

File created: 24-Mar-92 14:56:18 {DSK}<usr>local>lde>lispcore>sources>CMLDEFFER.;3

changes to: (IL:VARS IL:CMLDEFFERCOMS)

previous date: 4-Jan-92 15:32:26 {DSK}<usr>local>lde>lispcore>sources>CMLDEFFER.;2

Read Table: XCL

Package: XEROX-COMMON-LISP

Format: XCCS

; Copyright (c) 1986, 1990, 1987, 1988, 1990, 1992 by Venue & Xerox Corporation. All rights reserved.

(IL:RPAQQ IL:CMLDEFFERCOMS

(

::: DEF-DEFINE-TYPE and DEFDEFINER -- Your One-Stop Providers of Customized File Manager Facilities.

:: BE VERY CAREFUL CHANGING ANYTHING IN THIS FILE!!! It is heavily self-referential and thick with bootstrapping problems. All  
:: but the most trivial changes (and some of those) are very tricky to make without blowing yourself out of the water... You have been  
:: warned.

::: Also see the file deffer-runtime for stuff that must be defined before fasl files may be loaded into the init

```
(IL:COMS ; Filepkg interface
  (IL:FUNCTIONS REMOVE-COMMENTS PPRINT-DEFINER PPRINT-DEFINER-FITP PPRINT-DEFINER-RECURSE)
  (IL:VARIABLES IL:*REMOVE-INTERLISP-COMMENTS*)
  ; Share with xcl?
  (IL:FUNCTIONS %DEFINE-TYPE-DELDEF %DEFINE-TYPE-GETDEF %DEFINE-TYPE-FILE-DEFINITIONS
    %DEFINE-TYPE-FILEGETDEF %DEFINE-TYPE-SAVE-DEFN %DEFINE-TYPE-PUTDEF)
(IL:COMS ; Compatibility with old cmldiffer
  (IL:DECLARE\ : IL:DOCOPY IL:DONTEVAL@LOAD (IL:P (IL:MOVD ' %DEFINE-TYPE-DELDEF
    ' IL:\\DEFINE-TYPE-DELDEF)
    (IL:MOVD ' %DEFINE-TYPE-GETDEF
    ' IL:\\DEFINE-TYPE-GETDEF)
    (IL:MOVD ' %DEFINE-TYPE-FILE-DEFINITIONS
    ' IL:\\DEFINE-TYPE-FILE-DEFINITIONS)
    (IL:MOVD ' %DEFINE-TYPE-FILEGETDEF
    ' IL:\\DEFINE-TYPE-FILEGETDEF)
    (IL:MOVD ' %DEFINE-TYPE-SAVE-DEFN
    ' IL:\\DEFINE-TYPE-SAVE-DEFN)
    (IL:MOVD ' %DEFINE-TYPE-PUTDEF
    ' IL:\\DEFINE-TYPE-PUTDEF)
    (IL:MOVD ' PPRINT-DEFINER ' IL:PPRINT-DEFINER))))
(IL:DECLARE\ : IL:DOCOPY IL:DONTEVAL@LOAD
  (IL:P ; Set up fake definer prototype stuff for FNS
    (ADD-PROTOTYPE-FN ' IL:FNS ' IL:NLAMBDA
      #' (LAMBDA (NAME)
        (AND (SYMBOLP NAME)
          `(IL:DEFINEQ (,NAME (IL:NLAMBDA ,@( %MAKE-FUNCTION-PROTOTYPE))))))
    (ADD-PROTOTYPE-FN ' IL:FNS ' IL:LAMBDA
      #' (LAMBDA (NAME)
        (AND (SYMBOLP NAME)
          `(IL:DEFINEQ (,NAME (IL:LAMBDA ,@( %MAKE-FUNCTION-PROTOTYPE))))))
(IL:COMS ; The groundwork for bootstrapping
  (IL:DEFINE-TYPES IL:DEFINE-TYPES IL:FUNCTIONS IL:VARIABLES)
  ; DefDefiner itself and friends
  (IL:FUNCTIONS SI::EXPANSION-FUNCTION SI::MACRO-FUNCALL WITHOUT-FILEPKG)
(IL:COMS ; Compatibility with old cmldiffer
  (IL:FUNCTIONS IL:WITHOUT-FILEPKG)
  ; Some special forms
  (IL:FUNCTIONS DEFINER DEFINER-VARIABLE-TYPE NAMED-PROGN)
(IL:COMS ; Auxiliary functions
  (IL:FUNCTIONS GET-DEFINER-NAME %DELETE-DEFINER)
  (IL:FUNCTIONS DEF-DEFINE-TYPE DEFDEFINER DEFDEFINER-VARIABLE-TYPE)
  (IL:FUNCTIONS %EXPAND-DEFINER %DEFINER-NAME))
(IL:COMS ; The most commonly-used definers
  (IL:FUNCTIONS DEFUN DEFINLINE DEFMACRO)
  (IL:FUNCTIONS DEFVAR DEFPARAMETER DEFCONSTANT DEFGLOBALVAR DEFGLOBALPARAMETER))
(IL:COMS ; Here so that the evaluator can be in the init without definers
  ; being in the init.
  (IL:DEFINE-TYPES IL:SPECIAL-FORMS)
  (IL:FUNCTIONS %REMOVE-SPECIAL-FORM)
  (IL:FUNCTIONS DEFINE-SPECIAL-FORM)
  ; Form for defining interpreters of special forms
  )
(IL:COMS ; Don't note changes to these properties/variables
  (IL:PROP IL:PROPTYPE IL:MACRO-FN :UNDEFINERS IL:UNDEFINERS :DEFINER-FOR IL:DEFINER-FOR
    :DEFINED-BY IL:DEFINED-BY :DEFINITION-NAME IL:DEFINITION-NAME)
  ; Templates for definers not defined here. These should really
  ; be where they're defined.
  (IL:PROP :DEFINITION-PRINT-TEMPLATE DEFCOMMAND DEFINE-CONDITION DEFINE-MODIFY-MACRO
    DEFINE-SETF-METHOD DEFSETF DEFSTRUCT DEFTYPE))
:: Arrange for the correct compiler to be used.
```

```
(IL:PROP (IL:FILETYPE IL:MAKEFILE-ENVIRONMENT)
IL:CMLDEFFER))
```

;; DEF-DEFINE-TYPE and DEFDEFINER -- Your One-Stop Providers of Customized File Manager Facilities.

;; BE VERY CAREFUL CHANGING ANYTHING IN THIS FILE!!! It is heavily self-referential and thick with bootstrapping problems. All but the most trivial changes (and some of those) are very tricky to make without blowing yourself out of the water... You have been warned.

;; Also see the file deffer-runtime for stuff that must be defined before fasl files may be loaded into the init

;; Filepkg interface

```
(DEFUN REMOVE-COMMENTS (X)
```

;; Removes SEdit-style comments from the given list structure.

```
(COND
  ((NOT (CONSP X))
   X)
  ((AND (CONSP (CAR X))
        (EQ (CAAR X)
            'IL:*)
        (CONSP (CDAR X))
        (OR (MEMBER (CADAR X)
                    '(IL:\; IL:|;;| IL:|;;;| IL:|;;;| IL:\|)
                    :TEST
                    #'EQ)
            ; a sedit comment
            (EQ IL:*REMOVE-INTERLISP-COMMENTS* T) ; always strip
            (PROGN (IF (EQ IL:*REMOVE-INTERLISP-COMMENTS* ' :WARN)
                      (WARN "Possible comment not stripped ~S" (CAR X)))
                  NIL)))
   (REMOVE-COMMENTS (CDR X)))
  (T (LET ((A (REMOVE-COMMENTS (CAR X)))
           (D (REMOVE-COMMENTS (CDR X))))
      (IF (AND (EQ A (CAR X))
              (EQ D (CDR X)))
          X
          (CONS A D))))))
```

```
(DEFUN PPRINT-DEFINER (DEFINE-EXPRESSION)
```

```
(DECLARE (SPECIAL IL:FORMFLG IL:SPACEWIDTH)) ; Bound in prettyprinter
```

```
(COND
  ((OR (NULL IL:FORMFLG)
        (ATOM (CDR DEFINE-EXPRESSION)))
   ; Degenerate cases or printing as a quoted form--punt to default
   ; prettyprinting
   DEFINE-EXPRESSION)
  (T (LET ((IL:TAIL DEFINE-EXPRESSION)
           (IL:LEFT (IL:DSPXPOSITION))
           TEMPLATE TOP-LEVEL-P NEXT TYPE FORM NEWLINEP)
      (DECLARE (SPECIAL IL:TAIL IL:LEFT)) ; For comment printer
      (SETQ TOP-LEVEL-P (EQ IL:LEFT (IL:DSPLEFTMARGIN))) ; Printing definition to file, etc.
      (SETQ IL:LEFT (+ IL:LEFT (* 3 IL:SPACEWIDTH))) ; Place we will indent body
      (IL:PRIN1 "(")
      (IL:PRIN2 (CAR IL:TAIL))
      (SETQ TEMPLATE (OR (GET (POP IL:TAIL)
                             :DEFINITION-PRINT-TEMPLATE)
                        ' (:NAME))))
```

;; This code should, and doesn't, pay attention to the NAME function to determine where the name is to decide what should and shouldn't be bold. Right now, it always bolds the second thing. Fortunately, we currently don't have any definers that don't have either the second or CAR of the second as the definition name.

;; Also, this code should be careful about calling the NAME function on the form. Sometimes, the form is not really a call to the definer but instead a back-quoted expression in a macro. In most such cases, the name is not really there; some comma-quoted expression is there instead.

```
(IL:WHILE (CONSP IL:TAIL)
  IL:DO (COND
    ((AND (LISTP (SETQ NEXT (CAR IL:TAIL)))
          (EQ (CAR NEXT)
              IL:COMMENTFLG)
          (IL:SEMI-COLON-COMMENT-P NEXT))
     ; Comments can appear anywhere, so print this one without
     ; consuming the template. ENDLINE has side effect of printing
     ; comments
     (IL:SUBPRINT/ENDLINE IL:LEFT *STANDARD-OUTPUT*)
     (SETQ NEWLINEP T))
    ((OR (ATOM TEMPLATE)
          (EQ (SETQ TYPE (POP TEMPLATE))
              :BODY))
     ; Once we hit the body, there's nothing more special to do.
     (RETURN))
    (T (IL:SPACES 1)
      (CASE TYPE
        (:NAME
         ; Embolden the name of this thing
         (SETQ NEWLINEP NIL)
         (COND
          ((NOT TOP-LEVEL-P)
           ; Nothing special here--could even be a backquoted thing
```

```

      (PPRINT-DEFINER-RECURSE) )
      (T (POP IL:TAIL)
        (COND
          ((CONSP NEXT) ; Name is a list. Assume the real name is the car and the rest is
            ; an options list or something
            (UNLESS (EQ (IL:DSYPOSITION)
              (PROGN (IL:PRIN1 " ("
                (IL:PRINTOUT NIL IL:.FONT IL:LAMBDAFONT IL:.P2
                  (CAR NEXT)
                    IL:.FONT IL:DEFAULTFONT)
                (IL:SPACES 1)
                (IL:PRINTDEF (CDR NEXT)
                  T T T IL:FNSLST)
                (IL:PRIN1 " ")
                (IL:DSYPOSITION)))
            ; This thing took more than one line to print, so go to new line
            (IL:SUBPRINT/ENDLINE IL:LEFT *STANDARD-OUTPUT*)
            (SETQ NEWLINEP T)))
          (T ; Atomic name is bold
            (IL:PRINTOUT NIL IL:.FONT IL:LAMBDAFONT IL:.P2 NEXT IL:.FONT
              IL:DEFAULTFONT))))))
      (:ARG-LIST ; NEXT is some sort of argument list.
      (COND
        ((NULL NEXT) ; If NIL, be sure to print as ()
          (IL:PRIN1 " ()")
          (POP IL:TAIL))
        (T (PPRINT-DEFINER-RECURSE)))
      (SETQ NEWLINEP NIL))
      (T ; Just print it, perhaps starting a new line
        (UNLESS (OR NEWLINEP (PPRINT-DEFINER-FITP NEXT))
          ; Go to new line if getting crowded
          (IL:PRINENDLINE IL:LEFT))
        (PPRINT-DEFINER-RECURSE)
        (SETQ NEWLINEP NIL))))))
;; We've now gotten to the end of stuff we know how to print. Just prettyprint the rest
(UNLESS (NULL IL:TAIL)
  (COND
    (NEWLINEP ; Already on new line
      )
    ((OR (EQ TYPE :BODY)
      (NOT (PPRINT-DEFINER-FITP (CAR IL:TAIL))))
      ; Go to new line and indent a bit. Always do this for the part
      ; matching &BODY, whether or not the prettyprinter thought that
      ; the remainder would "fit"
      (IL:PRINENDLINE IL:LEFT NIL T))
    (T (IL:SPACES 1)))
  (IL:WHILE (AND (CONSP IL:TAIL)
    (ATOM (SETQ FORM (CAR IL:TAIL))))
    (IL:DO ; Print this doc string or whatever on its own line. This is because otherwise the prettyprinter gets confused and
      ; tries to put the next thing after the string
      (PPRINT-DEFINER-RECURSE)
      (WHEN (AND (KEYWORDP FORM)
        (CONSP IL:TAIL)) ; Some sort of keyword-value pair stuff--print it on same line
        (IL:SPACES 1)
        (PPRINT-DEFINER-RECURSE))
      (WHEN (NULL IL:TAIL)
        (RETURN))
      (IL:SUBPRINT/ENDLINE IL:LEFT *STANDARD-OUTPUT*))
      (IL:PRINTDEF IL:TAIL T T T IL:FNSLST))
      (IL:PRIN1 " ")
      NIL))))

```

```

(DEFUN PPRINT-DEFINER-FITP (ITEM)

```

;; True if it won't look silly to try to print ITEM at current position instead of starting new line

```

  (IF (CONSP ITEM)
    (OR (EQ (CAR ITEM)
      IL:COMMENTFLG)
      (AND (< (IL:COUNT ITEM)
        20)
        (IL:FITP ITEM)))
    (< (+ (IL:DSYPOSITION)
      (IL:STRINGWIDTH ITEM *STANDARD-OUTPUT*))
      (IL:DSPRIGHTMARGIN))))

```

```

(DEFUN PPRINT-DEFINER-RECURSE ()

```

;; Print and pop the next element. Prettyprinter uses the variable IL:TAIL for lookahead

```

  (DECLARE (SPECIAL IL:TAIL))
  (IL:SUPERPRINT (CAR IL:TAIL)
    IL:TAIL NIL *STANDARD-OUTPUT*)
  (SETQ IL:TAIL (CDR IL:TAIL))

```

```
(DEFVAR IL:*REMOVE-INTERLISP-COMMENTS* ' :WARN "Either NIL (don't) T (always do) or :WARN (don't and warn)")
```

:: Share with xcl?

```
(DEFUN %DEFINE-TYPE-DELDEF (NAME TYPE)
  ;; DELETE definition of definer-defined NAME as TYPE
  (UNDOABLY-SETF (DOCUMENTATION NAME TYPE)
    NIL)
  (LET* ((HT (GETHASH TYPE *DEFINITION-HASH-TABLE*))
        (DEFN (AND HT (GETHASH NAME HT))))
    (AND HT (IL:/PUTHASH NAME NIL HT))
    (DOLIST (FN (OR (GET TYPE ' :UNDEFINERS)
                    (GET TYPE ' IL:UNDEFINERS)))
      (FUNCCALL FN NAME))
    (DOLIST (FN (OR (GET (CAR DEFN)
                        ' :UNDEFINERS)
                    (GET (CAR DEFN)
                        ' IL:UNDEFINERS)))
      (FUNCCALL FN NAME))
    NAME))
```

```
(DEFUN %DEFINE-TYPE-GETDEF (NAME TYPE OPTIONS)
  ;; GETDEF method for all definers. The EDIT is so that when you say EDITDEF you get a copy & can know when you made edits.
  (LET* ((HASH-TABLE (GETHASH TYPE *DEFINITION-HASH-TABLE*))
        (DEFN (AND HASH-TABLE (GETHASH NAME HASH-TABLE))))
    (IF (TYPECASE OPTIONS
        (CONS (MEMBER ' IL:EDIT OPTIONS :TEST #'EQ))
        (T (EQ OPTIONS ' IL:EDIT)))
      (COPY-TREE DEFN)
      DEFN))
```

```
(DEFUN %DEFINE-TYPE-FILE-DEFINITIONS (TYPE NAMES)
  ;; get the definitions for NAMES suitable for printing on a file. Like GETDEF but checks.
  (MAPCAR #' (LAMBDA (NAME)
    (LET ((DEF (%DEFINE-TYPE-GETDEF NAME TYPE ' (IL:NOCOPY))))
      (IF (NULL DEF)
        (ERROR ' IL:NO-SUCH-DEFINITION :NAME NAME :TYPE TYPE)
        DEF)))
    NAMES))
```

```
(DEFUN %DEFINE-TYPE-FILEGETDEF (NAME TYPE SOURCE OPTIONS NOTFOUND)
  (LET ((VAL (IL:LOADFNS NIL SOURCE ' IL:GETDEF)
        ;; The bletcherous lambda form is require by the interface to loadfns (can't pass a closure)
        `(IL:LAMBDA (FIRST SECOND)
          (AND (MEMBER FIRST ' , (OR (GET TYPE ' :DEFINED-BY)
                                     (GET TYPE ' IL:DEFINED-BY))
              :TEST #'EQ)
            (LET ((NAMER (OR (GET FIRST ' :DEFINITION-NAME)
                            (GET FIRST ' IL:DEFINITION-NAME)
                            ' SECOND)))
              (IF (EQ NAMER ' SECOND)
                (EQUAL SECOND ' ,NAME)
                (EQUAL (FUNCCALL NAMER (REMOVE-COMMENTS (IL:READ)))
                        ' ,NAME)))))))
    (COND
      ((EQ (CAAR VAL)
           ' IL:NOT-FOUND\:)
        NOTFOUND)
      ((CDR VAL)
        (CONS ' PROG VAL))
      (T (CAR VAL))))))
```

```
(DEFUN %DEFINE-TYPE-SAVE-DEFN (NAME TYPE DEFINITION)
  (SETQ TYPE (IL:GETFILEPKGTYPE TYPE ' TYPE))
  (LET ((HASH-TABLE (GETHASH TYPE *DEFINITION-HASH-TABLE*))
        (WHEN (NULL HASH-TABLE)
          (WARN "Couldn't find a hash-table for ~S definitions.~%One will be created." TYPE)
          (SETQ HASH-TABLE (SETF (GETHASH TYPE *DEFINITION-HASH-TABLE*)
                                (MAKE-HASH-TABLE :TEST #'EQUAL :SIZE 50 :REHASH-SIZE 50))))
    (LET ((OLD-DEFINITION (GETHASH NAME HASH-TABLE))
          (UNLESS (EQUAL DEFINITION OLD-DEFINITION)
            (WHEN (AND OLD-DEFINITION (NOT (EQ IL:DFNFLG T)))
              (FORMAT *TERMINAL-IO* "~&New ~A definition for ~S~:[~; (but not installed)~].~%" TYPE NAME
                (MEMBER IL:DFNFLG ' (IL:PROP IL:ALLPROP)
                  :TEST
```

```

      #'EQ)))
      (IL:/PUTHASH NAME DEFINITION HASH-TABLE)
      (IL:MARKASCHANGED NAME TYPE (IF OLD-DEFINITION
        'IL:CHANGED
        'IL:DEFINED))))))

```

```

(DEFUN %DEFINE-TYPE-PUTDEF (NAME TYPE DEFINITION REASON)
  (IF (NULL DEFINITION)
    (%DEFINE-TYPE-DELDEF NAME TYPE)
    (LET ((DEFN-WITHOUT-COMMENTS (REMOVE-COMMENTS DEFINITION)))
      (UNLESS (AND (CONSP DEFN-WITHOUT-COMMENTS)
        (MEMBER (CAR DEFN-WITHOUT-COMMENTS)
          (OR (GET TYPE ':DEFINED-BY)
            (GET TYPE 'IL:DEFINED-BY)))
          :TEST #'EQ)
        (EQUAL NAME (FUNCALL (OR (GET (CAR DEFN-WITHOUT-COMMENTS)
          ':DEFINITION-NAME)
          (GET (CAR DEFN-WITHOUT-COMMENTS)
            'IL:DEFINITION-NAME)
          'SECOND)
          DEFN-WITHOUT-COMMENTS)))
          (SIGNAL 'IL:DEFINER-MISMATCH :NAME NAME :TYPE TYPE :DEFINITION DEFINITION))
      (SETQ DEFINITION (COPY-TREE DEFINITION))
      (EVAL (IF IL:LISPXHIST
        (MAKE-UNDOABLE DEFINITION)
        DEFINITION))))))

```

:: Compatibility with old cmldeffer

```

(IL:DECLARE\ : IL:DOCOPY IL:DONTEVAL@LOAD
(IL:MOVD '%DEFINE-TYPE-DELDEF 'IL:\\DEFINE-TYPE-DELDEF)
(IL:MOVD '%DEFINE-TYPE-GETDEF 'IL:\\DEFINE-TYPE-GETDEF)
(IL:MOVD '%DEFINE-TYPE-FILE-DEFINITIONS 'IL:\\DEFINE-TYPE-FILE-DEFINITIONS)
(IL:MOVD '%DEFINE-TYPE-FILEGETDEF 'IL:\\DEFINE-TYPE-FILEGETDEF)
(IL:MOVD '%DEFINE-TYPE-SAVE-DEFN 'IL:\\DEFINE-TYPE-SAVE-DEFN)
(IL:MOVD '%DEFINE-TYPE-PUTDEF 'IL:\\DEFINE-TYPE-PUTDEF)
(IL:MOVD 'PPRINT-DEFINER 'IL:PPRINT-DEFINER)
)
(IL:DECLARE\ : IL:DOCOPY IL:DONTEVAL@LOAD

```

:: Set up fake definer prototype stuff for FNS

```

(ADD-PROTOTYPE-FN 'IL:FNS 'IL:NLAMBDA #'(LAMBDA (NAME)
  (AND (SYMBOLP NAME)
    `(IL:DEFINEQ (,NAME (IL:NLAMBDA ,@(%MAKE-FUNCTION-PROTOTYPE)
    )))))
(ADD-PROTOTYPE-FN 'IL:FNS 'IL:LAMBDA #'(LAMBDA (NAME)
  (AND (SYMBOLP NAME)
    `(IL:DEFINEQ (,NAME (IL:LAMBDA ,@(%MAKE-FUNCTION-PROTOTYPE)
    )))))
)

```

:: The groundwork for bootstrapping

```

(DEF-DEFINE-TYPE IL:DEFINE-TYPES "Definition type")
(DEF-DEFINE-TYPE IL:FUNCTIONS "Common Lisp functions/macros"
  :UNDEFINER IL:UNDOABLY-FMAKUNBOUND)
(DEF-DEFINE-TYPE IL:VARIABLES "Common Lisp variables"
  :UNDEFINER UNDOABLY-MAKUNBOUND)

```

:: DefDefiner itself and friends

```

(DEFUN SI::EXPANSION-FUNCTION (NAME ARG-LIST BODY)

```

::: Shared code between DEFMACRO and DEFDEFINER. Takes the parts of a DEFMACRO and returns two values: a LAMBDA form for the expansion function, and the documentation string found, if any.

```

(MULTIPLE-VALUE-BIND (PARSED-BODY PARSED-DECLARATIONS PARSED-DOCSTRING)

```

```
(IL:PARSE-DEFMACRO ARG-LIST 'SI::$$MACRO-FORM BODY NAME NIL :ENVIRONMENT 'SI::$$MACRO-ENVIRONMENT)
(VALUE (LAMBDA (SI::$$MACRO-FORM SI::$$MACRO-ENVIRONMENT)
  ,@PARSED-DECLARATIONS
  (BLOCK ,NAME ,PARSED-BODY))
  PARSED-DOCSTRING))
```

```
(DEFMACRO SI::MACRO-FUNCALL (EXPANSION-FUNCTION MACRO-CALL ENV)
```

;; Used by DEFDEFINER as a mechanism for delaying macro-expansion until after checking the value of DFNFLG. The arguments (unevaluated) are a macro-expansion function and a call on that macro. The call to MACRO-FUNCALL should expand into the result of expanding the given macro-call.

```
(FUNCALL EXPANSION-FUNCTION MACRO-CALL ENV))
```

```
(DEFMACRO WITHOUT-FILEPKG (&BODY BODY)
  `(PROGN (EVAL-WHEN (LOAD)
    ,@BODY)
    (EVAL-WHEN (EVAL)
      (UNLESS (OR (EQ IL:DFNFLG 'IL:PROP)
        (EQ IL:DFNFLG 'IL:ALLPROP))
        (LET ((IL:FILEPKGFLG NIL)
          (IL:DFNFLG T))
            ,@BODY))))))
```

;; Compatibility with old cmldeffer

```
(DEFMACRO IL:WITHOUT-FILEPKG (&BODY BODY)
  `(WITHOUT-FILEPKG ,@BODY))
```

;; Some special forms

```
(DEFMACRO DEFINER (TYPE NAME DEFINITION &OPTIONAL ENV)
  (LET* ((EXPANDER (GET NAME :DEFINITION-EXPANDER))
    (DEFINITION-WITHOUT-COMMENTS (REMOVE-COMMENTS DEFINITION))
    (DEFINITION-NAME (FUNCALL (GET NAME :DEFINITION-NAME)
      DEFINITION-WITHOUT-COMMENTS)))
    `(PROGN (WITHOUT-FILEPKG (SI::MACRO-FUNCALL ,EXPANDER ,DEFINITION-WITHOUT-COMMENTS ,ENV))
      (EVAL-WHEN (EVAL)
        (UNLESS (NULL IL:FILEPKGFLG)
          (%DEFINE-TYPE-SAVE-DEFN ' ,DEFINITION-NAME ' ,TYPE ' ,DEFINITION))))
```

```
(DEFMACRO DEFINER-VARIABLE-TYPE (NAME DEFINITION &OPTIONAL ENV)
  (LET* ((DEFINITION-WITHOUT-COMMENTS (REMOVE-COMMENTS DEFINITION))
    (TYPE (FUNCALL (GET NAME :TYPE-DISCRIMINATOR)
      DEFINITION-WITHOUT-COMMENTS))
    (EXPANDER (GETF (GET NAME :DEFINITION-EXPANDER)
      TYPE))
    (DEFINITION-NAME (FUNCALL (GET NAME :DEFINITION-NAME)
      DEFINITION-WITHOUT-COMMENTS)))
    `(PROGN (WITHOUT-FILEPKG (SI::MACRO-FUNCALL ,EXPANDER ,DEFINITION-WITHOUT-COMMENTS ,ENV))
      (EVAL-WHEN (EVAL)
        (UNLESS (NULL IL:FILEPKGFLG)
          (%DEFINE-TYPE-SAVE-DEFN ' ,DEFINITION-NAME ' ,TYPE ' ,DEFINITION))))
```

```
(DEFMACRO NAMED-PROGN (DEFINER NAME &REST FORMS)
```

;; Used by the compiler when processing definers

```
`(PROGN ,@FORMS ' ,NAME))
```

;; Auxiliary functions

```
(DEFUN GET-DEFINER-NAME (DEFINER STRING)
  (VALUES (INTERN (CONCATENATE 'STRING STRING (STRING DEFINER))
    (SYMBOL-PACKAGE DEFINER))))
```

```
(DEFUN %DELETE-DEFINER (NAME)
  (AND (SYMBOLP NAME)
    (LET ((TYPE (OR (GET NAME ' :DEFINER-FOR)
      (GET NAME ' IL:DEFINER-FOR))))
      (IL:/REMPROP NAME ' :DEFINER-FOR)
      (IL:/REMPROP NAME ' IL:DEFINER-FOR)
      (IL:/REMPROP NAME ' :DEFINITION-NAME)
      (IL:/REMPROP NAME ' IL:DEFINITION-NAME)
      (IL:/REMPROP NAME ' :DEFINITION-EXPANDER)
      (WHEN TYPE
        (IF (GET TYPE ' :DEFINED-BY)
          (IL:/PUTPROP TYPE ' :DEFINED-BY (REMOVE NAME (GET TYPE ' :DEFINED-BY))))
```

```

(IL:/PUTPROP TYPE 'IL:DEFINED-BY (REMOVE NAME (GET TYPE 'IL:DEFINED-BY)))
;; need to remove the prototype function!
(LET* ((LOOKUP-TYPE (ASSOC TYPE *DEFINITION-PROTOTYPES* :TEST #'EQ))
      (IL:/RPLACD LOOKUP-TYPE (REMOVE NAME (CDR LOOKUP-TYPE)
                                           :KEY
                                           #'CAR))))))
(DEFDEFINER (DEF-DEFINE-TYPE (:PROTOTYPE (LAMBDA (NAME)
                                         (AND (SYMBOLP NAME)
                                               (DEF-DEFINE-TYPE ,NAME "Description string"))))
      IL:DEFINE-TYPES (NAME DESCRIPTION &KEY UNDEFINER &AUX (CHANGELST (INTERN (CONCATENATE 'STRING "CHANGED"
                                                                                          (STRING NAME)
                                                                                          "LST")
                                                                                          (SYMBOL-PACKAGE NAME))))
      "Define NAME as a new definition type"
      ;; This definition is a clean interface to a hokey implementation. It works even before the file package is loaded.
      `(PROGN (SETF (DOCUMENTATION ' ,NAME 'IL:DEFINE-TYPES)
                  ' ,DESCRIPTION)
             (PUSHNEW ' (,NAME X (IL:P IL:* (%DEFINE-TYPE-FILE-DEFINITIONS ' ,NAME 'X)))
                     IL:PRETTYDEFMACROS :TEST 'EQUAL)
             ;; the information about a type in the file package is split up into a number of different places. PRETTYTYPELIST contains a random
             ;; amount: the changelist is the variable whose top level value contains the list of changed items, and the description is a string used by
             ;; files? This is duplicated in the CL:DOCUMENTATION mechanism
             (PUSHNEW ' (,CHANGELST ,NAME ,DESCRIPTION)
                     IL:PRETTYTYPELIST :TEST 'EQUAL)
             (DEFGLOBALVAR ,CHANGELST NIL)
             ;; the definition hash table is where the definitions are really stored. Create an entry for this type. Note that definitions are compared using
             ;; CL:EQUAL so that names can be strings, lists, etc.
             (UNLESS (GETHASH ' ,NAME *DEFINITION-HASH-TABLE*)
                     (SETF (GETHASH ' ,NAME *DEFINITION-HASH-TABLE*)
                           (MAKE-HASH-TABLE :TEST 'EQUAL :SIZE 50 :REHASH-SIZE 50)))
             (PUSHNEW ' ,NAME IL:FILEPKGTYPES)
             (SETF (GET ' ,NAME 'IL:GETDEF)
                   '%DEFINE-TYPE-GETDEF)
             (SETF (GET ' ,NAME 'IL:DELDEF)
                   '%DEFINE-TYPE-DELDEF)
             (SETF (GET ' ,NAME 'IL:PUTDEF)
                   '%DEFINE-TYPE-PUTDEF)
             (SETF (GET ' ,NAME 'IL:FILEGETDEF)
                   '%DEFINE-TYPE-FILEGETDEF)
             (SETF (GET ' ,NAME 'IL:FILEPKGCONTENTS)
                   'IL:NILL)
             ,@(WHEN UNDEFINER
                 `(PUSHNEW ' ,UNDEFINER (GET ' ,NAME ' :UNDEFINERS))))))
(DEFDEFINER (DEFDEFINER (:NAME (LAMBDA (WHOLE)
                                (LET ((NAME (SECOND WHOLE))
                                      (IF (CONSP NAME)
                                          (CAR NAME)
                                          NAME))))
      (:PROTOTYPE (LAMBDA (NAME)
                  (AND (SYMBOLP NAME)
                      `(DEFDEFINER ,NAME , (IF (EQ (IL:EDITMODE)
                                                  'IL:SEDT)
                                              (SYMBOL-VALUE (INTERN "BASIC-GAP"
                                                                    "SEDT"))
                                              "Type") ,@(%MAKE-FUNCTION-PROTOTYPE)
                      )))
      (:UNDEFINER %DELETE-DEFINER)
      (:TEMPLATE (:NAME :TYPE :ARG-LIST :BODY)))
      IL:FUNCTIONS (NAME TYPE ARG-LIST &BODY BODY)
      (LET* ((OPTIONS (COND
                    ((CONSP NAME)
                     (PROG1 (CDR NAME)
                           (SETQ NAME (CAR NAME))))
                    (T NIL)))
            (NAME-FN NIL)
            (UNDEFINER NIL)
            (PROTOTYPE-FN NIL)
            (TEMPLATE NIL)
            (PRETTYMACRO NIL)
            (EDITDATE-OFFSET NIL))
            (DOLIST (OPT-LIST OPTIONS)
                    (CASE (CAR OPT-LIST)
                        ((:UNDEFINER) (SETQ UNDEFINER (CADR OPT-LIST)))
                        ((:NAME) (SETQ NAME-FN (CADR OPT-LIST)))
                        ((:PROTOTYPE) (SETQ PROTOTYPE-FN (CADR OPT-LIST)))
                        ((:TEMPLATE) (SETQ TEMPLATE (CADR OPT-LIST)))
                        ((:PRETTYPRINTMACRO) (SETQ PRETTYMACRO (CADR OPT-LIST)))
                        ((:EDITDATE-OFFSET) (SETQ EDITDATE-OFFSET (CADR OPT-LIST)))
                        (OTHERWISE (CERROR "Ignore the option" "Unrecognized option to DefDefiner: ~S" OPT-LIST))))))
      ;; Crap out now if junk in EDITDATE-OFFSET

```

```

(WHEN (AND EDITDATE-OFFSET (NOT (INTEGERP EDITDATE-OFFSET)))
      (ERROR ":EDITDATE-OFFSET must be an integer, not ~a" EDITDATE-OFFSET))
(MULTIPLE-VALUE-BIND (EXPANSION-FN DOC)
  (SI::EXPANSION-FUNCTION NAME ARG-LIST BODY)
  (UNLESS (OR TEMPLATE PRETTYMACRO (NOT (MEMBER '&BODY ARG-LIST)))
    ; Tell default prettyprinter where the body is
    (SETQ TEMPLATE (NCONC (IL:FOR X IL:IN ARG-LIST IL:UNTIL (EQ X '&BODY) IL:UNLESS (MEMBER X
      LAMBDA-LIST-KEYWORDS
    )
      IL:COLLECT NIL)
      (LIST :BODY)))
    (WHEN (AND (NULL (CAR TEMPLATE))
              (NULL NAME-FN))
      ; Name is in default place
      (SETF (CAR TEMPLATE) :NAME)))
  (LET ((EXPANDER-NAME (GET-DEFINER-NAME NAME "definition-expander-"))
        (NAME-FN-NAME (IF (CONSP NAME-FN)
                          (GET-DEFINER-NAME NAME "name-fn-"))))
    ` (PROGN (EVAL-WHEN (LOAD EVAL)
      (SETF (GET ',NAME ':DEFINER-FOR)
            ',TYPE)
      (PUSHNEW ',NAME (GET ',TYPE ':DEFINED-BY))
      (SETF (SYMBOL-FUNCTION ',EXPANDER-NAME)
            #',EXPANSION-FN)
      (SETF (GET ',NAME ':DEFINITION-EXPANDER)
            ',EXPANDER-NAME)
      ,@ (IF NAME-FN-NAME
            `((SETF (SYMBOL-FUNCTION ',NAME-FN-NAME)
                    #',NAME-FN))
            (SETF (GET ',NAME ':DEFINITION-NAME)
                  ',(OR NAME-FN-NAME NAME-FN 'SECOND))
            ,@ (AND UNDEFINER (LET ((UNDEFINER-FN-NAME (GET-DEFINER-NAME NAME
              "undefiner-fn-"))
                `((SETF (SYMBOL-FUNCTION ',UNDEFINER-FN-NAME)
                        #',UNDEFINER)
                  (PUSHNEW ',UNDEFINER-FN-NAME (GET ',NAME ':UNDEFINERS))))
              )
            ,@ (AND PROTOTYPE-FN (LET ((PROTOTYPE-FN-NAME (GET-DEFINER-NAME NAME
              "prototype-fn-"))
                `((SETF (SYMBOL-FUNCTION ',PROTOTYPE-FN-NAME)
                        #',PROTOTYPE-FN)
                  (ADD-PROTOTYPE-FN ',TYPE ',NAME ',PROTOTYPE-FN-NAME)))
              )
            ,@ (AND DOC `((SETF (DOCUMENTATION ',NAME 'FUNCTION)
                                ,DOC))
            ,@ (AND TEMPLATE `((SETF (GET ',NAME ':DEFINITION-PRINT-TEMPLATE)
                                      ',TEMPLATE))
            ,@ (AND EDITDATE-OFFSET `((SETF (GET ',NAME ':EDITDATE-OFFSET)
                                             ',EDITDATE-OFFSET))
            (PUSHNEW ',NAME ,@ (OR PRETTYMACRO 'PPRINT-DEFINER)
                    IL:PRETTYPRINTMACROS :TEST 'EQUAL))
            (DEFMACRO ,NAME (&WHOLE DEFINITION &ENVIRONMENT ENV)
              ` (DEFINER ',TYPE ',NAME ,DEFINITION ,ENV))))))

```

```

(DEFDEFINER (DEFDEFINER-VARIABLE-TYPE (:NAME (LAMBDA (WHOLE)
      (LET ((NAME (SECOND WHOLE))
            (IF (CONSP NAME)
                (CAR NAME)
                NAME))))
  (:PROTOTYPE
   (LAMBDA (NAME)
     (AND (SYMBOLP NAME)
          `(DEFDEFINER-VARIABLE-TYPE ,NAME
            , (IF (EQ (IL:EDITMODE)
                    (INTERN "SEdit" "SEdit"))
                (SYMBOL-VALUE (INTERN "BASIC-GAP" "SEdit"))
                "Type") ,@ (%MAKE-FUNCTION-PROTOTYPE))))
    (:UNDEFINER %DELETE-DEFINER)
    (:TEMPLATE (:NAME :TYPE :ARG-LIST :BODY)))
  IL:FUNCTIONS (NAME TYPES ARG-LIST &BODY BODY)

```

:: An extension to the DEFDEFINER universe, this allows the creation of definers that map to multiple file-package types. The test case, and the only case guaranteed to work, is DEFUN (which now must accept (DEFUN (SETF FOO)...), which needs to be stored as a SETFS file type).

```

(LET*
  ((OPTIONS (COND
    ((CONSP NAME)
     (PROG1 (CDR NAME)
       (SETQ NAME (CAR NAME))))
    (T NIL)))
   (NAME-FN NIL)
   (UNDEFINERS NIL)
   (PROTOTYPE-FNS NIL)
   (TEMPLATE NIL)
   (PRETTYMACRO NIL)
   (TYPE-DISCRIMINATOR NIL)
   (EXPANSION-FNS NIL)
   (DOCS NIL)

```



```

(EDITDATE-OFFSET NIL))
(DOLIST (OPT-LIST OPTIONS)
  (CASE (CAR OPT-LIST)
    ((:UNDEFINERS) (SETQ UNDEFINERS (CDR OPT-LIST)))
    ((:NAME) (SETQ NAME-FN (CADR OPT-LIST)))
    ((:PROTOTYPES) (SETQ PROTOTYPE-FNS (CDR OPT-LIST)))
    ((:TEMPLATE) (SETQ TEMPLATE (CADR OPT-LIST)))
    ((:PRETTYPRINTMACRO) (SETQ PRETTYMACRO (CADR OPT-LIST)))
    ((:TYPE-DISCRIMINATOR) (SETQ TYPE-DISCRIMINATOR (CADR OPT-LIST)))
    ((:EDITDATE-OFFSET) (SETQ EDITDATE-OFFSET (CADR OPT-LIST)))
    (OTHERWISE (CERROR "Ignore the option" "Unrecognized option to DefDefiner: ~S" OPT-LIST))))
(UNLESS (TYPE-DISCRIMINATOR (ERROR "DEFDEFINER-VARIABLE-TYPE must have a TYPE-DISCRIMINATOR"))
  ;; Crap out now if junk in EDITDATE-OFFSET
  (WHEN (AND EDITDATE-OFFSET (NOT (INTEGERP EDITDATE-OFFSET)))
    (ERROR ":EDITDATE-OFFSET must be an integer, not ~a" EDITDATE-OFFSET))
  (DOLIST (TYPE TYPES)
    (MULTIPLE-VALUE-BIND (EXPANSION-FN DOC)
      (SI::EXPANSION-FUNCTION NAME ARG-LIST (LET ((TB (GETF BODY TYPE)))
        (IF TB
          (LIST TB)
          (ERROR "No expansion-function for ~A" TYPE))))
      (SETF (GETF EXPANSION-FNS TYPE)
        EXPANSION-FN)
      (WHEN DOC
        (SETQ DOCS (CONCATENATE 'STRING DOCS (OR DOCS "
          ")
          (SYMBOL-NAME TYPE)
          ": " DOC))))))
    (UNLESS (OR TEMPLATE PRETTYMACRO (NOT (MEMBER '&BODY ARG-LIST)))
      ; Tell default prettyprinter where the body is
      (SETQ TEMPLATE (NCONC (IL:FOR X IL:IN ARG-LIST IL:UNTIL (EQ X '&BODY) IL:UNLESS (MEMBER X
        LAMBDA-LIST-KEYWORDS
        )
        IL:COLLECT NIL)
        (LIST :BODY)))
      (WHEN (AND (NULL (CAR TEMPLATE))
        (NULL NAME-FN))
        ; Name is in default place
        (SETF (CAR TEMPLATE)
          :NAME)))
      (LET
        ((NAME-FN-NAME (IF (CONSP NAME-FN)
          (GET-DEFINER-NAME NAME "name-fn-"))
          (TYPE-DISCRIMINATOR-NAME (GET-DEFINER-NAME NAME "type-discriminator-fn-"))
          `(PROGN
            (EVAL-WHEN
              (LOAD EVAL)
              (SETF (GET ',NAME ':DEFINER-FOR)
                ',TYPES)
              (SETF (SYMBOL-FUNCTION ',TYPE-DISCRIMINATOR-NAME)
                #' ,TYPE-DISCRIMINATOR)
              (SETF (GET ',NAME ':TYPE-DISCRIMINATOR)
                ',TYPE-DISCRIMINATOR-NAME)
              ,@(AND PROTOTYPE-FNS (MAPCAN #'(LAMBDA (TYPE)
                (LET ((PROTOTYPE-FN-NAME (GET-DEFINER-NAME NAME
                  (CONCATENATE 'STRING (SYMBOL-NAME
                    TYPE)
                    "-prototype-fn-"))))
                  `((SETF (SYMBOL-FUNCTION ',PROTOTYPE-FN-NAME)
                    #' ,(GETF PROTOTYPE-FNS TYPE))
                    (ADD-PROTOTYPE-FN ',TYPE ',NAME ',PROTOTYPE-FN-NAME))))
                TYPES))
              ,@(AND DOCS `((SETF (DOCUMENTATION ',NAME 'FUNCTION)
                ,DOCS)))
              ,@(AND TEMPLATE `((SETF (GET ',NAME ':DEFINITION-PRINT-TEMPLATE)
                ',TEMPLATE)))
              (SETF (GET ',NAME ':DEFINITION-NAME)
                ',(OR NAME-FN-NAME NAME-FN 'SECOND))
              ,@(MAPCAN #'(LAMBDA (TYPE)
                (LET ((EXPANDER-NAME (GET-DEFINER-NAME NAME (CONCATENATE 'STRING (SYMBOL-NAME TYPE)
                  "-definition-expander-"))
                  (EXPANSION-FN (GETF EXPANSION-FNS TYPE)))
                  `(PUSHNEW ',NAME (GET ',TYPE ':DEFINED-BY))
                  (SETF (SYMBOL-FUNCTION ',EXPANDER-NAME)
                    #' ,EXPANSION-FN)
                  (SETF (GETF (GET ',NAME ':DEFINITION-EXPANDER)
                    ',TYPE)
                    ',EXPANDER-NAME))))
                TYPES)
              ,@(IF NAME-FN-NAME
                `((SETF (SYMBOL-FUNCTION ',NAME-FN-NAME)
                  #' ,NAME-FN)))
              ,@(AND UNDEFINERS (MAPCAN #'(LAMBDA (TYPE)
                (WHEN (GETF UNDEFINERS TYPE)
                  (LET ((UNDEFINER-FN-NAME (GET-DEFINER-NAME NAME
                    (CONCATENATE 'STRING (SYMBOL-NAME
                      TYPE)

```

```

                                "-undefiner-fn-")))))
        \((SETF (SYMBOL-FUNCTION ',UNDEFINER-FN-NAME)
                #'(GETF UNDEFINERS TYPE))
           (PUSHNEW ',UNDEFINER-FN-NAME
                    (GETF (GET ',NAME ':UNDEFINERS)
                          ',TYPE))))))
        TYPES))
    ,@(AND EDITDATE-OFFSET \((SETF (GET ',NAME :EDITDATE-OFFSET)
                                   ,EDITDATE-OFFSET)))
      (PUSHNEW '(,NAME ,@(OR PRETTYMACRO 'PPRINT-DEFINER))
               IL:PRETTYPRINTMACROS :TEST 'EQUAL))
    (DEFMACRO ,NAME (&WHOLE DEFINITION &ENVIRONMENT ENV)
      \((DEFINER-VARIABLE-TYPE ',NAME ,DEFINITION ,ENV))))))

```

```

(DEFUN %EXPAND-DEFINER (DEFINER DEFINITION-WITHOUT-COMMENTS &OPTIONAL ENV)
  (FUNCALL (GET DEFINER :DEFINITION-EXPANDER)
            DEFINITION-WITHOUT-COMMENTS ENV))

```

```

(DEFUN %DEFINER-NAME (DEFINER DEFINITION-WITHOUT-COMMENTS)
  (FUNCALL (GET DEFINER :DEFINITION-NAME)
            DEFINITION-WITHOUT-COMMENTS))

```

:: The most commonly-used definers

```

(DEFDEFINER-VARIABLE-TYPE (DEFUN (:TYPE-DISCRIMINATOR (LAMBDA (WHOLE)
                                                       (LET ((NAME (SECOND WHOLE)))
                                                           (COND
                                                            ((SYMBOLP NAME)
                                                             'IL:FUNCTIONS)
                                                            ((CL::SETF-NAME-P NAME)
                                                             'IL:SETFS)
                                                            (T (ERROR "Can't determine type for
                                                                    DEFUN: ~s" NAME))))))
                             (:NAME (LAMBDA (WHOLE)
                                           (LET ((NAME (SECOND WHOLE)))
                                               (COND
                                                ((SYMBOLP NAME)
                                                 NAME)
                                                ((CL::SETF-NAME-P NAME)
                                                 (CADR NAME))
                                                (T (ERROR "Bad function-name for DEFUN: ~s" NAME))))))
                             (:PROTOTYPES IL:FUNCTIONS (LAMBDA (NAME)
                                                           \((DEFUN ,NAME ,@(%MAKE-FUNCTION-PROTOTYPE)))
                                                           IL:SETFS
                                                           (LAMBDA (NAME)
                                                             \((DEFUN (SETF ,NAME) ,@(%MAKE-FUNCTION-PROTOTYPE))))
                             (:TEMPLATE (:NAME :ARG-LIST :BODY))
                             (:EDITDATE-OFFSET 3))
  (IL:FUNCTIONS IL:SETFS) (NAME ARGS &BODY (BODY DECLS DOCUMENTATION)
                          &ENVIRONMENT ENV)
IL:FUNCTIONS
\ (PROGN (SETF (SYMBOL-FUNCTION ',NAME)
              #'(LAMBDA ,ARGS ,@DECLS (BLOCK ,NAME ,@BODY)))
        ,@(AND DOCUMENTATION \((SETF (DOCUMENTATION ',NAME 'FUNCTION)
                                     ,DOCUMENTATION))))
IL:SETFS

```

:: The form is (defun (setf foo) (store-var &rest args) body)

:: Strategy is to give the code a name with DEFUN-SETF-NAME. The name is stored on the :SETF-DEFUN property of the accessor. This name is there for convenience/documentation only; the name can't be reliably changed by smashing this property (i.e. (SETF (FDEFINITION '(SETF FOO)) #'BAR) essentially does (SETF (SYMBOL-FUNCTION (DEFUN-SETF-NAME 'FOO)) #'BAR); it does NOT change the :SETF-DEFUN property on :FOO).

```

(LET* ((REAL-NAME (SECOND NAME))
       (DEFUN-SETF-NAME (DEFUN-SETF-NAME REAL-NAME)))
  \ (PROGN (SETF (SYMBOL-FUNCTION ',DEFUN-SETF-NAME)
                #'(LAMBDA ,ARGS ,@DECLS (BLOCK ,REAL-NAME ,@BODY)))
          (SET-DEFUN-SETF ',REAL-NAME ',DEFUN-SETF-NAME)
          ,@(AND DOCUMENTATION \((SETF (DOCUMENTATION ',REAL-NAME 'SETF)
                                     ,DOCUMENTATION))))))

```

```

(DEFDEFINER (DEFININE (:PROTOTYPE (LAMBDA (NAME)
                                   (AND (SYMBOLP NAME)
                                        \ (DEFININE ,NAME ,@(%MAKE-FUNCTION-PROTOTYPE))))))
  (:TEMPLATE (:NAME :ARG-LIST :BODY)))
IL:FUNCTIONS (NAME ARG-LIST &BODY BODY &ENVIRONMENT ENV)

```

::: This is an INTERIM version of DEFININE. Eventually, this will just turn into a DEFUN and a PROCLAIM INLINE. (It says so right here.) If you're using this one, DO NOT make any recursive calls in the body of the DEFININE. If you do, the compiler will run forever trying to expand the optimizer... Once the INLINE version gets working (in the PavCompiler only) that restriction will be lifted.

```

(MULTIPLE-VALUE-BIND (CODE DECLS DOC)
  (PARSE-BODY BODY ENV T)
  (LET ((NEW-LAMBDA \(' LAMBDA ,ARG-LIST ,@DECLS (BLOCK ,NAME ,@CODE)))
        \ (PROGN (DEFUN ,NAME ,ARG-LIST

```

```

      ,@BODY)
      (DEFOPTIMIZER ,NAME ,(PACK (LIST "definline-" NAME)
      (SYMBOL-PACKAGE NAME))
      (&REST ARGS)
      (CONS ',NEW-LAMBDA ARGS))))))

(DEFDEFINER (DEFMACRO (:PROTOTYPE (LAMBDA (NAME)
      (AND (SYMBOLP NAME)
      `(DEFMACRO ,NAME ,@(%MAKE-FUNCTION-PROTOTYPE))))))
      (:UNDEFINER (LAMBDA (NAME)
      (REMPROP NAME 'IL:ARGNAMES)))
      (:TEMPLATE (:NAME :ARG-LIST :BODY)))
      IL:FUNCTIONS (NAME DEFMACRO-ARGS &BODY DEFMACRO-BODY)
      (UNLESS (AND NAME (SYMBOLP NAME))
      (ERROR "Illegal name used in DEFMACRO: ~S" NAME))
      (LET
      ((CMACRONAME (PACK (LIST "expand-" NAME)
      (SYMBOL-PACKAGE NAME))))
      (MULTIPLE-VALUE-BIND (EXPANSION-FN DOC-STRING)
      (SI::EXPANSION-FUNCTION NAME DEFMACRO-ARGS DEFMACRO-BODY)
      `(PROGN (SETF (SYMBOL-FUNCTION ',CMACRONAME)
      #' ,EXPANSION-FN)
      (SETF (MACRO-FUNCTION ',NAME)
      ',CMACRONAME)
      ,@(AND DOC-STRING `((SETF (DOCUMENTATION ',NAME 'FUNCTION)
      ,DOC-STRING)))
      ,@(WHEN COMPILER::*NEW-COMPILER-IS-EXPANDING*
      `((SETF (GET ',NAME 'IL:ARGNAMES)
      ',(MAPCAR #'(LAMBDA (ARG)
      (IF (MEMBER ARG LAMBDA-LIST-KEYWORDS)
      ARG
      (PRIN1-TO-STRING ARG))))
      (IL:\\SIMPLIFY.CL.ARGLIST DEFMACRO-ARGS))))))))))

(DEFDEFINER (DEFVAR (:PROTOTYPE (LAMBDA (NAME)
      (AND (SYMBOLP NAME)
      `(DEFVAR ,NAME))))))
      IL:VARIABLES (NAME &OPTIONAL (INITIAL-VALUE NIL IVP)
      DOCUMENTATION)
      `(PROGN (PROCLAIM '(SPECIAL ,NAME))
      ,@(AND IVP `(OR (BOUNDP ',NAME)
      (SETQ ,NAME ,INITIAL-VALUE))))
      ,@(AND DOCUMENTATION `(SETF (DOCUMENTATION ',NAME 'VARIABLE)
      ,DOCUMENTATION))))))

(DEFDEFINER (DEFPARAMETER (:PROTOTYPE (LAMBDA (NAME)
      (AND (SYMBOLP NAME)
      `(DEFPARAMETER ,NAME "Value"
      "Documentation string")))))
      IL:VARIABLES (NAME INITIAL-VALUE &OPTIONAL DOCUMENTATION)
      `(PROGN (PROCLAIM '(SPECIAL ,NAME))
      (SETQ ,NAME ,INITIAL-VALUE)
      ,@(AND DOCUMENTATION `(SETF (DOCUMENTATION ',NAME 'VARIABLE)
      ,DOCUMENTATION))))))

(DEFDEFINER (DEFCONSTANT (:PROTOTYPE (LAMBDA (NAME)
      (AND (SYMBOLP NAME)
      `(DEFCONSTANT ,NAME "Value"
      "Documentation string")))))
      IL:VARIABLES (NAME VALUE &OPTIONAL DOCUMENTATION)
      `(PROGN ,@(IF (CONSTANTP NAME)
      `(SET-CONSTANTP ',NAME NIL))
      (SETQ ,NAME ,VALUE)
      (PROCLAIM '(SI::CONSTANT ,NAME))
      ,@(AND DOCUMENTATION `(SETF (DOCUMENTATION ',NAME 'VARIABLE)
      ,DOCUMENTATION))))))

(DEFDEFINER (DEFGLOBALVAR (:PROTOTYPE (LAMBDA (NAME)
      (AND (SYMBOLP NAME)
      `(DEFGLOBALVAR ,NAME))))))
      IL:VARIABLES (NAME &OPTIONAL (INITIAL-VALUE NIL IVP)
      DOCUMENTATION)
      ;; Use IL:SETQ here or the INIT dies.
      `(PROGN (PROCLAIM '(GLOBAL ,NAME))
      ,@(AND IVP `(OR (BOUNDP ',NAME)
      (SETQ ,NAME ,INITIAL-VALUE))))
      ,@(AND DOCUMENTATION `(SETF (DOCUMENTATION ',NAME 'VARIABLE)
      ,DOCUMENTATION))))))

(DEFDEFINER (DEFGLOBALPARAMETER (:PROTOTYPE (LAMBDA (NAME)
      (AND (SYMBOLP NAME)
      `(DEFGLOBALPARAMETER ,NAME "Value"
      "Documentation string")))))
      IL:VARIABLES (NAME INITIAL-VALUE &OPTIONAL DOCUMENTATION)
      `(PROGN (PROCLAIM '(GLOBAL ,NAME))
      (SETQ ,NAME ,INITIAL-VALUE)

```

```
,@(AND DOCUMENTATION `((SETF (DOCUMENTATION ',NAME 'VARIABLE)
,DOCUMENTATION))))
```

:: Here so that the evaluator can be in the init without definers being in the init.

```
(DEF-DEFINE-TYPE IL:SPECIAL-FORMS "Common Lisp special forms"
:UNDEFINER %REMOVE-SPECIAL-FORM)
```

```
(DEFUN %REMOVE-SPECIAL-FORM (X)
(IL:/REMPROP X 'IL:SPECIAL-FORM))
```

```
(DEFDEFINER (DEFINE-SPECIAL-FORM (:TEMPLATE (:NAME :ARG-LIST :BODY))) IL:SPECIAL-FORMS (NAME ARGS &REST
BODY)
(COND
((NULL BODY)
(ASSERT (SYMBOLP NAME)
NIL "Ill-formed short DEFINE-SPECIAL-FORM; ~S is not a symbol." ARGS)
` (SETF (GET ',NAME 'IL:SPECIAL-FORM)
',ARGS))
(T (LET ((SF (INTERN (CONCATENATE 'STRING "interpret-" (STRING NAME))
(SYMBOL-PACKAGE NAME))))
(MULTIPLE-VALUE-BIND (PARSED-BODY DECLS DOC)
(IL:PARSE-DEFMACRO ARGS '$$TAIL BODY NAME NIL :PATH '$$TAIL :ENVIRONMENT '$$ENV)
` (PROGN (SETF (SYMBOL-FUNCTION ',SF)
#' (LAMBDA ($$TAIL $$ENV)
,@DECLS
(BLOCK ,NAME ,PARSED-BODY)))
(SETF (GET ',NAME 'IL:SPECIAL-FORM)
',SF))))))
```

:: Form for defining interpreters of special forms  
:: Don't note changes to these properties/variables

```
(IL:PUTPROPS IL:MACRO-FN IL:PROPTYPE IL:FUNCTIONS)
(IL:PUTPROPS :UNDEFINERS IL:PROPTYPE IGNORE)
(IL:PUTPROPS IL:UNDEFINERS IL:PROPTYPE IGNORE)
(IL:PUTPROPS :DEFINER-FOR IL:PROPTYPE IGNORE)
(IL:PUTPROPS IL:DEFINER-FOR IL:PROPTYPE IGNORE)
(IL:PUTPROPS :DEFINED-BY IL:PROPTYPE IGNORE)
(IL:PUTPROPS IL:DEFINED-BY IL:PROPTYPE IGNORE)
(IL:PUTPROPS :DEFINITION-NAME IL:PROPTYPE IGNORE)
(IL:PUTPROPS IL:DEFINITION-NAME IL:PROPTYPE IGNORE)
```

:: Templates for definers not defined here. These should really be where they're defined.

```
(IL:PUTPROPS DEFCOMMAND :DEFINITION-PRINT-TEMPLATE (:NAME :ARG-LIST :BODY))
(IL:PUTPROPS DEFINE-CONDITION :DEFINITION-PRINT-TEMPLATE (:NAME :VALUE :BODY))
(IL:PUTPROPS DEFINE-MODIFY-MACRO :DEFINITION-PRINT-TEMPLATE (:NAME :ARG-LIST))
(IL:PUTPROPS DEFINE-SETF-METHOD :DEFINITION-PRINT-TEMPLATE (:NAME NIL NIL :BODY))
(IL:PUTPROPS DEFSETF :DEFINITION-PRINT-TEMPLATE (:NAME :ARG-LIST :ARG-LIST :BODY))
(IL:PUTPROPS DEFSTRUCT :DEFINITION-PRINT-TEMPLATE (:NAME :BODY))
(IL:PUTPROPS DEFTYPE :DEFINITION-PRINT-TEMPLATE (:NAME NIL :BODY))
```

:: Arrange for the correct compiler to be used.

```
(IL:PUTPROPS IL:CMLDEFFER IL:FILETYPE :COMPILE-FILE)
(IL:PUTPROPS IL:CMLDEFFER IL:MAKEFILE-ENVIRONMENT (:READTABLE "XCL" :PACKAGE "XCL"))
(IL:PUTPROPS IL:CMLDEFFER IL:COPYRIGHT ("Venue & Xerox Corporation" 1986 1900 1987 1988 1990 1992))
```

---

**FUNCTION INDEX**

%DEFINE-TYPE-DELDEF .....	4	%DEFINER-NAME .....	10	PPRINT-DEFINER .....	2
%DEFINE-TYPE-FILE-DEFINITIONS .....	4	%DELETE-DEFINER .....	6	PPRINT-DEFINER-FITP .....	3
%DEFINE-TYPE-FILEGETDEF .....	4	%EXPAND-DEFINER .....	10	PPRINT-DEFINER-RECURSE .....	3
%DEFINE-TYPE-GETDEF .....	4	%REMOVE-SPECIAL-FORM .....	12	REMOVE-COMMENTS .....	2
%DEFINE-TYPE-PUTDEF .....	5	SI::EXPANSION-FUNCTION .....	5		
%DEFINE-TYPE-SAVE-DEFN .....	4	GET-DEFINER-NAME .....	6		

---

**PROPERTY INDEX**

IL:CMLDEFFER .....	12	:DEFINED-BY .....	12	IL:DEFINITION-NAME .....	12	:UNDEFINERS .....	12
DEFCOMMAND .....	12	IL:DEFINED-BY .....	12	DEFSETF .....	12	IL:UNDEFINERS .....	12
DEFINE-CONDITION .....	12	:DEFINER-FOR .....	12	DEFSTRUCT .....	12		
DEFINE-MODIFY-MACRO .....	12	IL:DEFINER-FOR .....	12	DEFTYPE .....	12		
DEFINE-SETF-METHOD .....	12	:DEFINITION-NAME .....	12	IL:MACRO-FN .....	12		

---

**DEFINER INDEX**

DEF-DEFINE-TYPE .....	7	DEFGLOBALPARAMETER .....	11	DEFMACRO .....	11
DEFCONSTANT .....	11	DEFGLOBALVAR .....	11	DEFPARAMETER .....	11
DEFDEFINER .....	7	DEFINE-SPECIAL-FORM .....	12	DEFVAR .....	11
DEFDEFINER-VARIABLE-TYPE .....	8	DEFINLINE .....	10		

---

**MACRO INDEX**

DEFINER .....	6	SI::MACRO-FUNCALL .....	6	WITHOUT-FILEPKG .....	6
DEFINER-VARIABLE-TYPE .....	6	NAMED-PROGN .....	6	IL:WITHOUT-FILEPKG .....	6

---

**DEFINE-TYPE INDEX**

IL:DEFINE-TYPES .....	5	IL:FUNCTIONS .....	5	IL:SPECIAL-FORMS .....	12	IL:VARIABLES .....	5
-----------------------	---	--------------------	---	------------------------	----	--------------------	---

---

**VARIABLE INDEX**

IL:*REMOVE-INTERLISP-COMMENTS* .....	4
--------------------------------------	---

---