File created: 13-Jun-90 16:19:18 **{PELE:MV:ENVOS}<LISPCORE>SOURCES>CMLSETF.;6**

changes to: (IL:FUNCTIONS GET-SETF-METHOD)

previous date: 11-Jun-90 15:06:52 {PELE:MV:ENVOS}<LISPCORE>SOURCES>CMLSETF.;5

Read Table: XCL

Package: LISP

Format: XCCS


```
(IL:RPAQQ IL:CMLSETFCOMS
          ((IL:FUNCTIONS GET-SETF-METHOD GET-SIMPLE-SETF-METHOD GET-SETF-METHOD-MULTIPLE-VALUE)
           (IL:DEFINE-TYPES IL:SETFS)
           (IL:FUNCTIONS DEFSETF DEFINE-MODIFY-MACRO DEFINE-SETF-METHOD)
           (IL:COMS
                  ;; Support for defstruct and friends
                  (IL:FUNCTIONS DEFINE-SHARED-SETF-MACRO DEFINE-SHARED-SETF GET-SHARED-SETF-METHOD))
           (IL:FUNCTIONS SETF SETF-ERROR)
           (IL:FUNCTIONS PSETF SHIFTF ROTATEF POP REMF)
           (IL:FUNCTIONS INCF DECF)
           (IL:FUNCTIONS MAYBE-MAKE-BINDING-FORM COUNT-OCCURRENCES)
           (IL:FUNCTIONS PUSH PUSHNEW)
           (IL:SETFS CAR CDR CAAAAR CAAADR CAAAR CAADAR CAADDR CAADR CAAR CADAAR CADADR CADAR CADDAR CADDDR
                  CADDR CADR CDAAAR CDAADR CDAAR CDADAR CDADDR CDADR CDAR CDDAAR CDDADR CDDAR CDDDAR CDDDDR
                  CDDDR CDDR FIRST SECOND THIRD FOURTH FIFTH SIXTH SEVENTH EIGHTH NINTH TENTH REST NTHCDR NTH
                  GETF APPLY LDB MASK-FIELD CHAR-BIT THE)
           (IL:COMS                                              ; Some IL setfs, for no especially good reason
                  (IL:SETFS IL:GETHASH)
                  (IL:FUNCTIONS IL:%SET-IL-GETHASH))
           (IL:PROP IL:PROPTYPE :SETF-METHOD-EXPANDER :SETF-INVERSE :SHARED-SETF-INVERSE)
           (IL:PROP (IL:FILETYPE IL:MAKEFILE-ENVIRONMENT)
                  IL:CMLSETF)
           (IL:DECLARE\: IL:DONTEVAL@LOAD IL:DOEVAL@COMPILE IL:DONTCOPY IL:COMPILERVARS (IL:ADDVARS (IL:NLAMA)
                                                                                                 (IL:NLAML)
                                                                                                 (IL:LAMA)))))

(DEFUN GET-SETF-METHOD (FORM &OPTIONAL ENVIRONMENT)
   (LET (TEMP)
        (COND
           ((SYMBOLP FORM)
            ;; Symbols have a simple, constant SETF method.
            (VALUES NIL NIL (LIST (SETQ TEMP (IL:GENSYM)))
                   '(SETQ ,FORM ,TEMP)
                   FORM))
           ((NOT (CONSP FORM))                                   ; Syntax error
            (SETF-ERROR FORM))
           ((SETQ TEMP (IL:LOCAL-MACRO-FUNCTION (CAR FORM)
                              ENVIRONMENT))
            ;; Lexical macros cannot have SETF methods defined upon them, so just expand this and try again.
            (GET-SETF-METHOD (FUNCALL TEMP FORM ENVIRONMENT)
                   ENVIRONMENT))
           ((SETQ TEMP (OR (GET (CAR FORM)
                                ':SETF-INVERSE)
                           (GET (CAR FORM)
                                'IL:SETF-INVERSE)
                           (GET (CAR FORM)
                                'IL:SETFN)))
            (GET-SIMPLE-SETF-METHOD FORM TEMP))
           ((SETQ TEMP (GET (CAR FORM)
                            ':SHARED-SETF-INVERSE))
            (GET-SHARED-SETF-METHOD FORM TEMP))
           ((SETQ TEMP (OR (GET (CAR FORM)
                                ':SETF-METHOD-EXPANDER)
                           (GET (CAR FORM)
                                'IL:SETF-METHOD-EXPANDER)))
            ;; Do check number of the Store Variables
            (MULTIPLE-VALUE-BIND (TEMPS VALUES STORES SETTER GETTER)
                (FUNCALL TEMP FORM ENVIRONMENT)
              (WHEN (/= (LENGTH STORES)
                        1)
                  (WARN "SETF method contains more than one store variable. Only top of the elements was
                        accepted.")
                  (SETQ STORES (LIST (CAR STORES))))
              (VALUES TEMPS VALUES STORES SETTER GETTER)))
           (T (MULTIPLE-VALUE-BIND (EXPANSION DONE)
                  (MACROEXPAND-1 FORM ENVIRONMENT)
                (IF (AND DONE (NOT (EQ EXPANSION FORM)))
                    (GET-SETF-METHOD EXPANSION ENVIRONMENT)
                    (SETF-ERROR (CAR FORM)
```

```
                          FORM)))))))

(DEFUN GET-SIMPLE-SETF-METHOD (FORM SETF-INVERSE)
    ;; Produce SETF method for a form that has a setf-inverse.  Five values to return are: temp vars, values to bind them to, store temp var, store form,
    ;; access form; the latter two are expressions that can use any of them temp vars.
    (LET ((NEW-VAR (IL:GENSYM))
          VARS VALS ARGS SETF-INVERSE-FORM GET-FORM)
         (SETQ ARGS (MAPCAR #'(LAMBDA (ARG)
                                (COND
                                   ((IF (CONSP ARG)
                                        (EQ (CAR ARG)
                                            'QUOTE)
                                        (CONSTANTP ARG))
                                     ;; We don't need gensym for this constant argument.  The test is a little more conservative than
                                     ;; CL:CONSTANTP because it's not obvious that it's ok to evaluate a "constant expression" multiple
                                     ;; times and get the same EQ object every time.
                                     ARG)
                                    (T ;; Anything else might be side-effected, so will need to bind
                                       (PUSH ARG VALS)
                                       (LET ((G (IL:GENSYM)))
                                            (PUSH G VARS)
                                            G))))
                            (CDR FORM)))
         (SETQ SETF-INVERSE-FORM (MACROEXPAND-1 `(,SETF-INVERSE ,@ARGS ,NEW-VAR)))
         (SETQ GET-FORM (MACROEXPAND-1 `(,(CAR FORM)
                                         ,@ARGS)))
        ;; ARGS is now the arguments to FORM with gensyms substituted for the non-constant expressions
        (VALUES (SETQ VARS (NREVERSE VARS))
                (SETQ VALS (NREVERSE VALS))
                (LIST NEW-VAR)
                SETF-INVERSE-FORM GET-FORM)))

(DEFUN GET-SETF-METHOD-MULTIPLE-VALUE (FORM &OPTIONAL ENVIRONMENT)
   (LET (TEMP)
        (COND
           ((SYMBOLP FORM)
            ;; Symbols have a simple, constant SETF method.
            (VALUES NIL NIL (LIST (SETQ TEMP (IL:GENSYM)))
                    `(SETQ ,FORM ,TEMP)
                    FORM))
           ((NOT (CONSP FORM))                                    ; Syntax error
            (SETF-ERROR FORM))
           ((SETQ TEMP (IL:LOCAL-MACRO-FUNCTION (CAR FORM)
                                 ENVIRONMENT))
            ;; Lexical macros cannot have SETF methods defined upon them, so just expand this and try again.
            (GET-SETF-METHOD (FUNCALL TEMP FORM ENVIRONMENT)
                    ENVIRONMENT))
           ((SETQ TEMP (OR (GET (CAR FORM)
                                ':SETF-INVERSE)
                           (GET (CAR FORM)
                                'IL:SETF-INVERSE)
                           (GET (CAR FORM)
                                'IL:SETFN)))
            (GET-SIMPLE-SETF-METHOD FORM TEMP))
           ((SETQ TEMP (GET (CAR FORM)
                            ':SHARED-SETF-INVERSE))
            (GET-SHARED-SETF-METHOD FORM TEMP))
           ((SETQ TEMP (OR (GET (CAR FORM)
                                ':SETF-METHOD-EXPANDER)
                           (GET (CAR FORM)
                                'IL:SETF-METHOD-EXPANDER)))
            ;; Does not check the number of Store Variables.
            (FUNCALL TEMP FORM ENVIRONMENT))
           (T (MULTIPLE-VALUE-BIND (EXPANSION DONE)
                  (MACROEXPAND-1 FORM ENVIRONMENT)
                (IF (AND DONE (NOT (EQ EXPANSION FORM)))
                    (GET-SETF-METHOD EXPANSION ENVIRONMENT)
                    (SETF-ERROR (CAR FORM)
                            FORM)))))))

(XCL:DEF-DEFINE-TYPE IL:SETFS "Common Lisp SETF definitions")

(XCL:DEFDEFINER (DEFSETF (:PROTOTYPE (LAMBDA (NAME)
                                       (AND (SYMBOLP NAME)
                                            `(DEFSETF ,NAME "Inverse function")))))
    IL:SETFS (NAME &REST REST &ENVIRONMENT ENV)
```

```
;;; Associates a SETF update function or macro with the specified access function or macro

   (COND
      ((NULL REST)
       (ERROR "No body for DEFSETF of ~A" NAME))
      ((AND (LISTP (CAR REST))
            (CDR REST)
            (LISTP (CADR REST)))
       ;; The complex form:
       ;; (defsetf access-fn args (store-var) {decl | doc}* {form}*)
       (XCL:DESTRUCTURING-BIND
        (ARG-LIST (STORE-VAR &REST OTHERS)
                  &BODY BODY)
        REST
        (IF OTHERS (CERROR "Ignore the extra items in the list." "Currently only one new-value variable is
                           allowed in DEFSETF."))
        (LET ((WHOLE-VAR (XCL:PACK (LIST NAME "-setf-form")
                                   (SYMBOL-PACKAGE NAME)))
              (ENVIRONMENT (XCL:PACK (LIST NAME "-setf-env")
                                     (SYMBOL-PACKAGE NAME)))
              (EXPANDER (XCL:PACK (LIST NAME "-setf-expander")
                                  (SYMBOL-PACKAGE NAME))))
             (MULTIPLE-VALUE-BIND (CODE DECLS DOC)
                 (IL:PARSE-DEFMACRO ARG-LIST WHOLE-VAR BODY NAME ENV :ENVIRONMENT ENVIRONMENT)
               `(PROGN (EVAL-WHEN (EVAL COMPILE LOAD)
                                  (SETF (SYMBOL-FUNCTION ',EXPANDER)
                                        #'(LAMBDA (ACCESS-FORM ,ENVIRONMENT)
                                                  (LET* ((DUMMIES (MAPCAR #'(LAMBDA (X)
                                                                                    (IL:GENSYM))
                                                                          (CDR ACCESS-FORM)))
                                                                 (,WHOLE-VAR (CONS (CAR ACCESS-FORM)
                                                                                   DUMMIES))
                                                                 (,STORE-VAR (IL:GENSYM)))
                                                                (VALUES DUMMIES (CDR ACCESS-FORM)
                                                                        (LIST ,STORE-VAR)
                                                                        (BLOCK ,NAME ,CODE)
                                                                        ,WHOLE-VAR))))
                                  (SET-SETF-METHOD-EXPANDER ',NAME ',EXPANDER))
                         ,@(AND DOC `((SETF (DOCUMENTATION ',NAME 'SETF)
                                            ,DOC))))))))
      ((SYMBOLP (CAR REST))
       ;; The short form:
       ;; (defsetf access-fn update-fn [doc])
       (LET ((UPDATE-FN (CAR REST))
             (DOC (CADR REST)))
            `(PROGN (EVAL-WHEN (LOAD COMPILE EVAL)
                               (SET-SETF-INVERSE ',NAME ',UPDATE-FN))
                    ,@(AND DOC `((SETF (DOCUMENTATION ',NAME 'SETF)
                                       ,DOC))))))
      (T (ERROR "Ill-formed DEFSETF for ~S." NAME))))

(XCL:DEFDEFINER (DEFINE-MODIFY-MACRO (:PROTOTYPE (LAMBDA (NAME)
                                                         (AND (SYMBOLP NAME)
                                                              `(DEFINE-MODIFY-MACRO ,NAME ,@(
                                                                                             XCL::%MAKE-FUNCTION-PROTOTYPE
                                                                                                                    ))))))
    IL:FUNCTIONS (NAME LAMBDA-LIST FUNCTION &OPTIONAL DOC-STRING)
   "Creates a new read-modify-write macro like PUSH or INCF."
   (LET ((OTHER-ARGS NIL)
         (REST-ARG NIL))
        (DO ((LL LAMBDA-LIST (CDR LL))
             (ARG NIL))
            ((NULL LL))
          (SETQ ARG (CAR LL))
          (COND
             ((EQ ARG '&OPTIONAL))
             ((EQ ARG '&REST)
              (SETQ REST-ARG (CADR LL))
              (RETURN NIL))
             ((SYMBOLP ARG)
              (PUSH ARG OTHER-ARGS))
             (T (PUSH (CAR ARG)
                      OTHER-ARGS))))
        (SETQ OTHER-ARGS (NREVERSE OTHER-ARGS))
        `(DEFMACRO ,NAME (SI::%$$MODIFY-MACRO-FORM ,@LAMBDA-LIST &ENVIRONMENT SI::%$$MODIFY-MACRO-ENVIRONMENT)
            ,DOC-STRING (MULTIPLE-VALUE-BIND (DUMMIES VALS NEWVALS SETTER GETTER)
                            (GET-SETF-METHOD SI::%$$MODIFY-MACRO-FORM SI::%$$MODIFY-MACRO-ENVIRONMENT)
                          `(,'LET* (,@(MAPCAR #'LIST DUMMIES VALS)
                                    (,(CAR NEWVALS)
                                     ,,(IF REST-ARG
                                           `(LIST* ',FUNCTION GETTER ,@OTHER-ARGS ,REST-ARG)
                                           `(LIST ',FUNCTION GETTER ,@OTHER-ARGS))))
                                ,SETTER)))))
```

```
(XCL:DEFDEFINER (DEFINE-SETF-METHOD (:PROTOTYPE (LAMBDA (NAME)
                                                  (AND (SYMBOLP NAME)
                                                       `(DEFINE-SETF-METHOD ,NAME ("Arg list") "Body")
                                                       ))))
    IL:SETFS (NAME LAMBDA-LIST &ENVIRONMENT ENV &BODY BODY)
   (LET ((WHOLE (XCL:PACK (LIST "whole-" NAME)
                          (SYMBOL-PACKAGE NAME)))
         (ENVIRONMENT (XCL:PACK (LIST "env-" NAME)
                                (SYMBOL-PACKAGE NAME)))
         (EXPANDER (XCL:PACK (LIST "setf-expander-" NAME)
                             (SYMBOL-PACKAGE NAME))))
        (MULTIPLE-VALUE-BIND (NEWBODY LOCAL-DECS DOC)
            (IL:PARSE-DEFMACRO LAMBDA-LIST WHOLE BODY NAME ENV :ENVIRONMENT ENVIRONMENT :ERROR-STRING "Setf
                   expander for ~S cannot be called with ~S args.")
          `(EVAL-WHEN (EVAL COMPILE LOAD)
                  (DEFUN ,EXPANDER (,WHOLE ,ENVIRONMENT)
                     ,@LOCAL-DECS (BLOCK ,NAME ,NEWBODY))
                  (SET-SETF-METHOD-EXPANDER ',NAME ',EXPANDER)
                  ,@(AND DOC `((SETF (DOCUMENTATION ',NAME 'SETF)
                                     ,DOC)))))))
```

;; Support for defstruct and friends

```
(XCL:DEFDEFINER DEFINE-SHARED-SETF-MACRO IL:FUNCTIONS (NAME ACCESSOR ARG-LIST STORE-VAR &BODY BODY
                                                       &ENVIRONMENT ENV)
```

;;; Defines a shared SETF update function for a family of accessores -- used by defstruct

```
   (IF (NOT (AND (CONSP STORE-VAR)
                 (EQ 1 (LENGTH STORE-VAR))))
       (ERROR "Store-var should be a list of one element: ~s" STORE-VAR))
   (MULTIPLE-VALUE-BIND (CODE DECLS DOC)
       (XCL:PARSE-BODY BODY ENV T)
     `(DEFMACRO ,NAME (,ACCESSOR ,@ARG-LIST ,@STORE-VAR)
        ,@DOC ,@DECLS ,@CODE)))
```

```
(XCL:DEFDEFINER DEFINE-SHARED-SETF IL:SETFS (NAME SHARED-EXPANDER &OPTIONAL DOC)
```

;;; Associates a shared SETF update macro with the specified accessor function -- used by defstruct

```
   `(PROGN (EVAL-WHEN (LOAD COMPILE EVAL)
                 (SET-SHARED-SETF-INVERSE ',NAME ',SHARED-EXPANDER))
           ,@(AND DOC `((SETF (DOCUMENTATION ',NAME 'SETF)
                              ,DOC)))))
```

```
(DEFUN GET-SHARED-SETF-METHOD (FORM SHARED-SETF-INVERSE)
```

;; Produce SETF method for a form that has a shared-setf-inverse. Five values to return are: temp vars, values to bind them to, store temp var, store
;; form, access form; the latter two are expressions that can use any of them temp vars.

```
   (LET ((NEW-VAR (IL:GENSYM))
         VARS VALS ARGS SHARED-SETF-INVERSE-FORM GET-FORM)
        (SETQ ARGS (MAPCAR #'(LAMBDA (ARG)
                                (COND
                                   ((IF (CONSP ARG)
                                        (EQ (CAR ARG)
                                            'QUOTE)
                                        (CONSTANTP ARG))
```

;; We don't need gensym for this constant argument. The test is a little more conservative than
;; CL:CONSTANTP because it's not obvious that it's ok to evaluate a "constant expression" multiple
;; times and get the same EQ object every time.

```
                                      ARG)
                                     (T
```

;; Anything else might be side-effected, so will need to bind

```
                                        (PUSH ARG VALS)
                                        (LET ((G (IL:GENSYM)))
                                             (PUSH G VARS)
                                             G))))
                             (CDR FORM)))
        (SETQ SHARED-SETF-INVERSE-FORM (MACROEXPAND-1 `(,SHARED-SETF-INVERSE ,(CAR FORM)
                                                                             ,@ARGS
                                                                             ,NEW-VAR)))
        (SETQ GET-FORM (MACROEXPAND-1 `(,(CAR FORM)
                                        ,@ARGS)))
```

;; ARGS is now the arguments to FORM with gensyms substituted for the non-constant expressions

```
        (VALUES (SETQ VARS (NREVERSE VARS))
                (SETQ VALS (NREVERSE VALS))
                (LIST NEW-VAR)
                SHARED-SETF-INVERSE-FORM GET-FORM)))
```

```
(DEFMACRO SETF (PLACE NEW-VALUE &REST OTHERS &ENVIRONMENT ENV)
```

;;; Takes pairs of arguments like SETQ. The first is a place and the second is the value that is supposed to go into that place. Returns the last value. The
;;; place argument may be any of the access forms for which SETF knows a corresponding setting form.

;;; We short-circuit the normal SETF-method mechanism for two very common special cases, so as to produce much simpler and more efficient code.
;;; The two cases are symbols and forms with simple inverses.

```
    (COND
       (OTHERS `(PROGN (SETF ,PLACE ,NEW-VALUE)
                       (SETF ,@OTHERS)))
       (T (PROG (TEMP)
            LP  (COND
                   ((SYMBOLP PLACE)
                    (RETURN `(SETQ ,PLACE ,NEW-VALUE)))
                   ((NOT (CONSP PLACE))
                    (SETF-ERROR PLACE))
                   ((SETQ TEMP (IL:LOCAL-MACRO-FUNCTION (CAR PLACE)
                                                        ENV))
                    ;; Before looking for an inverse, we have to macroexpand until it isn't a reference to a lexical macro, since those can't have
                    ;; SETF methods.
                    (SETQ PLACE (FUNCALL TEMP PLACE ENV)))
                   ((AND (SYMBOLP (CAR PLACE))
                         (SETQ TEMP (OR (GET (CAR PLACE)
                                             ':SETF-INVERSE)
                                        (GET (CAR PLACE)
                                             'IL:SETF-INVERSE)
                                        (GET (CAR PLACE)
                                             'IL:SETFN))))
                    (RETURN `(,TEMP ,@(CDR PLACE)
                                    ,NEW-VALUE)))
                   ((AND (SYMBOLP (CAR PLACE))
                         (SETQ TEMP (GET (CAR PLACE)
                                         ':SHARED-SETF-INVERSE)))
                    (RETURN `(,TEMP ,(CAR PLACE)
                                    ,@(CDR PLACE)
                                    ,NEW-VALUE)))
                   ((OR (GET (CAR PLACE)
                             ':SETF-METHOD-EXPANDER)
                        (GET (CAR PLACE)
                             'IL:SETF-METHOD-EXPANDER))
                    ;; General setf hair
                    (RETURN (MULTIPLE-VALUE-BIND (DUMMIES VALS NEWVALS SETTER GETTER)
                                (GET-SETF-METHOD PLACE ENV)
                              `(,'LET* (,@(MAPCAR #'LIST DUMMIES VALS)
                                        (,(CAR NEWVALS)
                                         ,NEW-VALUE))
                                ,SETTER))))
                   (T                                          ; Try macro expanding
                      (MULTIPLE-VALUE-BIND (EXPANSION DONE)
                          (MACROEXPAND-1 PLACE ENV)
                        (COND
                           ((AND DONE (NOT (EQ EXPANSION PLACE)))
                            (SETQ PLACE EXPANSION))
                           (T (RETURN (SETF-ERROR (CAR PLACE)
                                                  PLACE)))))))
                (GO LP)))))
```

```
(DEFUN SETF-ERROR (FN &OPTIONAL FORM)
    ;; Common error routine for invalid SETF's.  FN is the thing we tried to find a setf method for, FORM is its parent (not supplied when the form is a
    ;; non-list).
    (ERROR "~S is not a known location specifier for SETF." FN))


(DEFMACRO PSETF (&REST ARGS &ENVIRONMENT ENV)
   "This is to SETF as PSETQ is to SETQ.  Args are alternating place
     expressions and values to go into those places.  All of the subforms and
     values are determined, left to right, and only then are the locations
     updated.  Returns NIL."
   (DO ((A ARGS (CDDR A))
        (LET-LIST NIL)
        (SETF-LIST NIL))
       ((ATOM A)
        `(,'LET ,(REVERSE LET-LIST)
                ,@(REVERSE SETF-LIST)
                NIL))
     (IF (ATOM (CDR A))
         (ERROR "Odd number of args to PSETF."))
     (MULTIPLE-VALUE-BIND (DUMMIES VALS NEWVAL SETTER GETTER)
         (GET-SETF-METHOD (CAR A)
                  ENV)
       (DECLARE (IGNORE GETTER))
       (DO* ((D DUMMIES (CDR D))
             (V VALS (CDR V)))
            ((NULL D))
         (PUSH (LIST (CAR D)
                     (CAR V))
```

```
                LET-LIST))
        (PUSH (LIST (CAR NEWVAL)
                    (CADR A))
              LET-LIST)
        (PUSH SETTER SETF-LIST))))

(DEFMACRO SHIFTF (&REST ARGS &ENVIRONMENT ENV)
   "Assigns to each place the value of the form to its right, returns old value of 1st"
   (COND
      ((OR (NULL ARGS)
           (NULL (CDR ARGS)))
       (ERROR "SHIFTF needs at least two arguments"))
      (T (DO* ((A ARGS (CDR A))
               (LET-LIST NIL)
               (SETF-LIST NIL)
               (RESULT (IL:GENSYM))
               (NEXT-VAR RESULT))
              ((ATOM (CDR A))
               (PUSH (LIST NEXT-VAR (CAR A))
                     LET-LIST)
               `(,'LET* ,(REVERSE LET-LIST)
                        ,@(REVERSE SETF-LIST)
                        ,RESULT))
            (MULTIPLE-VALUE-BIND (DUMMIES VALS NEWVAL SETTER GETTER)
                (GET-SETF-METHOD (CAR A)
                                 ENV)
              (DO ((D DUMMIES (CDR D))
                   (V VALS (CDR V)))
                  ((NULL D))
                (PUSH (LIST (CAR D)
                            (CAR V))
                      LET-LIST))
              (PUSH (LIST NEXT-VAR GETTER)
                    LET-LIST)
              (PUSH SETTER SETF-LIST)
              (SETQ NEXT-VAR (CAR NEWVAL)))))))

(DEFMACRO ROTATEF (&REST ARGS &ENVIRONMENT ENV)
   "Assigns to each place the value of the form to its right; last gets first. Returns NIL."
   ;; forms evaluated in order
   (COND
      ((NULL ARGS)
       NIL)
      ((NULL (CDR ARGS))
       `(PROGN ,(CAR ARGS)
               NIL))
      (T (DO ((A ARGS (CDR A))
              (LET-LIST NIL)
              (SETF-LIST NIL)
              (NEXT-VAR NIL)
              (FIX-ME NIL))
             ((ATOM A)
              (RPLACA FIX-ME NEXT-VAR)
              `(,'LET* ,(REVERSE LET-LIST)
                       ,@(REVERSE SETF-LIST)
                       NIL))
           (MULTIPLE-VALUE-BIND (DUMMIES VALS NEWVAL SETTER GETTER)
               (GET-SETF-METHOD (CAR A)
                                ENV)
             (DO ((D DUMMIES (CDR D))
                  (V VALS (CDR V)))
                 ((NULL D))
               (PUSH (LIST (CAR D)
                           (CAR V))
                     LET-LIST))
             (PUSH (LIST NEXT-VAR GETTER)
                   LET-LIST)
             ;; We don't know the newval variable for the last form yet,so fake it for the first getter and fix it at the end.
             (UNLESS FIX-ME
                 (SETQ FIX-ME (CAR LET-LIST)))
             (PUSH SETTER SETF-LIST)
             (SETQ NEXT-VAR (CAR NEWVAL)))))))

(DEFMACRO POP (PLACE &ENVIRONMENT ENV)
   "Pops one item off the front of PLACE and returns it."
   (IF (SYMBOLP PLACE)
       `(PROG1 (CAR ,PLACE)
            (SETQ ,PLACE (CDR ,PLACE)))
       (MULTIPLE-VALUE-BIND (DUMMIES VALS NEWVAL SETTER GETTER)
           (GET-SETF-METHOD PLACE ENV)
         `(,'LET* (,@(MAPCAR #'LIST DUMMIES VALS)
                   ,(LIST (CAR NEWVAL)
```

```
                        GETTER))
                (PROG1 (CAR ,(CAR NEWVAL))
                    (SETQ ,(CAR NEWVAL)
                          (CDR ,(CAR NEWVAL)))
                    ,SETTER)))))
```

```
(DEFMACRO REMF (PLACE INDICATOR &ENVIRONMENT ENV)
   "Destructively remove INDICATOR from PLACE, returning T if it was present, NIL if not"
   (MULTIPLE-VALUE-BIND (DUMMIES VALS NEWVAL SETTER GETTER)
       (GET-SETF-METHOD PLACE ENV)
     (LET ((IND-TEMP (IL:GENSYM))
           (LOCAL1 (IL:GENSYM))
           (LOCAL2 (IL:GENSYM)))
          `(,'LET* (,@(MAPCAR #'LIST DUMMIES VALS)
                    (,(CAR NEWVAL)
                     ,GETTER)
                    (,IND-TEMP ,INDICATOR))
                   (DO ((,LOCAL1 ,(CAR NEWVAL)
                                 (CDDR ,LOCAL1))
                        (,LOCAL2 NIL ,LOCAL1))
                       ((ATOM ,LOCAL1)
                        NIL)
                     (COND
                        ((ATOM (CDR ,LOCAL1))
                         (ERROR "Odd-length property list in REMF."))
                        ((EQ (CAR ,LOCAL1)
                             ,IND-TEMP)
                         (COND
                            (,LOCAL2 (RPLACD (CDR ,LOCAL2)
                                             (CDDR ,LOCAL1))
                                     (RETURN T))
                            (T (SETQ ,(CAR NEWVAL)
                                     (CDDR ,(CAR NEWVAL)))
                               ,SETTER
                               (RETURN T)))))))))))
```

```
(DEFINE-MODIFY-MACRO INCF (&OPTIONAL (DELTA 1)) +
   "The first argument is some location holding a number.  This number is
     incremented by the second argument, DELTA, which defaults to 1.")
```

```
(DEFINE-MODIFY-MACRO DECF (&OPTIONAL (DELTA 1)) -
   "The first argument is some location holding a number.  This number is
     decremented by the second argument, DELTA, which defaults to 1.")
```

```
(DEFUN MAYBE-MAKE-BINDING-FORM (NEWVAL-FORM DUMMIES VALS NEWVAR SETTER GETTER)
   ;; For use in SETF-like forms to produce their final expression without using the NEWVAR gensym where possible.  DUMMIES thru GETTER are the
   ;; five values returned from the SETF method.  NEWVAL-FORM is an expression to which the (sole) NEWVAR is logically to be bound, written in
   ;; terms of the GETTER form.  If it looks like there are no side-effect problems, we substitute NEWVAL-FORM into SETTER; otherwise we return a
   ;; binding form that returns SETTER correctly.
   (IF (OR DUMMIES (> (COUNT-OCCURRENCES (CAR NEWVAR)
                                         SETTER)
                      1))                                    ; have to do messy binding form
       `(,'LET* (,@(MAPCAR #'LIST DUMMIES VALS)
                 (,(CAR NEWVAR)
                  ,NEWVAL-FORM))
                ,SETTER)                                     ; No temp vars, setter used only once, so nothing can be
                                                             ; side-effected, so store it directly
       (SUBST NEWVAL-FORM (CAR NEWVAR)
              SETTER)))
```

```
(DEFUN COUNT-OCCURRENCES (SYMBOL FORM)
   (COND
      ((CONSP FORM)
       (+ (COUNT-OCCURRENCES SYMBOL (CAR FORM))
          (COUNT-OCCURRENCES SYMBOL (CDR FORM))))
      ((EQ SYMBOL FORM)
       1)
      (T 0)))
```

```
(DEFMACRO PUSH (OBJ PLACE &ENVIRONMENT ENV)
   "Conses OBJ onto PLACE, returning the modified list."
   (IF (SYMBOLP PLACE)
       `(SETQ ,PLACE (CONS ,OBJ ,PLACE))
       (MULTIPLE-VALUE-BIND (DUMMIES VALS NEWVAL SETTER GETTER)
           (GET-SETF-METHOD PLACE ENV)
         (MAYBE-MAKE-BINDING-FORM `(CONS ,OBJ ,GETTER)
                DUMMIES VALS NEWVAL SETTER GETTER))))
```

```
(DEFMACRO PUSHNEW (OBJ PLACE &REST KEYS &ENVIRONMENT ENV)
```

```
   "Conses OBJ onto PLACE unless its already there, using :TEST if necessary"
   (IF (SYMBOLP PLACE)
       `(SETQ ,PLACE (ADJOIN ,OBJ ,PLACE ,@KEYS))
       (MULTIPLE-VALUE-BIND (DUMMIES VALS NEWVAL SETTER GETTER)
           (GET-SETF-METHOD PLACE ENV)
         (MAYBE-MAKE-BINDING-FORM `(ADJOIN ,OBJ ,GETTER ,@KEYS)
                DUMMIES VALS NEWVAL SETTER GETTER))))


(DEFSETF CAR (X) (V)
   `(CAR (RPLACA ,X ,V)))


(DEFSETF CDR (X) (V)
   `(CDR (RPLACD ,X ,V)))


(DEFSETF CAAAAR (X) (V)
   `(CAR (RPLACA (CAAAR ,X)
                 ,V)))


(DEFSETF CAAADR (X) (V)
   `(CAR (RPLACA (CAADR ,X)
                 ,V)))


(DEFSETF CAAAR (X) (V)
   `(CAR (RPLACA (CAAR ,X)
                 ,V)))


(DEFSETF CAADAR (X) (V)
   `(CAR (RPLACA (CADAR ,X)
                 ,V)))


(DEFSETF CAADDR (X) (V)
   `(CAR (RPLACA (CADDR ,X)
                 ,V)))


(DEFSETF CAADR (X) (V)
   `(CAR (RPLACA (CADR ,X)
                 ,V)))


(DEFSETF CAAR (X) (V)
   `(CAR (RPLACA (CAR ,X)
                 ,V)))


(DEFSETF CADAAR (X) (V)
   `(CAR (RPLACA (CDAAR ,X)
                 ,V)))


(DEFSETF CADADR (X) (V)
   `(CAR (RPLACA (CDADR ,X)
                 ,V)))


(DEFSETF CADAR (X) (V)
   `(CAR (RPLACA (CDAR ,X)
                 ,V)))


(DEFSETF CADDAR (X) (V)
   `(CAR (RPLACA (CDDAR ,X)
                 ,V)))


(DEFSETF CADDDR (X) (V)
   `(CAR (RPLACA (CDDDR ,X)
                 ,V)))


(DEFSETF CADDR (X) (V)
   `(CAR (RPLACA (CDDR ,X)
                 ,V)))


(DEFSETF CADR (X) (V)
   `(CAR (RPLACA (CDR ,X)
                 ,V)))
```

```
(DEFSETF CDAAAR (X) (V)
   `(CDR (RPLACD (CAAAR ,X)
                 ,V)))

(DEFSETF CDAADR (X) (V)
   `(CDR (RPLACD (CAADR ,X)
                 ,V)))

(DEFSETF CDAAR (X) (V)
   `(CDR (RPLACD (CAAR ,X)
                 ,V)))

(DEFSETF CDADAR (X) (V)
   `(CDR (RPLACD (CADAR ,X)
                 ,V)))

(DEFSETF CDADDR (X) (V)
   `(CDR (RPLACD (CADDR ,X)
                 ,V)))

(DEFSETF CDADR (X) (V)
   `(CDR (RPLACD (CADR ,X)
                 ,V)))

(DEFSETF CDAR (X) (V)
   `(CDR (RPLACD (CAR ,X)
                 ,V)))

(DEFSETF CDDAAR (X) (V)
   `(CDR (RPLACD (CDAAR ,X)
                 ,V)))

(DEFSETF CDDADR (X) (V)
   `(CDR (RPLACD (CDADR ,X)
                 ,V)))

(DEFSETF CDDAR (X) (V)
   `(CDR (RPLACD (CDAR ,X)
                 ,V)))

(DEFSETF CDDDAR (X) (V)
   `(CDR (RPLACD (CDDAR ,X)
                 ,V)))

(DEFSETF CDDDDR (X) (V)
   `(CDR (RPLACD (CDDDR ,X)
                 ,V)))

(DEFSETF CDDDR (X) (V)
   `(CDR (RPLACD (CDDR ,X)
                 ,V)))

(DEFSETF CDDR (X) (V)
   `(CDR (RPLACD (CDR ,X)
                 ,V)))

(DEFSETF FIRST (X) (V)
   `(CAR (RPLACA ,X ,V)))

(DEFSETF SECOND (X) (V)
   `(CAR (RPLACA (CDR ,X)
                 ,V)))

(DEFSETF THIRD (X) (V)
   `(CAR (RPLACA (CDDR ,X)
                 ,V)))

(DEFSETF FOURTH (X) (V)
   `(CAR (RPLACA (CDDDR ,X)
                 ,V)))
```

```
(DEFSETF FIFTH (X) (V)
   `(CAR (RPLACA (CDDDDR ,X)
                 ,V)))

(DEFSETF SIXTH (X) (V)
   `(CAR (RPLACA (CDR (CDDDDR ,X))
                 ,V)))

(DEFSETF SEVENTH (X) (V)
   `(CAR (RPLACA (CDDR (CDDDDR ,X))
                 ,V)))

(DEFSETF EIGHTH (X) (V)
   `(CAR (RPLACA (CDDDR (CDDDDR ,X))
                 ,V)))

(DEFSETF NINTH (X) (V)
   `(CAR (RPLACA (CDDDDR (CDDDDR ,X))
                 ,V)))

(DEFSETF TENTH (X) (V)
   `(CAR (RPLACA (CDR (CDDDDR (CDDDDR ,X)))
                 ,V)))

(DEFSETF REST (X) (V)
   `(CDR (RPLACD ,X ,V)))

(DEFSETF NTHCDR (N LIST) (NEWVAL)
   `(CDR (RPLACD (NTHCDR (1- ,N)
                         ,LIST)
                 ,NEWVAL)))

(DEFSETF NTH %SET-NTH)

(DEFINE-SETF-METHOD GETF (PLACE PROP &OPTIONAL DEFAULT &ENVIRONMENT ENV)
   (MULTIPLE-VALUE-BIND (TEMPS VALUES STORES SET GET)
       (GET-SETF-METHOD PLACE ENV)
     (LET ((NEWVAL (IL:GENSYM))
           (PTEMP (IL:GENSYM))
           (DEF-TEMP (IL:GENSYM)))
          (VALUES `(,@TEMPS ,(CAR STORES)
                            ,PTEMP
                            ,@(IF DEFAULT
                                  `(,DEF-TEMP)))
                  `(,@VALUES ,GET ,PROP ,@(IF DEFAULT
                                              `(,DEFAULT)))
                  `(,NEWVAL)
                  `(COND
                      ((NULL ,(CAR STORES))
                       (LET* ,(LIST (APPEND STORES `((LIST ,PTEMP ,NEWVAL))))
                             ,SET)
                       ,NEWVAL)
                      (T (IL:LISTPUT ,(CAR STORES)
                                     ,PTEMP
                                     ,NEWVAL)))
                  `(GETF ,(CAR STORES)
                         ,PTEMP
                         ,@(IF DEFAULT
                               `(,DEF-TEMP)))))))

(DEFINE-SETF-METHOD APPLY (FN &REST ARGS &ENVIRONMENT ENV)
   (IF (AND (CONSP FN)
            (EQ (LENGTH FN)
                2)
            (MEMBER (FIRST FN)
                    '(FUNCTION IL:FUNCTION QUOTE)
                    :TEST
                    #'EQ)
            (SYMBOLP (SECOND FN)))
       (SETQ FN (SECOND FN))
       (ERROR "Setf of Apply is only defined for function args of form #'symbol."))
   (MULTIPLE-VALUE-BIND (DUMMIES VALS NEWVAL SETTER GETTER)
       (GET-SETF-METHOD (CONS FN ARGS)
              ENV)
     ;; Make sure the place is one that we can handle.
```

```
     (UNLESS (AND (EQ (CAR (LAST ARGS))
                      (CAR (LAST VALS)))
                  (EQ (CAR (LAST GETTER))
                      (CAR (LAST DUMMIES)))
                  (EQ (CAR (LAST SETTER))
                      (CAR (LAST DUMMIES))))
             (ERROR "Apply of ~S not understood as a location for Setf." FN))
     (VALUES DUMMIES VALS NEWVAL `(APPLY #',(CAR SETTER)
                                         ,@(CDR SETTER))
             `(APPLY #',(CAR GETTER)
                     ,@(CDR GETTER)))))


(DEFINE-SETF-METHOD LDB (BYTESPEC PLACE &ENVIRONMENT ENV)
   "The first argument is a byte specifier.  The second is any place form
     acceptable to SETF.  Replaces the specified byte of the number in this
     place with bits from the low-order end of the new value."
   (MULTIPLE-VALUE-BIND (DUMMIES VALS NEWVAL SETTER GETTER)
       (GET-SETF-METHOD PLACE ENV)
     (LET ((BTEMP (IL:GENSYM))
           (GNUVAL (IL:GENSYM)))
          (VALUES (CONS BTEMP DUMMIES)
                  (CONS BYTESPEC VALS)
                  (LIST GNUVAL)
                  `(LET ((,(CAR NEWVAL)
                          (DPB ,GNUVAL ,BTEMP ,GETTER)))
                        ,SETTER
                        ,GNUVAL)
                  `(LDB ,BTEMP ,GETTER)))))


(DEFINE-SETF-METHOD MASK-FIELD (BYTESPEC PLACE &ENVIRONMENT ENV)
   "The first argument is a byte specifier.  The second is any place form
     acceptable to SETF.  Replaces the specified byte of the number in this place
     with bits from the corresponding position in the new value."
   (MULTIPLE-VALUE-BIND (DUMMIES VALS NEWVAL SETTER GETTER)
       (GET-SETF-METHOD PLACE)
     (LET ((BTEMP (IL:GENSYM))
           (GNUVAL (IL:GENSYM)))
          (VALUES (CONS BTEMP DUMMIES)
                  (CONS BYTESPEC VALS)
                  (LIST GNUVAL)
                  `(LET ((,(CAR NEWVAL)
                          (DEPOSIT-FIELD ,GNUVAL ,BTEMP ,GETTER)))
                        ,SETTER
                        ,GNUVAL)
                  `(MASK-FIELD ,BTEMP ,GETTER)))))


(DEFINE-SETF-METHOD CHAR-BIT (PLACE BIT-NAME &ENVIRONMENT ENV)
   "The first argument is any place form acceptable to SETF.  Replaces the
     specified bit of the character in this place with the new value."
   (MULTIPLE-VALUE-BIND (DUMMIES VALS NEWVAL SETTER GETTER)
       (GET-SETF-METHOD PLACE ENV)
     (LET ((BTEMP (IL:GENSYM))
           (GNUVAL (IL:GENSYM)))
          (VALUES `(,@DUMMIES ,BTEMP)
                  `(,@VALS ,BIT-NAME)
                  (LIST GNUVAL)
                  `(LET ((,(CAR NEWVAL)
                          (SET-CHAR-BIT ,GETTER ,BTEMP ,GNUVAL)))
                        ,SETTER
                        ,GNUVAL)
                  `(CHAR-BIT ,GETTER ,BTEMP)))))


(DEFINE-SETF-METHOD THE (TYPE PLACE &ENVIRONMENT ENV)
   (MULTIPLE-VALUE-BIND (DUMMIES VALS NEWVAL SETTER GETTER)
       (GET-SETF-METHOD PLACE ENV)
     (VALUES DUMMIES VALS NEWVAL (SUBST `(THE ,TYPE ,(CAR NEWVAL))
                                        (CAR NEWVAL)
                                        SETTER)
             `(THE ,TYPE ,GETTER))))

;; Some IL setfs, for no especially good reason


(DEFSETF IL:GETHASH IL:%SET-IL-GETHASH)


(DEFMACRO IL:%SET-IL-GETHASH (KEY HASH-TABLE &OPTIONAL NEWVALUE)
   ;; SETF inverse for IL:GETHASH.  Tricky parts are that args to IL:PUTHASH are in wrong order, and IL:GETHASH might default its second arg
   ;; (yuck, let's flush that), in which case the third arg is absent and the second is the new value.
   (COND
      ((NOT NEWVALUE)                                          ; Defaulted hash table
```

```
       `(IL:PUTHASH ,KEY ,HASH-TABLE))
      ((OR (IL:CONSTANTEXPRESSIONP NEWVALUE)
           (AND (SYMBOLP NEWVALUE)
                (SYMBOLP HASH-TABLE)))                          ; Ok to swap args
       `(IL:PUTHASH ,KEY ,NEWVALUE ,HASH-TABLE))
      (T `(LET (IL:$$GETHASH-TABLE)
               (DECLARE (IL:LOCALVARS IL:$$GETHASH-TABLE))
               (IL:PUTHASH ,KEY (PROGN (IL:SETQ IL:$$GETHASH-TABLE ,HASH-TABLE)
                                       ,NEWVALUE)
                     IL:$$GETHASH-TABLE)))))

(IL:PUTPROPS :SETF-METHOD-EXPANDER IL:PROPTYPE IGNORE)

(IL:PUTPROPS :SETF-INVERSE IL:PROPTYPE IGNORE)

(IL:PUTPROPS :SHARED-SETF-INVERSE IL:PROPTYPE IGNORE)

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

(IL:PUTPROPS IL:CMLSETF IL:MAKEFILE-ENVIRONMENT (:READTABLE "XCL" :PACKAGE "LISP"))

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

(IL:ADDTOVAR IL:NLAMA )

(IL:ADDTOVAR IL:NLAML )

(IL:ADDTOVAR IL:LAMA )
)

(IL:PUTPROPS IL:CMLSETF IL:COPYRIGHT ("Venue & Xerox Corporation" 1986 1987 1988 1990))
```

FUNCTION INDEX

SETF INDEX

MACRO INDEX

DEFINER INDEX

PROPERTY INDEX

DEFINE-TYPE INDEX