (CERROR "Ignore its contents." "Ill-formed EVAL-WHEN:~%~S" FORM)
                            (LET ((EVAL-SPECIFIED (OR (MEMBER 'IL:EVAL (CADR FORM)
                                                             :TEST
                                                             #'EQ)
                                                      (MEMBER 'EVAL (CADR FORM)
                                                             :TEST
                                                             #'EQ)))
                                  (LOAD-SPECIFIED (OR (MEMBER 'IL:LOAD (CADR FORM)
                                                             :TEST
                                                             #'EQ)
                                                      (MEMBER 'LOAD (CADR FORM)
                                                             :TEST
                                                             #'EQ)))
                                  (COMPILE-SPECIFIED (OR (MEMBER 'IL:COMPILE (CADR FORM)
                                                                :TEST
                                                                #'EQ)
                                                         (MEMBER 'COMPILE (CADR FORM)
                                                                :TEST
                                                                #'EQ))))
                                 (COND
                                    ((NOT LOAD-SPECIFIED)
                                     (WHEN (OR COMPILE-SPECIFIED (AND *EVAL-WHEN-COMPILE* EVAL-SPECIFIED))
                                         (LET ((VALUE NIL))
                                              (DOLIST (INNER-FORM (CDDR FORM))
                                                  (SETQ VALUE (EVAL INNER-FORM)))
                                              VALUE)))
                                    (T (LET ((*EVAL-WHEN-COMPILE* (OR COMPILE-SPECIFIED (AND *EVAL-WHEN-COMPILE*
                                                                                             EVAL-SPECIFIED))))
                                            (LET ((VALUE NIL))
                                                 (DOLIST (SUB-FORM (CDDR FORM))
                                                     (SETQ VALUE (SCAN-ONE-FORM SUB-FORM COMPILER-CONTEXT)))
                                                 VALUE)))))))
          ((DEFCONSTANT) (COMPILER-APPLY DEFCONSTANT COMPILER-CONTEXT FORM))
          ((IL:DECLARE\:) (COMPILER-APPLY IL:DECLARE\: COMPILER-CONTEXT FORM))
          ((IL:SETF-SYMBOL-FUNCTION) (COMPILER-APPLY IL:SETF-SYMBOL-FUNCTION COMPILER-CONTEXT FORM))
          ((IL:DEFINEQ) (COMPILER-APPLY IL:DEFINEQ COMPILER-CONTEXT FORM))
          ((IL:DEFINE-FILE-INFO) (COMPILER-APPLY IL:DEFINE-FILE-INFO COMPILER-CONTEXT FORM))
          ((MAKE-PACKAGE IN-PACKAGE SHADOW SHADOWING-IMPORT EXPORT UNEXPORT USE-PACKAGE UNUSE-PACKAGE IMPORT
                  DEFPACKAGE) (COMPILER-APPLY PACKAGE-FORM COMPILER-CONTEXT FORM))
          ((PROCLAIM) (COMPILER-APPLY PROCLAIM COMPILER-CONTEXT FORM))
          ((COMPILER-LET) (COMPILER-APPLY COMPILER-LET COMPILER-CONTEXT FORM))
          ((MACROLET SI::%MACROLET) (COMPILER-APPLY MACROLET COMPILER-CONTEXT FORM))
          ((DEFINER) (COMPILER-APPLY DEFINER COMPILER-CONTEXT FORM))
          ((NAMED-PROGN) (COMPILER-APPLY NAMED-PROGN COMPILER-CONTEXT FORM))
          (OTHERWISE (IF *MAKING-SECOND-PASS*
                         (COMPILER-APPLY PROCESS-LOOSE-FORM COMPILER-CONTEXT FORM)
                         (MULTIPLE-VALUE-BIND (NEW-FORM CHANGED-P)
                             (OPTIMIZE-AND-MACROEXPAND-1 FORM)
                           (IF (NOT CHANGED-P)
                               (COMPILER-APPLY PROCESS-LOOSE-FORM COMPILER-CONTEXT FORM)
                               (SCAN-ONE-FORM NEW-FORM COMPILER-CONTEXT))))))))


(DEFUN FUNCTION-P (FORM)
   (AND (CONSP FORM)
        (OR (EQ (FIRST FORM)
                'FUNCTION)
            (EQ (FIRST FORM)
                'IL:FUNCTION))
        (CONSP (SECOND FORM))))


(DEFMACRO COMPILER-MESSAGE (FORMAT-STRING &REST FORMAT-ARGS)
   `(FORMAT *ERROR-OUTPUT* ,FORMAT-STRING ,@FORMAT-ARGS))


(DEFMACRO COMPILING-MESSAGE (NAME)
   `(COMPILER-MESSAGE "Compiling ~a " ,NAME))


(DEFMACRO DONE-MESSAGE ()
   `(COMPILER-MESSAGE " Done~%"))


(DEFINE-CONDITION UNKNOWN-FUNCTION-WARNING (WARNING)
   (CALL-LIST)
   (:REPORT (LAMBDA (CONDITION *STANDARD-OUTPUT*)
                   (FORMAT T "The following functions were called in the code just compiled, but are not known
                          to exist:~%")
                   (DOLIST (PAIR (UNKNOWN-FUNCTION-WARNING-CALL-LIST CONDITION))
                       (FORMAT T "  ~S -- called from " (CAR PAIR))

                       ;; I almost used this hairy thing, but FORMAT is too slow...  Aren't you glad?

                       ;; "~:[nowhere?!~;~:*~{~#[~;~S~;~S and ~S~:;~@{~#[~;and ~]~S~^, ~}~]~}.~]~%"

                       (COND
                          ((NULL (CDR PAIR))
                           (FORMAT T "nowhere?!~%"))
                          ((NULL (CDDR PAIR))
                           (FORMAT T "~S.~%" (SECOND PAIR)))
                          ((NULL (CDDDR PAIR))
                           (FORMAT T "~S and ~S.~%" (SECOND PAIR)
                                  (THIRD PAIR)))
                          (T (DO ((TAIL (CDR PAIR)
                                        (CDR TAIL)))
                                 ((NULL TAIL))
                               (PRIN1 (CAR TAIL))
                               (COND
                                  ((CDDR TAIL)
                                   (PRINC ", "))
                                  ((CDR TAIL)
                                  (PRINC " and "))))
                           (PRINC ".")
                           (TERPRI)))))))
```

(DEFUN **CHECK-FOR-UNKNOWN-FUNCTION** (NAME)

```
   (WHEN (AND (NOT (FBOUNDP NAME))
              (NOT (MEMBER NAME *PROCESSED-FUNCTIONS* :TEST 'EQ))
              (OR (ENV-INLINE-DISALLOWED *ENVIRONMENT* NAME)
                  (NOT (OR (GET NAME 'OPTIMIZER-LIST)
                           (GET NAME 'TRANSFORM)
                           (GET NAME 'IL:DOPVAL)))))
       (LET ((LOOKUP (ASSOC NAME *UNKNOWN-FUNCTIONS* :TEST 'EQ)))
            (IF (NULL LOOKUP)
                (PUSH (LIST NAME *CURRENT-FUNCTION*)
                      *UNKNOWN-FUNCTIONS*)
                (PUSHNEW *CURRENT-FUNCTION* (CDR LOOKUP))))))
```

(DEFUN **WARN-ABOUT-UNKNOWN-FUNCTIONS** ()

;;; If there's anything on *UNKNOWN-FUNCTIONS*, issue a summary and warning.

```
   (WHEN (NOT (NULL *UNKNOWN-FUNCTIONS*))
       (WARN 'UNKNOWN-FUNCTION-WARNING :CALL-LIST *UNKNOWN-FUNCTIONS*)))
```

(DEFVAR ***PROCESSED-FUNCTIONS***

;;; A list of the names of the global functions processed during this compilation. Used in conjunction with *UNKNOWN-FUNCTIONS* to produce a
;;; warning at the end of compilation if there are any functions called but not defined.

```
   )
```

(DEFVAR ***UNKNOWN-FUNCTIONS***

;;; A list containing the names of undefined global functions called from code in the current compilation. Actually, it's an AList mapping the unknown
;;; function to the list of functions in which it is called. Used in conjunction with *PROCESSED-FUNCTIONS* to produce a warning at the end of
;;; compilation if there are any functions called but not defined.

```
   )
```

(DEFVAR ***CURRENT-FUNCTION***

;;; The name of the unit currently being compiled.

```
   )
```

(DEFINE-CONDITION **ASSEMBLER-ERROR**

;;; Signalled by an assembler when it encounters an unrecoverable error. The compiler catches such, prints an error message, and continues with the
;;; next form on the file.

```
    (ERROR)
   (FORMAT-STRING FORMAT-ARGUMENTS)
   (:REPORT (LAMBDA (CONDITION *STANDARD-OUTPUT*)
                   (FORMAT *ERROR-OUTPUT* "Error during assembly:~%  ~?" (ASSEMBLER-ERROR-FORMAT-STRING
                                                                           CONDITION)
                          (ASSEMBLER-ERROR-FORMAT-ARGUMENTS CONDITION)))))
```

(DEFUN **ASSEMBLER-ERROR** (STRING &REST ARGUMENTS)

```
   (ERROR 'ASSEMBLER-ERROR :FORMAT-STRING STRING :FORMAT-ARGUMENTS ARGUMENTS))
```

;; Reading the #, macro

(DEFVAR ***COMPILER-IS-READING*** NIL

```
   "Bound to T during compile-file so that READ can properly treat #,")
```

(DEFSTRUCT **EVAL-WHEN-LOAD**

```
   "Structure wrapping a form to be evaluated at load time.  Used in the implementation of the #, reader
   macro."
   IL:FORM)
```

;; Support for Block Compilation

(DEFVAR ***BLOCK-HASH-TABLE*** NIL

;;; A mapping from function names to lists of BLOCK-DECL structures describing blocks that include that function. Initialized from the list of BLOCK:
;;; declarations gathered into *BLOCKS* (q.v.) during the preprocessing scan.

```
    )
```

(DEFVAR ***BLOCKS*** NIL

;;; A list of the Interlisp block descriptions found on the file. This list is added to during the preprocessing scan of the file and then used for initialising
;;; *BLOCK-HASH-TABLE* (q.v.)

```
    )
```

(DEFVAR ***CURRENT-BLOCK*** NIL

;;; Bound during compilation of a LAMBDA to the BLOCK-DECL structure describing the block containing the current function. This is NIL if the function
;;; is not a part of any block.

```
    )
```

(DEFSTRUCT (**BLOCK-DECL** (:INLINE NIL))

;;; A BLOCK-DECL holds the information describing a particular Interlisp BLOCK: declaration.

;;; NAME is the symbol naming the block or NIL if this is only a pseudo-block.

;;; FN-NAME-MAP is an AList mapping internal function names to their new \BLOCK/FN style name.

;;; SPECVARS, LOCALVARS, LOCALFREEVARS and GLOBALVARS contain the values those variables should have during the compilation of
;;; functions in this block.

```
   NAME
   FN-NAME-MAP
   SPECVARS
   LOCALVARS
   LOCALFREEVARS
   GLOBALVARS)
```

(DEFUN **SET-UP-BLOCK-DECLS** (DECLS)

;;; Parse the given list of Interlisp BLOCK: declarations and return a hash-table mapping functions named therein to a list of the BLOCK-DECLs
;;; representing decls mentioning that function.

```
   (LET ((HASH-TABLE (MAKE-HASH-TABLE)))
        (DOLIST (DECL DECLS)
            (LET* ((BLOCK-NAME (CAR DECL))
                   (BD (MAKE-BLOCK-DECL :NAME BLOCK-NAME))
                   (IL:SPECVARS IL:SPECVARS)
                   (IL:LOCALVARS IL:LOCALVARS)
                   (IL:LOCALFREEVARS NIL)
                   (IL:GLOBALVARS IL:GLOBALVARS)
                   (NOT-RENAMED-FNS (CONS BLOCK-NAME (UNION IL:RETFNS IL:NOLINKFNS)))
                   (FNS NIL))
                  (DECLARE (SPECIAL IL:SPECVARS IL:LOCALVARS IL:LOCALFREEVARS IL:GLOBALVARS IL:NOLINKFNS))

                  ;; We do this next bit because BCOMPL2 does it.

                  (COND
                     ((NULL BLOCK-NAME)
                      (SETQ IL:SPECVARS T)
                      (SETQ IL:LOCALVARS IL:SYSLOCALVARS))
                     (T (SETQ IL:LOCALVARS T)
                        (SETQ IL:SPECVARS IL:SYSSPECVARS)))

                  ;; For each item in the declaration, either add it to the list of functions or make the appropriate modifications to the named
                  ;; variable.

                  (DOLIST (ITEM (CDR DECL))
                      (COND
                         ((SYMBOLP ITEM)
                          (PUSH ITEM FNS)
                          (PUSH BD (GETHASH ITEM HASH-TABLE)))
                         ((CONSP ITEM)
                          (CASE (CAR ITEM)
                              ((IL:SPECVARS IL:LOCALVARS) (IL:EVAL ITEM))
                              ((IL:GLOBALVARS IL:LOCALFREEVARS) (LET ((VARIABLE (CAR ITEM))
                                                                      (VALUE (CDR ITEM)))
                                                                     (WHEN (AND (CONSP VALUE)
                                                                                (EQ (CAR VALUE)
                                                                                    'IL:*))
                                                                         (SETQ VALUE (IL:EVAL (CADR VALUE))))
                                                                     (IF (LISTP VALUE)
                                                                         (SET VARIABLE
                                                                              (UNION (CDR ITEM)
                                                                                     (SYMBOL-VALUE (CAR ITEM))))
                                                                         (SET VARIABLE VALUE))))
                              ((IL:BLKLIBRARY IL:LINKFNS) (WARN "The ~S feature is no longer supported."
                                                                (CAR ITEM)))
                              ((IL:DONTCOMPILEFNS) (WARN "DONTCOMPILEFNS is not supported in BLOCK:
                                                         declarations."))
                              ((IL:BLKAPPLYFNS IL:NOLINKFNS IL:RETFNS IL:ENTRIES)
                                                                           ; These functions should not be renamed, according to
                                                                           ; BYTEBLOCKCOMPILE2.
                                 (WHEN (CONSP (CDR ITEM))
                                     (SETQ NOT-RENAMED-FNS (APPEND (CDR ITEM)
                                                                    NOT-RENAMED-FNS))))
                              (OTHERWISE (CERROR "Ignore the unknown variable." "Unknown variable ~S mentioned
                                                 in a BLOCK: declaration" (CAR ITEM)))))
                         (T (CERROR "Ignore the illegal item" "Illegal item in a BLOCK: declaration: ~S" ITEM)))
)
                  (SETF (BLOCK-DECL-SPECVARS BD)
                        IL:SPECVARS)
                  (SETF (BLOCK-DECL-LOCALVARS BD)
                        IL:LOCALVARS)
                  (SETF (BLOCK-DECL-LOCALFREEVARS BD)
                        IL:LOCALFREEVARS)
                  (SETF (BLOCK-DECL-GLOBALVARS BD)
                        IL:GLOBALVARS)
                  (LET* ((BLOCK-NAME-STRING (STRING BLOCK-NAME))
                         (BLOCK-PACKAGE (SYMBOL-PACKAGE BLOCK-NAME)))
                        (UNLESS (NULL BLOCK-NAME)                  ; NIL blocks don't do renaming.
                            (SETF (BLOCK-DECL-FN-NAME-MAP BD)
                                  (IL:|for| FN IL:|in| (NSET-DIFFERENCE FNS NOT-RENAMED-FNS)
                                     IL:|collect| (CONS FN (INTERN (CONCATENATE 'STRING "\\" BLOCK-NAME-STRING "/"
                                                                               (STRING FN))
                                                                    BLOCK-PACKAGE)))))))
        HASH-TABLE))
```

;; Processing of top-level forms in a file

```
(DEFCONSTANT PASS 'PASS "Useful for ameliorating the obvious quoting bug.")
```

```
(DEFUN CONSTANT-EXPRESSION-P (FORM)
   (OR (CONSTANTP FORM)
       (AND (CONSP FORM)
            (LET* ((FN (CAR FORM))
                   (S-E-DATA (GET FN 'SIDE-EFFECTS-DATA)))
                  (AND (EQ (CAR S-E-DATA)
                           :NONE)
                       (EQ (CDR S-E-DATA)
                           :NONE)
                       (DOLIST (ARG (CDR FORM)
                                    T)
                           (IF (NOT (CONSTANT-EXPRESSION-P ARG))
                               (RETURN NIL))))))))
```

```
(DEFUN COMPILE-AND-DUMP (NAME DEFN KIND)
   (LET ((*CURRENT-BLOCK* NIL)                                    ; So that we aren't dependent upon the top-level binding.

         )
        (COND
           ((AND (SYMBOLP NAME)
                 (EQ KIND :FUNCTION))
            (WHEN (MEMBER NAME IL:DONTCOMPILEFNS :TEST 'EQ)
                  (RETURN-FROM COMPILE-AND-DUMP))
            (LET ((BD-LIST (AND *BLOCK-HASH-TABLE* (GETHASH NAME *BLOCK-HASH-TABLE*))))
                 (COND
                    ((NULL BD-LIST)
                     (COMPILE-AND-DUMP-1 NAME DEFN KIND))
                    (T (DOLIST (*CURRENT-BLOCK* BD-LIST)
                           (LET* ((LOOKUP (ASSOC NAME (BLOCK-DECL-FN-NAME-MAP *CURRENT-BLOCK*)))
                                  (NEW-NAME (IF (NULL LOOKUP)
                                                NAME
                                                (CDR LOOKUP)))
                                  (IL:SPECVARS (BLOCK-DECL-SPECVARS *CURRENT-BLOCK*))
                                  (IL:LOCALVARS (BLOCK-DECL-LOCALVARS *CURRENT-BLOCK*))
                                  (IL:LOCALFREEVARS (BLOCK-DECL-LOCALFREEVARS *CURRENT-BLOCK*))
                                  (IL:GLOBALVARS (BLOCK-DECL-GLOBALVARS *CURRENT-BLOCK*)))
                                 (DECLARE (SPECIAL IL:SPECVARS IL:LOCALVARS IL:LOCALFREEVARS IL:GLOBALVARS))
                                 (COMPILE-AND-DUMP-1 NEW-NAME DEFN KIND)))))))
           (T (COMPILE-AND-DUMP-1 NAME DEFN KIND)))))
```

```
(DEFUN COMPILE-AND-DUMP-1 (NAME DEFN KIND)
   (WHEN (EQ KIND :FUNCTION)
       (PUSH NAME *PROCESSED-FUNCTIONS*)
       (SETQ *UNKNOWN-FUNCTIONS* (REMOVE NAME *UNKNOWN-FUNCTIONS* :KEY 'CAR)))
   (LET* ((*CURRENT-FUNCTION* NAME)
          (LAP-FN (COMPILE-ONE-LAMBDA NAME DEFN))
          DCODE)
         (WHEN (STREAMP *LAP-STREAM*)
             (PPRINT LAP-FN *LAP-STREAM*)
```

```
            (TERPRI *LAP-STREAM*)
            (TERPRI *LAP-STREAM*))
        (PRINC ".")
        (IL:BLOCK)
        (CONDITION-CASE (SETQ DCODE (D-ASSEM:ASSEMBLE-FUNCTION LAP-FN))
               (ASSEMBLER-ERROR (CONDITION)
                      (FORMAT *ERROR-OUTPUT* "~&~A~%" CONDITION)
                      (PRINC "Aborted.")
                      (TERPRI)
                      (RETURN-FROM COMPILE-AND-DUMP-1 NIL)))
        (PRINC ".")
        (IL:BLOCK)
        (ECASE KIND
            ((:FUNCTION) (FASL:DUMP-FUNCTION-DEF *FASL-HANDLE* DCODE NAME))
            ((:ONE-SHOT) (FASL:DUMP-FUNCALL *FASL-HANDLE* DCODE)))
        (PRINC ".")
        (IL:BLOCK)
        (WHEN (NOT (NULL *LOAD-COMPILED-CODE*))
            (ECASE KIND
                (:FUNCTION
                   (WHEN (AND (EQ :SAVE *LOAD-COMPILED-CODE*)
                              (FBOUNDP NAME)
                              (CONSP (SYMBOL-FUNCTION NAME))
                              (NOT (IL:HASDEF NAME 'IL:FUNCTIONS)))
                       (SETF (GET NAME 'IL:EXPR)
                             (SYMBOL-FUNCTION NAME)))
                   (SETF (SYMBOL-FUNCTION NAME)
                         (D-ASSEM:INTERN-DCODE DCODE)))
                (:ONE-SHOT (LET ((IL:FILEPKGFLG NIL))          ; so that things don't get marked as changed when you execute
                                                               ; the one-shot.
                                (DECLARE (SPECIAL IL:FILEPKGFLG))
                                (FUNCALL (D-ASSEM:INTERN-DCODE DCODE)))))))
```

(DEFUN **COMPILE-ONE-LAMBDA** (NAME DEFN)

;;; Return a LAP function for the given function definition. NAME is the symbol with which the definition will be associated at load time and DEFN is the
;;; LAMBDA-expression to be compiled.

```
   (LET ((*CONTEXT* *NULL-CONTEXT*)
         (*AUTOMATIC-SPECIAL-DECLARATIONS* NIL))
        (LET ((TREE (ALPHA-LAMBDA DEFN :NAME NAME))
              LAP-CODE)
             (UNWIND-PROTECT
                 (SETQ LAP-CODE (PEEPHOLE-OPTIMIZE (GENERATE-CODE (ANNOTATE-TREE (META-EVALUATE TREE)))))
                 (RELEASE-TREE TREE))
             LAP-CODE)))
```

(DEFUN **OPTIMIZE-AND-MACROEXPAND** (FORM &OPTIONAL (ENVIRONMENT *ENVIRONMENT*)
                                              (CONTEXT *CONTEXT*))

;;; Analagous to MACROEXPAND: keep trying OPTIMIZE-AND-MACROEXPAND-1 until it fails to change the form.

```
   (PROG (NEW-FORM CHANGED-P)
         (MULTIPLE-VALUE-SETQ (NEW-FORM CHANGED-P)
                (OPTIMIZE-AND-MACROEXPAND-1 FORM ENVIRONMENT CONTEXT))
         (UNLESS CHANGED-P
             (RETURN (VALUES FORM NIL)))
     LOOP
         (MULTIPLE-VALUE-SETQ (NEW-FORM CHANGED-P)
                (OPTIMIZE-AND-MACROEXPAND-1 NEW-FORM ENVIRONMENT CONTEXT))
         (IF CHANGED-P
             (GO LOOP)
             (RETURN (VALUES NEW-FORM T)))))
```

(DEFUN **OPTIMIZE-AND-MACROEXPAND-1** (FORM &OPTIONAL (ENVIRONMENT *ENVIRONMENT*)
                                                (CONTEXT *CONTEXT*))

;;; If the given form is a list, then look for macros and optimizers defined for its CAR. Return two values like MACROEXPAND-1.

```
   (LET ((*NEW-COMPILER-IS-EXPANDING* T))
        (COND
           ((OR (ATOM FORM)
                (NOT (SYMBOLP (CAR FORM))))
            (VALUES FORM NIL))
           (T ;; Check for compiler optimizers.
              (LET ((OPTIMIZERS (OPTIMIZER-LIST (CAR FORM))))
                   (WHEN (AND (NOT (NULL OPTIMIZERS))
                              (NOT (ENV-FBOUNDP ENVIRONMENT (CAR FORM)
                                          :LEXICAL-ONLY T))
                              (NOT (ENV-INLINE-DISALLOWED ENVIRONMENT (CAR FORM))))
                                                               ; Optimizers cannot apply to lexical functions or macros or to
                                                               ; functions declared NOTINLINE.
                       (DOLIST (OPT-FN OPTIMIZERS)
                           (LET ((RESULT (FUNCALL OPT-FN FORM ENVIRONMENT CONTEXT)))
                                (UNLESS (OR (EQ RESULT 'PASS)
                                            (EQ RESULT 'IL:IGNOREMACRO)
                                            (EQ RESULT FORM))      ; This optimizer fired.
                                    (RETURN-FROM OPTIMIZE-AND-MACROEXPAND-1 (VALUES RESULT T)))))))
              ;; Check for a macro expansion function.

              (MACROEXPAND-1 FORM ENVIRONMENT)))))


(DEFMACRO EXPAND-DEFINER (DEFINER BODY-WITHOUT-COMMENTS &OPTIONAL ENVIRONMENT)
   `(LET ((*NEW-COMPILER-IS-EXPANDING* T))
         (XCL::%EXPAND-DEFINER ,DEFINER ,BODY-WITHOUT-COMMENTS ,ENVIRONMENT)))


(DEFUN PROCESS-FORMS (PROCESS-ENTIRE-FILE)
   (LET ((*DEFERRED-FORMS* NIL)
         (*BLOCKS* NIL)
         (*BLOCK-HASH-TABLE* NIL)
         (*PREPROCESSING-PHASE* PROCESS-ENTIRE-FILE)
         (EOF-VALUE '(NIL))
         FORM)
        (LOOP (IL:SKIPSEPRS *INPUT-STREAM*)
              (WHEN (IL:EOFP *INPUT-STREAM*)
                    (RETURN))
              (SETQ FORM (LET ((*COMPILER-IS-READING* T))
                              (READ *INPUT-STREAM* NIL EOF-VALUE)))
              (WHEN (EQ FORM EOF-VALUE)
                    (RETURN))
              (IF PROCESS-ENTIRE-FILE
                  (LET ((NEW-FORM (CASE (AND (CONSP FORM)
                                             (CAR FORM))
                                      (IL:PRETTYCOMPRINT
                                         (SETQ *INPUT-FILECOMS-VARIABLE* (CADR FORM))
                                         NIL)
                                      (IL:RPAQQ (IF (EQ (SECOND FORM)
                                                        *INPUT-FILECOMS-VARIABLE*)

                                                    ;; Don't remove comments from file coms

                                                    FORM
                                                    (REMOVE-COMMENTS FORM)))
                                      (IL:DEFCLASS

                                         ;; Don't remove comments from LOOPS DEFCLASS forms

                                         FORM)
                                      (IL:DATATYPE

                                         ;; Don't remove comments from record declarations

                                         FORM)
                                      (IL:RECORD

                                         ;; Don't remove comments from record declarations

                                         FORM)
                                      (IL:BLOCKRECORD

                                         ;; Don't remove comments from record declarations

                                         FORM)
                                      (IL:DECLARE\:

                                         ;; Process each form inside this as though it were at top-level

                                         (IL:FOR X IL:IN FORM IL:COLLECT (COND
                                                                             ((NOT (CONSP X))
                                                                              X)
                                                                             (T (CASE (CAR X)
                                                                                    (IL:DEFCLASS X)
                                                                                    (IL:DATATYPE X)
                                                                                    (IL:RECORD X)
                                                                                    (IL:BLOCKRECORD X)
                                                                                    (OTHERWISE (REMOVE-COMMENTS
                                                                                                X)))))))
                                      (OTHERWISE (REMOVE-COMMENTS FORM)))))
                       (SCAN-ONE-FORM NEW-FORM *COMPILE-SCAN-CONTEXT*))
                  (SCAN-ONE-FORM FORM *COMPILE-FILE-CONTEXT*)))
        (WHEN PROCESS-ENTIRE-FILE
            (LET ((*MAKING-SECOND-PASS* T)
                  (*BLOCK-HASH-TABLE* (SET-UP-BLOCK-DECLS *BLOCKS*)))
                 (MAPC #'(LAMBDA (FORM)
                                (SCAN-ONE-FORM FORM *COMPILE-FILE-CONTEXT*))
                       (NREVERSE *DEFERRED-FORMS*))))
        (COMPILER-APPLY PROCESS-OUTSTANDING-LOOSE-FORMS *COMPILE-FILE-CONTEXT*)))


(DEFUN MAYBE-REMOVE-COMMENTS (FORM)
   (COND
      ((EQ 'IL:DEFCLASS (CAR FORM))
       IL:FORM)
      (T (REMOVE-COMMENTS FORM))))
```

```
(DEFUN COMPILE-FILE-SETF-SYMBOL-FUNCTION (COMPILER-CONTEXT FORM)
   (LET ((NAME-FORM (SECOND FORM))
         (FUNCTION-FORM (THIRD FORM)))
        (COND
           ((AND (CONSTANTP NAME-FORM)
                 (SYMBOLP (EVAL NAME-FORM))
                 (FUNCTION-P FUNCTION-FORM))
            (COMPILER-APPLY PROCESS-OUTSTANDING-LOOSE-FORMS COMPILER-CONTEXT)
            (WHEN *EVAL-WHEN-COMPILE* (EVAL FORM))
            (LET ((NAME (SECOND NAME-FORM))
                  (DEFINITION (SECOND FUNCTION-FORM)))
                 (COMPILER-APPLY PROCESS-FUNCTION COMPILER-CONTEXT (FORMAT NIL "~s ~a" (CAR DEFINITION)
                                                                          NAME)
                        NAME DEFINITION)))
           (T (COMPILER-APPLY PROCESS-LOOSE-FORM COMPILER-CONTEXT FORM)))))


(DEFUN COMPILE-FILE-DEFINEQ (COMPILER-CONTEXT FORM)
   (WHEN *EVAL-WHEN-COMPILE* (IL:EVAL FORM))
   (COMPILER-APPLY PROCESS-OUTSTANDING-LOOSE-FORMS COMPILER-CONTEXT)
   (MAPCAR #'(LAMBDA (DEFN)
                    (LET ((REAL-DEFN (IF (NULL (CDDR DEFN))
                                         (SECOND DEFN)
                                         (CONS 'IL:LAMBDA (CDR DEFN)))))
                         (COMPILER-APPLY PROCESS-FUNCTION COMPILER-CONTEXT (FORMAT NIL "~s ~s" (CAR REAL-DEFN)
                                                                                   (CAR DEFN))
                                (CAR DEFN)
                                REAL-DEFN)))
          (CDR FORM)))


(DEFUN COMPILE-FILE-DEFCONSTANT (COMPILER-CONTEXT FORM)
   (COMPILER-APPLY PROCESS-OUTSTANDING-LOOSE-FORMS COMPILER-CONTEXT)
   (DESTRUCTURING-BIND (NAME SYMBOL INITIAL-VALUE &OPTIONAL DOC)
          FORM
          (LET ((VALUE NIL))
               (IF (AND (CONSTANT-EXPRESSION-P INITIAL-VALUE)
                        (VALUE-FOLDABLE-P (SETQ VALUE (EVAL INITIAL-VALUE))))
                   (SETF (CONSTANT-VALUE SYMBOL)
                         VALUE)
                   (ENV-DECLARE-A-GLOBAL (FIND-TOP-ENVIRONMENT *ENVIRONMENT*)
                          SYMBOL)))
          (SCAN-ONE-FORM `(NAMED-PROGN DEFCONSTANT ,SYMBOL (LOCALLY (DECLARE (GLOBAL ,SYMBOL))
                                                                  ,(EXPAND-DEFINER 'DEFCONSTANT (
                                                                                          REMOVE-COMMENTS
                                                                                           FORM)
                                                                          *ENVIRONMENT*)))
                 COMPILER-CONTEXT)))


(DEFUN COMPILE-FILE-DECLARE\: (COMPILER-CONTEXT FORM &OPTIONAL (DOCOPY T))
   (LET ((*EVAL-WHEN-COMPILE* *EVAL-WHEN-COMPILE*))
        (DO ((TAIL (CDR FORM)
                   (CDR TAIL)))
            ((ENDP TAIL))
          (COND
             ((SYMBOLP (CAR TAIL))
              (CASE (CAR TAIL)
                  ((IL:EVAL@LOAD IL:DOEVAL@LOAD IL:DONTEVAL@LOAD) NIL)
                  ((IL:EVAL@LOADWHEN) (POP TAIL))
                  ((IL:EVAL@COMPILE IL:DOEVAL@COMPILE) (SETQ *EVAL-WHEN-COMPILE* T))
                  ((IL:DONTEVAL@COMPILE) (SETQ *EVAL-WHEN-COMPILE* NIL))
                  ((IL:EVAL@COMPILEWHEN) (SETQ *EVAL-WHEN-COMPILE* (IL:EVAL (CAR (SETQ TAIL (CDR TAIL))))))
                  ((IL:COPY IL:DOCOPY) (SETQ DOCOPY T))
                  ((IL:DONTCOPY) (SETQ DOCOPY NIL))
                  ((IL:COPYWHEN) (SETQ DOCOPY (IL:EVAL (CAR (SETQ TAIL (CDR TAIL))))))
                  ((IL:FIRST) )
                  ((IL:NOTFIRST IL:COMPILERVARS) )
                  (OTHERWISE (COMPILER-MESSAGE "Warning: Ignoring unrecognized DECLARE: tag: ~S~%"
                                    (CAR TAIL)))))
             ((EQ 'IL:DECLARE\: (CAR (CAR TAIL)))
              (COMPILER-APPLY IL:DECLARE\: COMPILER-CONTEXT (CAR TAIL)
                     DOCOPY))
             ((EQ 'IL:BLOCK\: (CAR (CAR TAIL)))
              (IF (NULL *PREPROCESSING-PHASE*)
                  (CERROR "Ignore the BLOCK: declaration." "Files with Interlisp BLOCK: declarations must be
                         compiled with :PROCESS-ENTIRE-FILE = T.")
                  (PUSH (CDR (CAR TAIL))
                        *BLOCKS*)))
             (T (WHEN *EVAL-WHEN-COMPILE*
                    (IL:EVAL (CAR TAIL)))
                (WHEN DOCOPY
                    (SCAN-ONE-FORM (CAR TAIL)
                           COMPILER-CONTEXT)))))))
```

```
(DEFUN COMPILE-FILE-DEFINE-FILE-INFO (COMPILER-CONTEXT FORM)
   (LET ((*STANDARD-INPUT* *INPUT-STREAM*)
         IL:FILECREATEDLOC)
        (DECLARE (SPECIAL *STANDARD-INPUT* IL:FILECREATEDLOC))
        (EVAL FORM))
   (COMPILER-APPLY PROCESS-LOOSE-FORM COMPILER-CONTEXT `(LET ((*STANDARD-INPUT* (OPEN "{Null}" :DIRECTION
                                                                                         :OUTPUT))
                                                               IL:FILECREATEDLOC)
                                                              (DECLARE (SPECIAL *STANDARD-INPUT*
                                                                                IL:FILECREATEDLOC))
                                                              ,FORM))
   (COMPILER-APPLY PROCESS-OUTSTANDING-LOOSE-FORMS COMPILER-CONTEXT))


(DEFUN COMPILE-FILE-PACKAGE-FORM (COMPILER-CONTEXT FORM)
   (UNLESS *MAKING-SECOND-PASS* (EVAL FORM))
   (COMPILER-APPLY PROCESS-LOOSE-FORM COMPILER-CONTEXT FORM)
   (COMPILER-APPLY PROCESS-OUTSTANDING-LOOSE-FORMS COMPILER-CONTEXT))


(DEFUN COMPILE-FILE-PROCLAMATION (COMPILER-CONTEXT FORM)
   (DECLARE (SPECIAL IL:GLOBALVARS IL:SPECVARS IL:LOCALVARS))
   (LET ((FORM (EVAL (SECOND FORM)))
         (TOP-ENV (FIND-TOP-ENVIRONMENT *ENVIRONMENT*)))
        (IF (ATOM FORM)
            (CERROR "Ignore the proclamation." "Illegal form in PROCLAIM:~%~S" FORM)
            (CASE (CAR FORM)
                ((SPECIAL) (MAPC #'(LAMBDA (SYMBOL)
                                          (ENV-PROCLAIM-SPECIAL TOP-ENV SYMBOL))
                                 (CDR FORM)))
                ((GLOBAL) (MAPC #'(LAMBDA (SYMBOL)
                                         (ENV-PROCLAIM-GLOBAL TOP-ENV SYMBOL))
                                (CDR FORM)))
                ((IL:GLOBALVARS) (SETQ IL:GLOBALVARS (UNION IL:GLOBALVARS (CDR FORM))))
                ((IL:SPECVARS) (COND
                                  ((CONSP (CDR FORM))
                                   (UNLESS (EQ IL:SPECVARS T)
                                       (SETQ IL:SPECVARS (UNION IL:SPECVARS (CDR FORM)))))
                                  ((EQ (CDR FORM)
                                       T)
                                   (SETQ IL:SPECVARS T)
                                   (SETQ IL:LOCALVARS IL:SYSLOCALVARS))
                                  (T (CERROR "Ignore it" "Illegal SPECVARS proclamation: ~S" FORM))))
                ((IL:LOCALVARS) (COND
                                   ((CONSP (CDR FORM))
                                    (UNLESS (EQ IL:LOCALVARS T)
                                        (SETQ IL:LOCALVARS (UNION IL:LOCALVARS (CDR FORM)))))
                                   ((EQ (CDR FORM)
                                        T)
                                    (SETQ IL:LOCALVARS T)
                                    (SETQ IL:SPECVARS IL:SYSSPECVARS))
                                   (T (CERROR "Ignore it" "Illegal LOCALVARS proclamation: ~S" FORM))))
                ((TYPE FTYPE IL:FUNCTION FUNCTION) NIL)
                ((INLINE) (ENV-ALLOW-INLINES TOP-ENV (CDR FORM)))
                ((NOTINLINE) (ENV-DISALLOW-INLINES TOP-ENV (CDR FORM)))
                ((IGNORE OPTIMIZE) NIL)
                ((DECLARATION) (ENV-ADD-DECLS TOP-ENV (CDR FORM)))
                (OTHERWISE (UNLESS (OR (CL::TYPE-EXPANDER (CAR FORM))
                                       (XCL::DECL-SPECIFIER-P (CAR FORM))
                                       (ENV-DECL-P TOP-ENV (CAR FORM)))
                               (CERROR "Ignore it." "Unknown declaration specifier in PROCLAIM: ~S."
                                      (CAR FORM)))))))
   (COMPILER-APPLY PROCESS-LOOSE-FORM COMPILER-CONTEXT FORM))


(DEFUN COMPILE-FILE-COMPILER-LET (COMPILER-CONTEXT FORM)
   (COMPILER-APPLY PROCESS-OUTSTANDING-LOOSE-FORMS COMPILER-CONTEXT)
   (DESTRUCTURING-BIND (BINDING-LIST &REST INNER-FORMS)
          (CDR FORM)
          (LET (VARS VALS)
               (DOLIST (BINDING BINDING-LIST)
                   (COND
                      ((ATOM BINDING)
                       (PUSH BINDING VARS)
                       (PUSH NIL VALS))
                      ((NULL (CDR BINDING))
                       (PUSH (CAR BINDING)
                             VARS)
                       (PUSH NIL VALS))
                      ((AND (CONSP (CDR BINDING))
                            (NULL (CDDR BINDING)))
                       (PUSH (CAR BINDING)
                             VARS)
                       (PUSH (EVAL (CADR BINDING))
                             VALS))
                      (T (CERROR "Bind the CAR of the binding to NIL" "Bad binding in COMPILER-LET: ~S" BINDING)
                        (PUSH (CAR BINDING)
                              VARS)
                        (PUSH NIL VALS))))
              (PROGV VARS VALS
                  (MAPC #'(LAMBDA (FORM)
                                 (SCAN-ONE-FORM FORM COMPILER-CONTEXT))
                        INNER-FORMS)
                  (COMPILER-APPLY PROCESS-OUTSTANDING-LOOSE-FORMS COMPILER-CONTEXT)))))


(DEFUN COMPILE-FILE-MACROLET (COMPILER-CONTEXT FORM)
   (DESTRUCTURING-BIND (JUNK MACRO-DEFNS &BODY BODY)
          FORM
          (COMPILER-APPLY PROCESS-OUTSTANDING-LOOSE-FORMS COMPILER-CONTEXT)
          (LET ((*ENVIRONMENT* (MAKE-CHILD-ENV *ENVIRONMENT*)))
               (DOLIST (MACRO-DEFN MACRO-DEFNS)
                   (ENV-BIND-FUNCTION *ENVIRONMENT* (CAR MACRO-DEFN)
                          :MACRO
                          (CRACK-DEFMACRO (CONS 'DEFMACRO MACRO-DEFN))))
               (DOLIST (FORM BODY)
                   (SCAN-ONE-FORM FORM COMPILER-CONTEXT))
               (COMPILER-APPLY PROCESS-OUTSTANDING-LOOSE-FORMS COMPILER-CONTEXT))))


(DEFUN COMPILE-FILE-DEFINER (COMPILER-CONTEXT FORM)
   (DESTRUCTURING-BIND (TYPE DEFINER DEFINITION &OPTIONAL ENV)
          (CDR FORM)
          (LET* ((MACRO-DEFINITION (REMOVE-COMMENTS DEFINITION))
                 (NAME (XCL::%DEFINER-NAME DEFINER MACRO-DEFINITION))
                 (BODY (EXPAND-DEFINER DEFINER MACRO-DEFINITION ENV)))
                (IF *COMPILING-DEFINER*
                    (SCAN-ONE-FORM BODY COMPILER-CONTEXT)
                    (PROGN (COMPILER-APPLY PROCESS-OUTSTANDING-LOOSE-FORMS COMPILER-CONTEXT)
                           (LET ((*COMPILING-DEFINER* T)
                                 (*LOOSE-NAME* (FORMAT NIL "~s ~s" DEFINER NAME)))
                                (COMPILING-MESSAGE *LOOSE-NAME*)
                                (SCAN-ONE-FORM BODY COMPILER-CONTEXT)
                                (COMPILER-APPLY PROCESS-OUTSTANDING-LOOSE-FORMS COMPILER-CONTEXT)
                                (DONE-MESSAGE))))))


(DEFUN COMPILE-FILE-NAMED-PROGN (COMPILER-CONTEXT FORM)
   (DESTRUCTURING-BIND (DEFINER-NAME NAME &REST PROGN-FORMS)
          (CDR FORM)
          (IF *COMPILING-DEFINER*
              (MAPC #'(LAMBDA (FORM)
                             (SCAN-ONE-FORM FORM COMPILER-CONTEXT))
                    PROGN-FORMS)
              (PROGN (COMPILER-APPLY PROCESS-OUTSTANDING-LOOSE-FORMS COMPILER-CONTEXT)
                     (LET ((*COMPILING-DEFINER* T)
                           (*LOOSE-NAME* (FORMAT NIL "~s ~s" DEFINER-NAME NAME)))
                          (COMPILING-MESSAGE *LOOSE-NAME*)
                          (MAPC #'(LAMBDA (FORM)
                                         (SCAN-ONE-FORM FORM COMPILER-CONTEXT))
                                PROGN-FORMS)
                          (COMPILER-APPLY PROCESS-OUTSTANDING-LOOSE-FORMS COMPILER-CONTEXT)
                          (DONE-MESSAGE))))))


(DEFUN COMPILE-FILE-OUTSTANDING-LOOSE-FORMS (COMPILER-CONTEXT)
   (WHEN (NOT (NULL *OUTSTANDING-LOOSE-FORMS*))
       (IF *COMPILING-DEFINER*
           (COMPILE-AND-DUMP *LOOSE-NAME* `(LAMBDA NIL ,@(REVERSE *OUTSTANDING-LOOSE-FORMS*))
                  :ONE-SHOT)
           (LET ((NAME (FORMAT NIL "~&~D top-level form~:P" (LIST-LENGTH *OUTSTANDING-LOOSE-FORMS*))))
                (COMPILING-MESSAGE NAME)
                (COMPILE-AND-DUMP NAME `(LAMBDA NIL ,@(REVERSE *OUTSTANDING-LOOSE-FORMS*))
                       :ONE-SHOT)
                (DONE-MESSAGE)))
       (SETQ *OUTSTANDING-LOOSE-FORMS* NIL)))


(DEFUN COMPILE-FILE-LOOSE-FORM (COMPILER-CONTEXT FORM)
   (WHEN *EVAL-WHEN-COMPILE* (EVAL FORM))
   (PUSH FORM *OUTSTANDING-LOOSE-FORMS*))


(DEFUN COMPILE-FILE-PROCESS-FUNCTION (COMPILER-CONTEXT MESSAGE NAME DEFINITION)
   (IF *COMPILING-DEFINER*
       (COMPILE-AND-DUMP NAME DEFINITION :FUNCTION)
       (PROGN (COMPILING-MESSAGE MESSAGE)
              (COMPILE-AND-DUMP NAME DEFINITION :FUNCTION)
              (DONE-MESSAGE)))
   NAME)


(DEFUN CRACK-DEFMACRO (FORM)
```

;;; FORM is a call to DEFMACRO. Return two values: the LAMBDA-expression representing the expansion function for the macro and the
;;; documentation string, if present.

```
   (LET ((NAME (SECOND FORM))
         (ARG-LIST (THIRD FORM))
         (BODY (CDDDR FORM))
         (WHOLE (GENSYM))
         (ENV-VAR (GENSYM)))
        (MULTIPLE-VALUE-BIND (CODE DECLS DOC)
            (IL:PARSE-DEFMACRO ARG-LIST WHOLE BODY NAME *ENVIRONMENT* :ENVIRONMENT ENV-VAR)
          (VALUES `(LAMBDA (,WHOLE ,ENV-VAR)
                          ,@DECLS
                          (BLOCK ,NAME ,CODE))
                 DOC))))
```

(DEFUN **ESTABLISH-MACRO-IN-COMPILER** (NAME EXPN-FN)

;;; Arrange for the symbol NAME to refer to a macro with the given expansion-function EXPN-FN within this compilation.

```
   (ENV-BIND-FUNCTION (FIND-TOP-ENVIRONMENT *ENVIRONMENT*)
          NAME :MACRO EXPN-FN))
```

;; Support for :Process-Entire-File

(DEFVAR ***DEFERRED-FORMS*** NIL

```
   "A list onto which most forms will be pushed if we are preprocessing an Interlisp-format file.  After the
   first pass through the file is done, and all macros and other eval-when(compile) forms have been processed, a
   second pass will be made down this list to actually compile the forms.")
```

(DEFVAR ***MAKING-SECOND-PASS*** NIL

;;; Bound to T during second pass over saved forms; used for :Process-Entire-File option to compile-file.

```
   )
```

(DEFVAR ***PREPROCESSING-PHASE*** NIL

;;; Bound to T during the preprocessing phase so that inferiors can tell.

```
   )
```

(DEFUN **COMPILE-SCAN-DECLARE\:** (COMPILER-CONTEXT FORM &OPTIONAL (DOCOPY T)
                                                (DOFIRST NIL))

```
   (LET ((FIRST-FORMS NIL)
         (IL:DFNFLG IL:DFNFLG)
         (*EVAL-WHEN-COMPILE* *EVAL-WHEN-COMPILE*))
        (DO ((TAIL (CDR FORM)
                   (CDR TAIL)))
            ((ENDP TAIL)
             (WHEN FIRST-FORMS (MERGE-FIRST-FORMS FIRST-FORMS)))
          (COND
             ((SYMBOLP (CAR TAIL))
              (CASE (CAR TAIL)
                  ((IL:EVAL@LOAD IL:DOEVAL@LOAD IL:DONTEVAL@LOAD) NIL)
                  ((IL:EVAL@LOADWHEN) (POP TAIL))
                  ((IL:EVAL@COMPILE IL:DOEVAL@COMPILE) (SETQ *EVAL-WHEN-COMPILE* T))
                  ((IL:DONTEVAL@COMPILE) (SETQ *EVAL-WHEN-COMPILE* NIL))
                  ((IL:EVAL@COMPILEWHEN) (SETQ *EVAL-WHEN-COMPILE* (IL:EVAL (CAR (SETQ TAIL (CDR TAIL))))))
                  ((IL:COPY IL:DOCOPY) (SETQ DOCOPY T))
                  ((IL:DONTCOPY) (SETQ DOCOPY NIL))
                  ((IL:COPYWHEN) (SETQ DOCOPY (IL:EVAL (CAR (SETQ TAIL (CDR TAIL))))))
                  ((IL:FIRST) (SETQ DOFIRST T))
                  ((IL:NOTFIRST) (SETQ DOFIRST NIL))
                  ((IL:COMPILERVARS) (SETQ IL:DFNFLG T))
                  (OTHERWISE (COMPILER-MESSAGE "Warning: Ignoring unrecognized DECLARE: tag: ~S~%"
                                    (CAR TAIL)))))
             ((EQ 'IL:DECLARE\: (CAR (CAR TAIL)))
              (COMPILER-APPLY IL:DECLARE\: COMPILER-CONTEXT (CAR TAIL)
                     DOCOPY DOFIRST))
             ((EQ 'IL:BLOCK\: (CAR (CAR TAIL)))
              (PUSH (CDR (CAR TAIL))
                    *BLOCKS*))
             (T (WHEN *EVAL-WHEN-COMPILE*
                    (IL:EVAL (CAR TAIL)))
                (WHEN DOCOPY
                    (IF DOFIRST
                        (LET ((*DEFERRED-FORMS* NIL))
                             (SCAN-ONE-FORM (CAR TAIL)
                                    COMPILER-CONTEXT)
                             (SETQ FIRST-FORMS (APPEND FIRST-FORMS *DEFERRED-FORMS*)))
                        (SCAN-ONE-FORM (CAR TAIL)
                              COMPILER-CONTEXT)))))))


(DEFUN COMPILE-SCAN-DEFINE-FILE-INFO (COMPILER-CONTEXT FORM)
   (LET ((*STANDARD-INPUT* *INPUT-STREAM*)
         IL:FILECREATEDLOC)
        (DECLARE (SPECIAL *STANDARD-INPUT* IL:FILECREATEDLOC))
        (EVAL FORM))
   (COMPILER-APPLY PROCESS-LOOSE-FORM COMPILER-CONTEXT FORM)
   (COMPILER-APPLY PROCESS-OUTSTANDING-LOOSE-FORMS COMPILER-CONTEXT))


(DEFUN COMPILE-SCAN-MACROLET (COMPILER-CONTEXT FORM)
   (DESTRUCTURING-BIND (JUNK MACRO-DEFNS &BODY BODY)
          FORM
          (LET (OUTER-DEFERRED-FORMS)
               (LET ((*DEFERRED-FORMS* NIL)
                     (*ENVIRONMENT* (MAKE-CHILD-ENV *ENVIRONMENT*)))
                    (DOLIST (MACRO-DEFN MACRO-DEFNS)
                        (ENV-BIND-FUNCTION *ENVIRONMENT* (CAR MACRO-DEFN)
                               :MACRO
                               (CRACK-DEFMACRO (CONS 'DEFMACRO MACRO-DEFN))))
                    (DOLIST (FORM BODY)
                        (SCAN-ONE-FORM FORM COMPILER-CONTEXT))
                    (SETQ OUTER-DEFERRED-FORMS *DEFERRED-FORMS*))
               (WHEN (NOT (NULL OUTER-DEFERRED-FORMS))
                   (COMPILER-APPLY PROCESS-LOOSE-FORM COMPILER-CONTEXT `(MACROLET ,MACRO-DEFNS
                                                                                  ,@(REVERSE OUTER-DEFERRED-FORMS)
                                                                                  ))))))


(DEFUN COMPILE-SCAN-DEFINER (COMPILER-CONTEXT FORM)
   (DESTRUCTURING-BIND (TYPE DEFINER DEFINITION &OPTIONAL ENV)
          (CDR FORM)
          (COMPILER-APPLY PROCESS-LOOSE-FORM COMPILER-CONTEXT (LET* ((*DEFERRED-FORMS* NIL)
                                                                       (MACRO-DEFINITION (REMOVE-COMMENTS
                                                                                          DEFINITION))
                                                                       (NAME (XCL::%DEFINER-NAME DEFINER
                                                                                  MACRO-DEFINITION))
                                                                       (BODY (EXPAND-DEFINER DEFINER
                                                                                  MACRO-DEFINITION ENV)))
                                                                      (SCAN-ONE-FORM BODY COMPILER-CONTEXT)
                                                                      `(NAMED-PROGN ,DEFINER ,NAME
                                                                              ,@(NREVERSE *DEFERRED-FORMS*))))))


(DEFUN COMPILE-SCAN-LOOSE-FORM (COMPILER-CONTEXT FORM)
   (WHEN *EVAL-WHEN-COMPILE* (EVAL FORM))
   (PUSH FORM *DEFERRED-FORMS*))


(DEFUN COMPILE-SCAN-OUTSTANDING-LOOSE-FORMS (COMPILER-CONTEXT)
   NIL)


(DEFUN MERGE-FIRST-FORMS (FORMS)
   (DO* ((TAIL *DEFERRED-FORMS* (CDR TAIL))
         (NEW-TAIL (CDR TAIL)
                (CDR TAIL)))
        ((ENDP TAIL)
         (IF (NULL NEW-TAIL)
             (NCONC *DEFERRED-FORMS* FORMS))
         NIL)
      (WHEN (EQL (CAAR NEW-TAIL)
                 'IL:FILECREATED)
          (SETF (CDR TAIL)
                FORMS)
          (SETF (CDR (LAST FORMS))
                NEW-TAIL)
          (RETURN))))

;; for compiling definers


(DEFVAR *LAP-FLG* NIL)


(DEFVAR *AUTOMATIC-SPECIAL-DECLARATIONS*)


(DEFUN COMPILE (NAME &OPTIONAL DEFN &KEY LAP)
   (WHEN (NULL DEFN)
       (IL:VIRGINFN NAME T)
       (SETQ DEFN (IL:GETD NAME))
       (TYPECASE DEFN
           (CONS NIL)
           ((OR NULL IL:COMPILED-CLOSURE)
               (IF (NULL DEFN)
                   (FORMAT T "There's nothing in the function cell of ~S.~%" NAME)
                   (FORMAT T "~S is already compiled.~%" NAME))
               (WHEN (AND (IL:HASDEF NAME 'IL:FUNCTIONS)
                          (Y-OR-N-P "Shall I use the FUNCTIONS definition? "))
                   (RETURN-FROM COMPILE (COMPILE-DEFINER NAME 'IL:FUNCTIONS :LAP LAP)))
               (WHEN (AND (GET NAME 'IL:EXPR)
                          (Y-OR-N-P "Shall I use the definition on the EXPR property? " NAME))
                   (RETURN-FROM COMPILE (COMPILE NAME (GET NAME 'IL:EXPR)
                                                 :LAP LAP)))
               (RETURN-FROM COMPILE))
            (OTHERWISE
               (FORMAT T "There's something funny in the function cell of ~S.~%I'm not going any further.~%" NAME
                      )
               (RETURN-FROM COMPILE))))
   (LET* ((*ENVIRONMENT* (MAKE-ENV :PARENT T :TARGET-ARCHITECTURE *HOST-ARCHITECTURE*))
          (IL:SPECVARS IL:SPECVARS)
          (IL:LOCALVARS IL:LOCALVARS)
          (IL:LOCALFREEVARS NIL)
          (IL:GLOBALVARS IL:GLOBALVARS)
          (*CONSTANTS-HASH-TABLE* (MAKE-HASH-TABLE))
          (*PROCESSED-FUNCTIONS* (LIST NAME))
          (*UNKNOWN-FUNCTIONS* NIL)
          (*CURRENT-FUNCTION* NAME)
          (*INPUT-STREAM* NIL)
          (*LAP-FLG* LAP)                                        ; FXAR-111
          (COMPILED-DEFN (RAW-COMPILE NAME DEFN)))
         (DECLARE (SPECIAL IL:SPECVARS IL:LOCALVARS IL:LOCALFREEVARS IL:GLOBALVARS))
         (WARN-ABOUT-UNKNOWN-FUNCTIONS)
         (IF (NULL NAME)
             COMPILED-DEFN
             (PROGN (WHEN (AND (NOT (IL:HASDEF NAME 'IL:FUNCTIONS))
                               (CONSP (IL:GETD NAME)))
                          (SETF (GET NAME 'IL:EXPR)
                                (IL:GETD NAME)))
                    (SETF (SYMBOL-FUNCTION NAME)
                          COMPILED-DEFN)
                    NAME))))


(DEFUN COMPILE-DEFINER (NAME TYPE &KEY LAP)
   (LET ((*ENVIRONMENT* (MAKE-ENV :PARENT T))
         (*OUTSTANDING-LOOSE-FORMS* NIL)
         (*EVAL-WHEN-COMPILE* NIL))
        (COMPILE-FORM (IL:GETDEF NAME TYPE NIL '(IL:NOCOPY T)))
        NAME))


(DEFUN COMPILE-FORM (FORM &KEY LAP)
   (LET ((*CONTEXT* (MAKE-CONTEXT :TOP-LEVEL-P T :VALUES-USED 0))
         (*ENVIRONMENT* (MAKE-ENV :PARENT T))
         (*CONSTANTS-HASH-TABLE* (MAKE-HASH-TABLE))
         (*PROCESSED-FUNCTIONS* NIL)
         (*UNKNOWN-FUNCTIONS* NIL)
         (*OUTSTANDING-LOOSE-FORMS* NIL)
         (*LAP-FLG* LAP)
         (IL:SPECVARS IL:SPECVARS)
         (IL:LOCALVARS IL:LOCALVARS)
         (IL:LOCALFREEVARS NIL)
         (IL:GLOBALVARS IL:GLOBALVARS)
         VALUE)
        (DECLARE (SPECIAL IL:SPECVARS IL:LOCALVARS IL:LOCALFREEVARS IL:GLOBALVARS))
        (SETQ VALUE (MULTIPLE-VALUE-LIST (SCAN-ONE-FORM (REMOVE-COMMENTS FORM)
                                                *COMPILE-DEFINER-CONTEXT*)))
        (IF *OUTSTANDING-LOOSE-FORMS*
            (SETQ VALUE (MULTIPLE-VALUE-LIST (COMPILER-APPLY PROCESS-OUTSTANDING-LOOSE-FORMS
                                                    *COMPILE-DEFINER-CONTEXT*))))
        (WARN-ABOUT-UNKNOWN-FUNCTIONS)
        (VALUES-LIST VALUE)))


(DEFUN RAW-COMPILE (NAME DEFINITION)
   (LET* ((*CURRENT-FUNCTION* NAME)
          (LAP-FN (COMPILE-ONE-LAMBDA NAME DEFINITION))
          COMPILED-DEFN)
         (WHEN (NOT (NULL *LAP-FLG*))
             (PPRINT LAP-FN (IF (STREAMP *LAP-FLG*)
                                *LAP-FLG*
                                *STANDARD-OUTPUT*)))
         (CONDITION-CASE (SETQ COMPILED-DEFN (LET ((DCODE (D-ASSEM:ASSEMBLE-FUNCTION LAP-FN)))
                                                  (UNWIND-PROTECT
                                                      (D-ASSEM:INTERN-DCODE DCODE)
                                                      (D-ASSEM:RELEASE-DCODE DCODE))))
                (ASSEMBLER-ERROR (CONDITION)
                       (FORMAT *ERROR-OUTPUT* "~&~A~%" CONDITION)
                       (RETURN-FROM RAW-COMPILE NIL)))
        COMPILED-DEFN))
```

```
(DEFUN COMPILE-DEFINER-DEFINER (COMPILER-CONTEXT FORM)
   (DESTRUCTURING-BIND (TYPE DEFINER DEFINITION &OPTIONAL ENV)
          (CDR FORM)
          (LET* ((MACRO-DEFINITION (REMOVE-COMMENTS DEFINITION))
                 (NAME (XCL::%DEFINER-NAME DEFINER MACRO-DEFINITION))
                 (BODY (EXPAND-DEFINER DEFINER MACRO-DEFINITION ENV)))
                (IF *COMPILING-DEFINER*
                    (SCAN-ONE-FORM BODY COMPILER-CONTEXT)
                    (PROGN (COMPILER-APPLY PROCESS-OUTSTANDING-LOOSE-FORMS COMPILER-CONTEXT)
                           (LET ((*COMPILING-DEFINER* T)
                                 (*LOOSE-NAME* (FORMAT NIL "~s ~s" DEFINER NAME))
                                 VALUE)
                                (SETQ VALUE (SCAN-ONE-FORM BODY COMPILER-CONTEXT))
                                (IF *OUTSTANDING-LOOSE-FORMS*
                                    (SETQ VALUE (COMPILER-APPLY PROCESS-OUTSTANDING-LOOSE-FORMS
                                                       COMPILER-CONTEXT)))
                                VALUE))))))

(DEFUN COMPILE-DEFINER-NAMED-PROGN (COMPILER-CONTEXT FORM)
   (DESTRUCTURING-BIND (DEFINER-NAME NAME &REST PROGN-FORMS)
          (CDR FORM)
          (IF *COMPILING-DEFINER*
              (MAPC #'(LAMBDA (FORM)
                             (SCAN-ONE-FORM FORM COMPILER-CONTEXT))
                    PROGN-FORMS)
              (PROGN (COMPILER-APPLY PROCESS-OUTSTANDING-LOOSE-FORMS COMPILER-CONTEXT)
                     (LET ((*COMPILING-DEFINER* T)
                           (*LOOSE-NAME* (FORMAT NIL "~s ~s" DEFINER-NAME NAME)))
                          (MAPC #'(LAMBDA (FORM)
                                         (SCAN-ONE-FORM FORM COMPILER-CONTEXT))
                                PROGN-FORMS)
                          (COMPILER-APPLY PROCESS-OUTSTANDING-LOOSE-FORMS COMPILER-CONTEXT))))
          NAME))

(DEFUN COMPILE-DEFINER-PROCESS-FUNCTION (COMPILER-CONTEXT MESSAGE NAME DEFINITION)
   (PUSH NAME *PROCESSED-FUNCTIONS*)
   (SETQ *UNKNOWN-FUNCTIONS* (REMOVE NAME *UNKNOWN-FUNCTIONS* :KEY 'CAR))
   (LET ((*ENVIRONMENT* (COPY-ENV *ENVIRONMENT*))
         COMPILED-DEFN)

        ;; The resulting function is defined locally, so we have to compile for the host architecture rather than the target architecture:

        (SETF (ENV-TARGET-ARCHITECTURE *ENVIRONMENT*)
              *HOST-ARCHITECTURE*)
        (SETQ COMPILED-DEFN (RAW-COMPILE NAME DEFINITION))
        (WHEN (AND (NOT (IL:HASDEF NAME 'IL:FUNCTIONS))
                   (CONSP (IL:GETD NAME)))
            (SETF (GET NAME 'IL:EXPR)
                  (IL:GETD NAME)))
        (SETF (SYMBOL-FUNCTION NAME)
              COMPILED-DEFN)
        NAME))

(DEFUN COMPILE-DEFINER-OUTSTANDING-LOOSE-FORMS (COMPILER-CONTEXT)

   ;; Compile any outstanding loose forms in the context of a structure definition being compiled

   (WHEN (NOT (NULL *OUTSTANDING-LOOSE-FORMS*))
       (LET* ((*ENVIRONMENT* (COPY-ENV *ENVIRONMENT*))
              COMPILED-DEFN)

             ;; The resulting function is executed locally, so have to compile for the host architecture rather than the target architecture:

             (SETF (ENV-TARGET-ARCHITECTURE *ENVIRONMENT*)
                   *HOST-ARCHITECTURE*)
             (SETQ COMPILED-DEFN (RAW-COMPILE *LOOSE-NAME* `(LAMBDA NIL ,@(REVERSE *OUTSTANDING-LOOSE-FORMS*)))
                   )
             (SETQ *OUTSTANDING-LOOSE-FORMS* NIL)
             (FUNCALL COMPILED-DEFN))))

;; Arrange for correct compiler to be used.

(IL:PUTPROPS IL:XCLC-TOP-LEVEL IL:FILETYPE :COMPILE-FILE)

;; Arrange for the correct makefile environment

(IL:PUTPROPS IL:XCLC-TOP-LEVEL IL:MAKEFILE-ENVIRONMENT (:READTABLE "XCL" :PACKAGE (DEFPACKAGE "COMPILER"
                                                                                      (:USE "LISP" "XCL"))))

(IL:PUTPROPS IL:XCLC-TOP-LEVEL IL:COPYRIGHT ("Venue & Xerox Corporation" 1986 1987 1988 1989 1990 1991 1994 2020)
)
```

{MEDLEY}<sources>XCLC-TOP-LEVEL.;1 28-Jun-2024 18:34:03

-- Listed on 30-Jun-2024 13:16:31 --

FUNCTION INDEX

VARIABLE INDEX

STRUCTURE INDEX

MACRO INDEX

PROPERTY INDEX

CONSTANT INDEX