

File created: 12-Oct-93 22:28:39 {Pele:mv:envos}<LispCore>Sources>CLTL2>WRAPPERS.;1

changes to: (IL:FUNCTIONS HAS-CALLS)

previous date: 5-Oct-92 23:22:18 {DSK}<mo>usr>users>sybalsky>cltl2>sources>WRAPPERS.;1

Read Table: XCL

Package: SYSTEM

Format: XCCS

; Copyright (c) 1987, 1988, 1990, 1991, 1992, 1993 by Venue & Xerox Corporation. All rights reserved.

(IL:RPAQQ **IL:WRAPPERSCOMS**

```
((IL:FUNCTIONS COMPILED-FUNCTION-ARGLIST COMPILED-FUNCTION-DEBUGGING-INFO COMPILED-FUNCTION-INTERLISP?
  FUNCTION-WRAPPER-INFO CLEAN-UP-CL-ARGLIST GET-STORED-ARGLIST NAMED-FUNCTION-WRAPPER-INFO
  PARSE-CL-ARGLIST)
 (IL:FUNCTIONS HAS-CALLS CHANGE-CALLS CHANGE-CALLS-IN-CCODE CHANGE-CALLS-IN-LAMBDA ADD-CHANGED-CALL
  %WITH-CHANGED-CALLS RESTORE-CALLS)
```

;;; Support for function name mapping

```
(IL:FUNCTIONS XCL::NAME-OF-EXECUTABLE XCL::COMMON-LISP-MAPPER)
(IL:VARIABLES XCL::*FUNCTION-NAME-MAPPERS*)
(IL:FNS IL:VIRGINFN CONSTRUCT-MIDDLE-MAN)
(IL:PROP IL:PROPTYPE IL:NAMESCHANGED)
```

;; Arrange for the proper compiler and package/readtable.

```
(IL:PROP (IL:FILETYPE IL:MAKEFILE-ENVIRONMENT)
  IL:WRAPPERS)
(IL:DECLARE\ : IL:DOEVAL@COMPILE IL:DONTCOPY (IL:FILES (IL:LOADCOMP
  IL:ACODE))
 (IL:DECLARE\ : IL:DONTEVAL@LOAD IL:DOEVAL@COMPILE IL:DONTCOPY IL:COMPILEVAR (IL:ADDVARS (IL:NLAMA
  (IL:NLAML)
  (IL:LAMA
  CONSTRUCT-MIDDLE-MAN
  ))))
```

(DEFUN **COMPILED-FUNCTION-ARGLIST** (FN &KEY INTERLISP?)

```
(LET ((DEBUGGING-INFO (COMPILED-FUNCTION-DEBUGGING-INFO FN)))
  (COND
```

(DEBUGGING-INFO ; Oh, good. Its argument list is easy to get.

```
(IF INTERLISP?
  (IL:|for| X IL:|in| (CAR DEBUGGING-INFO) IL:|join| (COND
    ((STRINGP X)
     (LIST (IL:MKATOM X)))
    ((EQ X '&OPTIONAL)
     NIL)
    (T (LIST X))))
  (COPY-TREE (CAR DEBUGGING-INFO))))
```

(T ; Rats. We have to go to some trouble.

```
(IL:\CCODEARGLIST (IL:|fetch| (IL:COMPILED-CLOSURE IL:FNHEADER) IL:|of| FN))))
```

(DEFUN **COMPILED-FUNCTION-DEBUGGING-INFO** (FN)

;;; Given a compiled-function object, extract the debugging-info list from it. If it's ByteCompiled, it won't have such a list and we should return NIL. We can tell if there is such a list by the length allowed for the local name table. If there's a multiple of a quadword there, it's a name table. Otherwise, it should be exactly one cell long and contain a pointer to the debugging-info list.

```
(LET* ((FNHEADER (IL:|fetch| (IL:COMPILED-CLOSURE IL:FNHEADER) IL:|of| FN))
      (START-PC (IF (IL:|fetch| (IL:FNHEADER IL:NATIVE) IL:|of| FNHEADER)
        (- (IL:|fetch| (IL:FNHEADER IL:STARTPC) IL:|of| FNHEADER)
          4)
        (IL:|fetch| (IL:FNHEADER IL:STARTPC) IL:|of| FNHEADER)))
      (NAME-TABLE-WORDS (LET ((SIZE (IL:|fetch| (IL:FNHEADER IL:NTSIZE) IL:|of| FNHEADER)))
        (IF (ZEROP SIZE)
            IL:WORDSPERQUAD
            (* SIZE 2))))
      (PAST-NAME-TABLE-IN-WORDS (+ (IL:|fetch| (IL:FNHEADER IL:OVERHEADWORDS) IL:|of| FN)
        NAME-TABLE-WORDS)))
  (AND (= (- START-PC (* IL:BYTESPERWORD PAST-NAME-TABLE-IN-WORDS))
    IL:BYTESPERCELL)
    ;; It's got a debugging-info list.
    (IL:\GETBASEPTR FNHEADER PAST-NAME-TABLE-IN-WORDS))))
```

(DEFUN **COMPILED-FUNCTION-INTERLISP?** (FN)

;;; Given a compiled-function, return true if and only if the function is an Interlisp one.

```
(LET ((DEBUGGING-INFO (COMPILED-FUNCTION-DEBUGGING-INFO FN))
      (OR (MEMBER (IL:ARGTYPE FN)
        '(1 3))
        ; NLambda's are always Interlisp
```

```

(NULL DEBUGGING-INFO) ; ByteCompiled code is always Interlisp.
(GETF (CDR DEBUGGING-INFO)
 :INTERLISP) ; PavCompiled Interlisp code should have this marker in it.

)))

```

```

(DEFUN FUNCTION-WRAPPER-INFO (WRAPPED-FN FN-TO-CALL)
 (LET* ((NAME (AND (SYMBOLP WRAPPED-FN)
 WRAPPED-FN))
 (DEFN (IF NAME
 (IL:GETD NAME)
 WRAPPED-FN)))
 (NAMED-FUNCTION-WRAPPER-INFO NAME DEFN FN-TO-CALL)))

```

```

(DEFUN CLEAN-UP-CL-ARGLIST (ARG-LIST)
 (IL:bind| (STATE IL:_ :REQUIRED) IL:|for| PARAM IL:|in| ARG-LIST
 IL:|collect| (COND
 ((MEMBER PARAM '(&OPTIONAL &REST &KEY &ALLOW-OTHER-KEYS))
 (SETQ STATE PARAM)
 PARAM)
 ((CONSP PARAM)
 (CASE STATE
 (&OPTIONAL (FIRST PARAM))
 (&KEY (IF (CONSP (FIRST PARAM))
 (FIRST (FIRST PARAM))
 (INTERN (STRING (FIRST PARAM))
 "KEYWORD"))))
 (OTHERWISE
 (WARN "Illegal form in argument-list: ~S" PARAM)
 'USER::%LOSE%)))
 ((EQ STATE '&KEY)
 (INTERN (STRING PARAM)
 "KEYWORD"))
 (T PARAM))))

```

```

(DEFUN GET-STORED-ARGLIST (NAME)

```

;;; The IL:ARGNAMES property is either the argument list itself or a list of the form (NIL arglist-1 . arglist-2) where arglist-1 is semantically void and arglist-2 is interesting. Since NIL is not a legal argument list, we can tell the cases apart. Ugh.

```

(LET ((ARGNAMES (GET NAME 'IL:ARGNAMES))
 (AND ARGNAMES (COND
 ((ATOM ARGNAMES)
 (ERROR "Illegal ARGNAMES property for ~S" NAME))
 ((NULL (CAR ARGNAMES)) ; It's the fancy case.
 (CDDR ARGNAMES))
 (T ; It's the simple case.
 ARGNAMES)))))

```

```

(DEFUN NAMED-FUNCTION-WRAPPER-INFO (NAME DEFN FN-TO-CALL)

```

```

(LET
 ((STORED-ARGLIST (AND NAME (GET-STORED-ARGLIST NAME)))
 (ETYPESCASE DEFN
 (NULL ; It's an undefined function.
 (ASSERT (NOT (NULL NAME))
 NIL "Null definition passed to SI::FUNCTION-WRAPPER-INFO")
 (VALUES 'LAMBDA '(&REST XCL:ARGLIST)
 '(ERROR 'UNDEFINED-FUNCTION :NAME (CONS ',NAME XCL:ARGLIST))))
 (CONS ; It's an interpreted function.
 (ECASE (CAR DEFN)
 ((IL:LAMBDA)
 (ETYPESCASE (CADR DEFN)
 (LIST ; Lambda spread
 (VALUES 'IL:LAMBDA (OR STORED-ARGLIST (CADR DEFN))
 '(FUNCALL ',FN-TO-CALL ,@(OR STORED-ARGLIST (CADR DEFN))))
 (SYMBOL ; Lambda no-spread
 (VALUES
 'IL:LAMBDA
 (OR STORED-ARGLIST (CADR DEFN))
 '(APPLY ',FN-TO-CALL
 , (IF (CONSP STORED-ARGLIST)
 '(LIST ,@STORED-ARGLIST)
 '(IL:FOR $FWI$ IL:TO , (OR STORED-ARGLIST (CADR DEFN))
 IL:COLLECT (IL:ARG , (OR STORED-ARGLIST (CADR DEFN))
 $FWI$))))))))
 ((IL:NLAMBDA) (ETYPESCASE (CADR DEFN)
 (LIST ; NLambda spread
 (VALUES 'IL:NLAMBDA (OR STORED-ARGLIST (CADR DEFN))
 '(FUNCALL ',FN-TO-CALL ,@(OR STORED-ARGLIST (CADR DEFN))))
 (SYMBOL ; NLambda no-spread
 (VALUES 'IL:NLAMBDA (OR STORED-ARGLIST (CADR DEFN))
 '(FUNCALL ',FN-TO-CALL , (IF (CONSP STORED-ARGLIST)
 '(LIST ,@STORED-ARGLIST)

```

```

((LAMBDA (VALUES 'LAMBDA (CLEAN-UP-CL-ARGLIST (CADR DEFN))
                \ (APPLY ' ,FN-TO-CALL XCL:ARGLIST))))))
(OR STORED-ARGLIST (CADR DEFN))))))
( (LAMBDA) (VALUES 'LAMBDA (CLEAN-UP-CL-ARGLIST (CADR DEFN))
                  \ (APPLY ' ,FN-TO-CALL XCL:ARGLIST))))))
( (COMPILED-FUNCTION
  (IF (NOT (COMPILED-FUNCTION-INTERLISP? DEFN)) ; It's compiled.
      (VALUES 'LAMBDA (COMPILED-FUNCTION-ARGLIST DEFN) ; Common Lisp function.
              \ (APPLY ' ,FN-TO-CALL XCL:ARGLIST))
      (ECASE (IL:ARGTYPE DEFN)
        (0 ; Lambda spread function.
          (LET ((ARGLIST (OR STORED-ARGLIST (COMPILED-FUNCTION-ARGLIST DEFN :INTERLISP? T)))
                (VALUES 'IL:LAMBDA ARGLIST \ (FUNCALL ' ,FN-TO-CALL ,@ARGLIST))))
            (1 ; NLambda spread function.
              (LET ((ARGLIST (OR STORED-ARGLIST (COMPILED-FUNCTION-ARGLIST DEFN :INTERLISP? T)))
                    (VALUES 'IL:NLAMBDA ARGLIST \ (FUNCALL ' ,FN-TO-CALL ,@ARGLIST))))
                (2 ; Lambda no-spread function.
                  (IF (SYMBOLP STORED-ARGLIST)
                      (VALUES 'IL:LAMBDA 'IL:U \ (APPLY ' ,FN-TO-CALL
                                                         (IL:FOR $FWI$ IL:TO , (OR STORED-ARGLIST 'IL:U)
                                                         IL:COLLECT (IL:ARG , (OR STORED-ARGLIST
                                                         'IL:U)
                                                         $FWI$))))
                      (VALUES 'IL:LAMBDA STORED-ARGLIST \ (FUNCALL ' ,FN-TO-CALL ,@STORED-ARGLIST))))
                    ; NLambda no-spread function.
                (3
                  ;; Its arglist may be a symbol, or NIL, or IL:U. COMPILED-FUNCTION-ARGLIST will return a symbol in this case.
                  (LET ((ARGLIST (OR (AND (IL:NEQ STORED-ARGLIST 'IL:U)
                                         STORED-ARGLIST)
                                     (COMPILED-FUNCTION-ARGLIST DEFN :INTERLISP? T))))
                      (COND
                        ((SYMBOLP ARGLIST)
                         (VALUES 'IL:NLAMBDA ARGLIST \ (APPLY ' ,FN-TO-CALL (IL:MKLIST ,ARGLIST))))
                        (T (VALUES 'IL:NLAMBDA ARGLIST \ (FUNCALL ' ,FN-TO-CALL ,ARGLIST))))))))))

```

```

(DEFUN PARSE-CL-ARGLIST (ARG-LIST)
  (LET ((REQUIRED NIL)
        (OPTIONAL NIL)
        (REST NIL)
        (KEY NIL)
        (KEY-APPEARED? NIL)
        (ALLOW-OTHER-KEYS NIL)
        (STATE :REQUIRED))
    (IL:|for| PARAM IL:|in| ARG-LIST IL:|do| (IF (MEMBER PARAM (&OPTIONAL &KEY &REST))
        (SETQ STATE PARAM)
        (CASE STATE
          (:REQUIRED (PUSH PARAM REQUIRED))
          (&OPTIONAL (PUSH PARAM OPTIONAL))
          (&REST (SETQ REST PARAM))
          (&KEY (IF (EQ PARAM '&ALLOW-OTHER-KEYS)
                    (SETQ ALLOW-OTHER-KEYS T)
                    (PUSH PARAM KEY))))
        (WHEN (EQ PARAM '&KEY)
          (SETQ KEY-APPEARED? T)))
    (VALUES (REVERSE REQUIRED)
            (REVERSE OPTIONAL)
            REST
            (REVERSE KEY)
            KEY-APPEARED? ALLOW-OTHER-KEYS)))

```

```

(DEFUN HAS-CALLS (CALLER CALLEE)
  ;; Tell if CALLEE is called by CALLER at all.
  ;; [JDS 3-10-93: Used to use CALLS to find callee list; changed to CALLSCCODE, because CALLS isn't always loaded.]
  (LET ((REAL-CALLER (OR (GET CALLER 'IL:ADVISED)
                        (GET CALLER 'IL:BROKEN)
                        CALLER)))
    (OR (CONSP (IL:GETD REAL-CALLER))
        (FIND CALLEE (CADR (IL:CALLSCCODE REAL-CALLER))
                 :TEST
                 'EQ))))

```

```

(DEFUN CHANGE-CALLS (FROM TO FN &OPTIONAL FIXER)
  ;; Side-effect the definition of FN to change all calls to FROM into calls to TO. Also save enough information that SI::RESTORE-CALLS can fix up the
  ;; definition again.
  (LET* ((REAL-FN-SYMBOL (OR (GET FN 'IL:ADVISED)
                            (GET FN 'IL:BROKEN)
                            FN))
         (REAL-FN-DEFN (IL:GETD REAL-FN-SYMBOL)))
    (TYPECASE REAL-FN-DEFN
      (CONS
       (CONS
        (WHEN (NULL (GET FN 'IL:NAMESCHANGED))
          ; The function is interpreted.
          ; The first time we change calls, get a copy so as to avoid
          ; sharing structure with the DEFUN form. Ugh.
          (IL:PUTD REAL-FN-SYMBOL (SETQ REAL-FN-DEFN (COPY-TREE REAL-FN-DEFN))))

```

```
(CHANGE-CALLS-IN-LAMBDA FROM TO REAL-FN-DEFN)
(IL:COMPILED-CLOSURE (CHANGE-CALLS-IN-CCODE FROM TO REAL-FN-DEFN))
(OTHERWISE (ERROR "SI::CHANGE-CALLS called on a non-function: ~S" FN))))
```

;; If there's an opposite entry already in the info, just remove it. We assume that we're being called from the same fellow that called us before and  
 ;; that we want to simply undo that other call.

```
(UNLESS (EQ FIXER 'RESTORE-CALLS)
  (FLET ((MATCHING (ENTRY)
          (AND (EQ (FIRST ENTRY)
                  TO)
               (EQ (SECOND ENTRY)
                   FROM))))
    (LET ((CURRENT-INFO (GET FN 'IL:NAMESCHANGED)))
      (IF (SOME #'MATCHING CURRENT-INFO)
          (IF (NULL (CDR CURRENT-INFO))
              (REMPROP FN 'IL:NAMESCHANGED)
              (SETF (GET FN 'IL:NAMESCHANGED)
                    (DELETE-IF #'MATCHING CURRENT-INFO)))
          (PUSH (LIST FROM TO FIXER)
                (GET FN 'IL:NAMESCHANGED))))))
  NIL)
```

```
(DEFUN CHANGE-CALLS-IN-CCODE (FROM TO CCODE)
```

;; Change the calls in a compiled-code object??

```
(IL:FOR REFMAP IL:IN (CDR (IL:CHANGECCODE FROM FROM CCODE))
  IL:DO (LET ((BASE (IL:FETCH (IL:REFMAP IL:CODEARRAY) IL:OF REFMAP)))
        (IL:FOR LOC IL:IN (IL:FETCH (IL:REFMAP IL:DEFLOCS) IL:OF REFMAP)
          IL:DO (IL:CODEBASESETATOM BASE LOC (IL:NEW-SYMBOL-CODE TO (IL:\\ATOMDEFINDEX TO)))))))
```

```
(DEFUN CHANGE-CALLS-IN-LAMBDA (FROM TO LAMBDA-FORM)
```

;;; Wrap all of the right parts of the given LAMBDA-FORM in the proper %WITH-CHANGED-CALLS forms changing calls to FROM into calls to TO.  
 ;;; Actually side-effect the LAMBDA-FORM to make this change.

```
(ECASE (CAR LAMBDA-FORM)
  ((IL:LAMBDA IL:NLAMBDA) (SETF (CDDR LAMBDA-FORM)
                               (ADD-CHANGED-CALL FROM TO (CDDR LAMBDA-FORM))))
  ((LAMBDA) ; For Common Lisp functions, we have to be careful to wrap up
            ; the init-forms for any &OPTIONAL, &KEY, and &AUX
            ; parameters.

  (LET ((STATE :REQUIRED))
    (IL:|for| PARAM IL:|in| (SECOND LAMBDA-FORM)
      IL:|do| (COND
              ((CONSP PARAM)
               (WHEN (AND (CONSP (CDR PARAM))
                          (MEMBER STATE '(&OPTIONAL &KEY &AUX)
                                     :TEST
                                     'EQ))
                 (SETF (SECOND PARAM)
                       (CAR (ADD-CHANGED-CALL FROM TO (LIST (SECOND PARAM))))))
               ((MEMBER PARAM '(&OPTIONAL &REST &KEY &AUX)
                          :TEST
                          'EQ)
                (SETQ STATE PARAM))))
      (SETF (CDDR LAMBDA-FORM)
            (ADD-CHANGED-CALL FROM TO (CDDR LAMBDA-FORM))))))
  NIL)
```

```
(DEFUN ADD-CHANGED-CALL (FROM TO BODY)
```

;;; BODY is a list of forms in which calls to FROM should be changed into calls to TO. If the BODY contains a single form that is a call to the macro  
 ;;; SI::%WITH-CHANGED-CALLS, then we just side-effect that form to add another (FROM . TO) pair. Otherwise, we wrap up the BODY in a new call to  
 ;;; SI::%WITH-CHANGED-CALLS. In either case, we return a list of the SI::%WITH-CHANGED-CALLS form.

;;; Actually, I lied. If it's already a SI::%WITH-CHANGED-CALLS form, and the pair (TO . FROM) is in the list of changes, then we simply remove it from  
 ;;; the list. If the list is now empty, then we remove the SI::%WITH-CHANGED-CALLS form entirely and actually return the former body of the  
 ;;; macro-call.

;;; The effect of this is that you can undo previous additions simply by exchanging the FROM and TO arguments to this function.

```
(COND
  ((AND (NULL (REST BODY))
        (EQ (CAR (FIRST BODY))
            '%WITH-CHANGED-CALLS))
   ;; It's already a call to %WITH-CHANGED-CALLS.
   (LET ((WCC-FORM (FIRST BODY)))
     (COND
      ((MEMBER (CONS TO FROM)
               (SECOND WCC-FORM)
               :TEST
               'EQUAL)
       ;; We're undoing a previous call to ADD-CHANGED-CALL.
```

```
(COND
  ( (NULL (REST (SECOND WCC-FORM))) ; There won't be anything left, so return the old body.
    (CDDR WCC-FORM)
    (T ; Oh, well, there'll still be something there. Just remove the
      ; particular pair.
      (SETF (SECOND WCC-FORM)
            (DELETE (CONS TO FROM)
                   (SECOND WCC-FORM)
                   :TEST
                   'EQUAL))
            (LIST WCC-FORM)))
      (T (PUSH (CONS FROM TO)
              (SECOND WCC-FORM)
              (LIST WCC-FORM))))
      (T ;; It's not already a %WITH-CHANGED-CALLS form, so make it into one.
        `((%WITH-CHANGED-CALLS (, (CONS FROM TO))
                               ,@BODY))))))
```

```
(DEFMACRO %WITH-CHANGED-CALLS (A-LIST &BODY BODY)
  `(MACROLET ,(IL:FOR PAIR IL:IN A-LIST IL:COLLECT `((, (CAR PAIR)
                                                    (&REST ARGS)
                                                    (CONS ', (CDR PAIR)
                                                       ARGS))))
    ,@BODY))
```

```
(DEFUN RESTORE-CALLS (FN)
  (IL:|for| ENTRY IL:|in| (GET FN 'IL:NAMESCHANGED) IL:|do| (DESTRUCTURING-BIND (FROM TO FIXER)
  ENTRY
  (CHANGE-CALLS TO FROM FN 'RESTORE-CALLS)
  (FUNCALL FIXER FROM TO FN)))
  (AND (REMPROP FN 'IL:NAMESCHANGED)
  T))
```

;;; Support for function name mapping

```
(DEFUN XCL::NAME-OF-EXECUTABLE (XCL::FN-NAME) ; Edited 8-Jan-92 13:28 by jrb:
  ;; *FUNCTION-NAME-MAPPERS* is a list of functions that get called to attempt to map FN-NAME to a symbol where the executable for FN-NAME
  ;; resides under Medley. Currently there is only one, named COMMON-LISP-MAPPER, which handles symbols naming functions and macros and
  ;; (SETF FOO) forms.
  (DO ((XCL::FNS XCL::*FUNCTION-NAME-MAPPERS* (CDR XCL::FNS))
      (XCL::RESULT)
      (XCL::NO-IN-FN))
    ((NULL XCL::FNS)
     NIL)
    (MULTIPLE-VALUE-SETQ (XCL::RESULT XCL::NO-IN-FN)
      (FUNCALL (CAR XCL::FNS)
               XCL::FN-NAME))
    (WHEN XCL::RESULT
      (UNLESS (EQ XCL::RESULT XCL::FN-NAME)
        (SETF (GET XCL::RESULT 'TRUE-NAME)
              XCL::FN-NAME))
      (RETURN-FROM XCL::NAME-OF-EXECUTABLE (VALUES XCL::RESULT XCL::NO-IN-FN))))))
```

```
(DEFUN XCL::COMMON-LISP-MAPPER (XCL::FN-NAME) ; Edited 5-Oct-92 22:50 by jrb:
  ;; Recognizes standard Common Lisp names of executable things and returns the symbol where their executable should reside; the second value is
  ;; true when it's impossible to selectively break/trace/advise (i.e. (FOO :IN :BAR))
  (LET (XCL::RESULT)
    (COND
      ((SYMBOLP XCL::FN-NAME)
       (IF (AND (SETQ XCL::RESULT (MACRO-FUNCTION XCL::FN-NAME))
                (SYMBOLP XCL::RESULT)
                ;; These two symbols can be returned by MACRO-FUNCTION if FN-NAME is an Interlisp CLISP word or an NLAMBDA of
                ;; some sort
                (NOT (EQ XCL::RESULT 'IL:CLISPEXPANSION))
                (NOT (EQ XCL::RESULT 'IL:\\INTERLISP-NLAMBDA-MACRO))))
          (VALUES XCL::RESULT T)
          (XCL::FN-NAME)))
      ((CL::SETF-NAME-P XCL::FN-NAME)
       (OR (GET (SECOND XCL::FN-NAME)
                :SETF-DEFUN)
           (XCL::DEFUN-SETF-NAME (SECOND XCL::FN-NAME))))))
```

```
(DEFVAR XCL::*FUNCTION-NAME-MAPPERS* '(XCL::COMMON-LISP-MAPPER))
```

```
(IL:DEFINEQ
```

**(IL:VIRGINFN**

(IL:LAMBDA (IL:FN IL:MAKE-VIRGIN?) ; Edited 17-Dec-91 20:23 by jrb:

(LET\* ((IL:EXECUTABLE-NAME (XCL::NAME-OF-EXECUTABLE IL:FN))
(IL:BROKEN-DEFN (IL:GETPROP IL:EXECUTABLE-NAME 'IL:BROKEN))
(IL:ADVISED-DEFN (IL:GETPROP IL:EXECUTABLE-NAME 'IL:ADVISED))
(IL:CHANGED-NAMES (IL:GETPROP IL:EXECUTABLE-NAME 'IL:NAMESCHANGED))
(IL:EXPR-DEFN (IL:GETPROP IL:EXECUTABLE-NAME 'IL:EXPR))
IL:REAL-DEFN

(IL:|if| IL:MAKE-VIRGIN?
IL:|then|

;; We're supposed to return the function to its virgin state, without any breaks, advice, or changed names.

(IL:|if| IL:BROKEN-DEFN
IL:|then| (XCL:UNBREAK-FUNCTION IL:FN)
(FORMAT \*TERMINAL-IO\* "~S unbroken.~%" IL:FN))
(IL:|if| IL:ADVISED-DEFN
IL:|then| (IL:APPLY 'IL:UNADVISE (LIST IL:FN))
(FORMAT \*TERMINAL-IO\* "~S unadvised.~%" IL:FN))
(IL:|if| IL:CHANGED-NAMES
IL:|then| (RESTORE-CALLS IL:EXECUTABLE-NAME)
(FORMAT \*TERMINAL-IO\* "Names restored in ~S.~%" IL:FN))
(IL:SETQ IL:REAL-DEFN (IL:GETD IL:EXECUTABLE-NAME))
(IL:|if| (AND (NOT (IL:EXPRP IL:REAL-DEFN))
(NOT (NULL IL:EXPR-DEFN)))
IL:|then| (IL:SETQ IL:REAL-DEFN IL:EXPR-DEFN))
IL:REAL-DEFN

IL:|else|

;; We're not supposed to change the state of the function with respect to breaking, advising or changed names. We're just
;; supposed to return the real, core definition.

(IL:SETQ IL:REAL-DEFN (IL:GETD (OR IL:ADVISED-DEFN IL:BROKEN-DEFN IL:EXECUTABLE-NAME)))
(IL:|if| (OR (IL:NLISTP IL:REAL-DEFN)
(IL:NLISTP (CDR IL:REAL-DEFN)))
IL:|then| (OR IL:EXPR-DEFN IL:REAL-DEFN)
IL:|else| (IL:|if| IL:CHANGED-NAMES
IL:|then| (IL:SETQ IL:REAL-DEFN (IL:COPY IL:REAL-DEFN))
(IL:|for| IL:X IL:|in| IL:CHANGED-NAMES
IL:|do| (XCL:DESTRUCTURING-BIND (IL:FROM IL:TO)
IL:X
(CHANGE-CALLS-IN-LAMBDA IL:TO IL:FROM IL:REAL-DEFN))))
IL:REAL-DEFN))))

**(CONSTRUCT-MIDDLE-MAN**

(LAMBDA (OBJECT-FN IN-FN)
(BLOCK CONSTRUCT-MIDDLE-MAN
(LET ((\*PRINT-CASE\* :UPCASE)
(INTERN (FORMAT NIL "~A in ~A::~~A" OBJECT-FN (PACKAGE-NAME (SYMBOL-PACKAGE IN-FN))
IN-FN)
(SYMBOL-PACKAGE OBJECT-FN))))))

)

(IL:PUTPROPS IL:NAMESCHANGED IL:PROPTYPE IGNORE)

;; Arrange for the proper compiler and package/readtable.

(IL:PUTPROPS IL:WRAPPERS IL:FILETYPE :FAKE-COMPILE-FILE)

(IL:PUTPROPS IL:WRAPPERS IL:MAKEFILE-ENVIRONMENT (:READTABLE "XCL" :PACKAGE "SI"))

(IL:DECLARE\ : IL:DOEVAL@COMPILE IL:DONTCOPY

(IL:FILESLOAD (IL:LOADCOMP)
IL:ACODE)

)

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

(IL:ADDTOVAR IL:NLAMA )

(IL:ADDTOVAR IL:NLAML )

(IL:ADDTOVAR IL:LAMA CONSTRUCT-MIDDLE-MAN)

)

(IL:PUTPROPS IL:WRAPPERS IL:COPYRIGHT ("Venue & Xerox Corporation" 1987 1988 1990 1991 1992 1993))

---

**FUNCTION INDEX**

ADD-CHANGED-CALL .....	4	COMPILED-FUNCTION-ARGLIST .....	1	HAS-CALLS .....	3
CHANGE-CALLS .....	3	COMPILED-FUNCTION-DEBUGGING-INFO .	1	XCL::NAME-OF-EXECUTABLE .....	5
CHANGE-CALLS-IN-CCODE .....	4	COMPILED-FUNCTION-INTERLISP? .....	1	NAMED-FUNCTION-WRAPPER-INFO .....	2
CHANGE-CALLS-IN-LAMBDA .....	4	CONSTRUCT-MIDDLE-MAN .....	6	PARSE-CL-ARGLIST .....	3
CLEAN-UP-CL-ARGLIST .....	2	FUNCTION-WRAPPER-INFO .....	2	RESTORE-CALLS .....	5
XCL::COMMON-LISP-MAPPER .....	5	GET-STORED-ARGLIST .....	2	IL:VIRGINFN .....	6

---

**PROPERTY INDEX**

IL:NAMESCHANGED .....	6	IL:WRAPPERS .....	6
-----------------------	---	-------------------	---

---

**VARIABLE INDEX**

XCL::*FUNCTION-NAME-MAPPERS* .....	5
------------------------------------	---

---

**MACRO INDEX**

%WITH-CHANGED-CALLS .....	5
---------------------------	---

---