File created: 6-Nov-91 16:29:23 **{DSK}<python>RELEASE>loops>2.0>src>LOOPSMETHODS.;3**

changes to: (FUNCTIONS SubclassResponsibility)
(OPTIMIZERS _ SEND)
(FNS Cached-FetchMethodOrHelp)

previous date: 15-Aug-91 12:38:42 {DSK}<python>RELEASE>loops>2.0>src>LOOPSMETHODS.;2

Read Table: INTERLISP

Package: INTERLISP

Format: XCCS

```
;;
```


```
(RPAQQ LOOPSMETHODSCOMS
       [;; WARNING: YOU MUST SET *Compile-Local-Message-Cache* TO NIL BEFORE COMPILING THIS FILE. Failing to do so, you'll get a
        ;; Cached-FetchMethodOrHelp that calls itself recursively forever.
        (DECLARE%: EVAL@COMPILE DONTCOPY (FILES (LOADCOMP)
                                                LOOPSSTRUC))


;;; Translation of METHOD forms to LAMBDA forms

        (COMS ;; Ways to send a message
              (FUNCTIONS _ SEND _! __ _New _IV)
              (MACROS _Proto _Try _Process _Process! SENDSUPER)
              (FNS FetchMethodLocally _Apply)
              (MACROS DOAPPLY* DoMethod FetchMethod FindSelectorIndex GetNthMethod MapSupersForm?)
              (COMS                                                ; Optimizer for _
                    (INITVARS (*Compile-Local-Message-Cache* T))
                    (OPTIMIZERS _ SEND)))
        (FUNCTIONS SubclassResponsibility)
        (FNS AddMethod ApplyMethod ApplyMethodInTtyProcess DefMethObj DefineMethod DeleteMethod DoFringeMethods
             DoMethod FindSuperMethod IVFunction BootInstallMethod FullInstallMethod InstanceNotMethod LoopsHelp
             METH METHOBJ MessageAuthor MethName MoveMethod RenameMethod \ApplyMethod FindLocalMethod
             FindSelectorIndex FetchMethod FetchMethodOrHelp GetCallerClass GetNthMethod GetSuperMethod
             PutMethodNth DCM)
        (P (MOVD? 'BootInstallMethod 'InstallMethod))


;;; Method lookup caching stuff

        (FNS Cached-FetchMethodOrHelp)


;;; Other stuff ???

        (MACROS MapSupersForm MapSupersUnlessBadList NextSuperClass)
        (FNS GetMethodSource CheckMethodChanged CheckMethodForm)
        (INITVARS (LoopsDebugFlg T))
        (P (ADDTOVAR NLAMA METH DoMethod DoFringeMethods))
        (DECLARE%: DONTCOPY (PROP FILETYPE LOOPSMETHODS)
              (PROP MAKEFILE-ENVIRONMENT LOOPSMETHODS))
        (DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS (ADDVARS (NLAMA METH DoMethod
                                                                                    DoFringeMethods)
                                                                                (NLAML METHOBJ)
                                                                                (LAMA LoopsHelp])
```

;; WARNING: YOU MUST SET *Compile-Local-Message-Cache* TO NIL BEFORE COMPILING THIS FILE. Failing to do so, you'll get a
;; Cached-FetchMethodOrHelp that calls itself recursively forever.

```
(DECLARE%: EVAL@COMPILE DONTCOPY

(FILESLOAD (LOADCOMP)
       LOOPSSTRUC)
)


;;; Translation of METHOD forms to LAMBDA forms
;; Ways to send a message


(DEFMACRO _ (self selector &REST args)
   `(_!
     ,self
     ',selector
     ,@args))

(DEFMACRO SEND (self selector &REST args)
```

```
   `(_ ,self ,selector ,@args))


(DEFMACRO _! (self selector &REST args)
   [Once-Only (self)
          `(APPLY* (FetchMethodOrHelp ,self ,selector)
                   ,self
                   ,@args])


(DEFMACRO ___ (self selector &REST args)
   `[_ ,self ,selector ,@(for x in args collect (KWOTE x])


(DEFMACRO _New (class &OPTIONAL (selector NIL selector-supplied-p)
                      &REST args)
   (if selector-supplied-p
       then [LET ((self (GENSYM)))
                 `(LET ((,self (_ ,class New)))
                       (DECLARE (LOCALVARS ,self))
                       (_ ,self ,selector ,@args)
                       ,self]
     else `(_ ,class New)))


(DEFMACRO _IV (self IVName &REST args)
   [Once-Only (self)
          `(APPLY* (IVFunction ,self ',IVName)
                   ,self
                   ,@args])

(DECLARE%: EVAL@COMPILE

(PUTPROPS _Proto MACRO ((obj . args)
                        (_ (_ obj Prototype) . args)))

(PUTPROPS _Try MACRO [(obj action . args)
                      (PROG ((obj%  obj))
                            (RETURN (DOAPPLY* (OR (FetchMethod (Class obj% )
                                                               'action)
                                                  (RETURN 'NotSent))
                                              obj%  . args])

(PUTPROPS _Process MACRO [X (LET [(obj (CAR X))
                                  (selector (CADR X))
                                  (args (CONS 'LIST (CDDR X]
                                 `(ADD.PROCESS (LIST 'ApplyMethod (KWOTE ,obj)
                                                     (KWOTE ',selector)
                                                     (KWOTE ,args))
                                        'NAME
                                        ',selector])

(PUTPROPS _Process! MACRO [X (LET [(obj (CAR X))
                                   (selector (CADR X))
                                   (args (CONS 'LIST (CDDR X]
                                  `(ADD.PROCESS (LIST 'ApplyMethod (KWOTE ,obj)
                                                      (KWOTE ,selector)
                                                      (KWOTE ,args))
                                         'NAME
                                         ',selector])

(PUTPROPS SENDSUPER MACRO ((obj action . args)
                           (_Super
                            obj action . args)))
)

(DEFINEQ

(FetchMethodLocally
  [LAMBDA (classRec selector)                                   ; Edited 16-Mar-88 16:29 by jrb:
    (LET (index)
         (COND
            ((SETQ index (FindSelectorIndex classRec selector))
             (GetNthMethod classRec index])


(_Apply
  [LAMBDA (argList)                                             ; Edited 14-Aug-90 16:53 by jds
                                                                (* Apply the selected method to the already evaluated args in argList.)
    (APPLY [OR (FetchMethod (fetch (OBJECT CLASS) of (CAR argList))
                            (CADR argList))
               (ERROR (CADR argList)
                      (CONCAT "not a selector in " (fetch (OBJECT CLASS) of (CAR argList]
           (CONS (CAR argList)
                 (CDDR argList])
```

```
)

(DECLARE%: EVAL@COMPILE

(PUTPROPS DOAPPLY* MACRO (arg (CONS 'CL:FUNCALL arg)))

(PUTPROPS DoMethod MACRO [(obj action class . args)
                          (LET ((obj%  obj)
                                (class%  class))
                               (if (Class? (OR class%  (fetch (OBJECT CLASS) of obj% )))
                                   then (DOAPPLY* (OR (FetchMethod (OR class%  (fetch (OBJECT CLASS) of obj% ))
                                                                   action)
                                                      (ERROR action "not found for DoMethod"))
                                                  obj%  . args)
                                 else (ERROR (OR class%  (fetch (OBJECT CLASS) of obj% ))
                                             "not a class"])

(PUTPROPS FetchMethod MACRO [OPENLAMBDA (classRec selector)     (* smL "17-Sep-85 17:45")
                                                                (* Returns the function for selector or NIL)
                             (PROG ((pos (LLSH (LOGAND 1023 (LOGXOR (\LOLOC classRec)
                                                                    (\LOLOC selector)))
                                               3))
                                    (class classRec)
                                    meth index supers)
                                   (DECLARE (LOCALVARS . T))
                                   [COND
                                      ((AND (EQ class (\GETBASEPTR *Global-Method-Cache* pos))
                                            (EQ selector (\GETBASEPTR (\ADDBASE *Global-Method-Cache* 2)
                                                                      pos)))
                                       (RETURN (\GETBASEPTR (\ADDBASE *Global-Method-Cache* 4)
                                                            pos]
                                   (SETQ supers (fetch (class supers) of classRec))
                                LP (COND
                                      ((SETQ index (FindSelectorIndex class selector))
                                       (SETQ meth (GetNthMethod class index))
                                       (\PUTBASEPTR *Global-Method-Cache* pos classRec)
                                       (\PUTBASEPTR (\ADDBASE *Global-Method-Cache* 2)
                                              pos selector)
                                       (\PUTBASEPTR (\ADDBASE *Global-Method-Cache* 4)
                                              pos meth)
                                       (RETURN meth))
                                      ((SETQ class (pop supers))
                                       (GO LP))
                                      (T (RETURN NIL])

(PUTPROPS FindSelectorIndex MACRO [OPENLAMBDA (classrec selector)
                                   (PROG NIL

        (* Prog is only so one can bomb out in case of NIL selectors of class)

                                         (RETURN (\FindEntryIndex selector (OR (fetch (class selectors)
                                                                                  of classrec)
                                                                               (RETURN])

(PUTPROPS GetNthMethod MACRO [OPENLAMBDA (classrec n)
                              (LET ((meths (fetch (class methods) of classrec)))
                                   (COND
                                      ((LISTP meths)
                                       (GetNth meths n))
                                      (T (\GetNthEntry meths n])

(PUTPROPS MapSupersForm? MACRO ((mappingForm classRec . progArgs)
                                                                (* dgb%: "12-JAN-82 14:55")

        (* Maps through a class and its supers in order. Returns if form has return statement, or NIL when finished %.
        form can use class as free variable)

                               (PROG (supers (class classRec) . progArgs)
                                     (COND
                                        ((NULL class)
                                         (RETURN NIL)))
                                     (SETQ supers (Supers class))
                                LP
                                mappingForm
                                                                (* this is where the substitution goes)
                                ON  (COND
                                       ((SETQ class (pop supers))
                                                                (* If there is a Super, iterate around the Loop)
                                        (GO LP)))               (* Returns NotSetValue if not found)
                                     (RETURN NotSetValue))))
)

;; Optimizer for _

(RPAQ? *Compile-Local-Message-Cache* T)
```

```
(DEFOPTIMIZER _ (object selector &REST args)
               [if (AND *Compile-Local-Message-Cache* *BYTECOMPILER-IS-EXPANDING*)
                   then `(LET [(/\obj/\ ,object)
                               (*LOOPS-INLINE-METHOD-CACHE* (LOADTIMECONSTANT (\Make-Method-Cache-Entry]
                              (DECLARE (LOCALVARS /\obj/\ *LOOPS-INLINE-METHOD-CACHE*))
                              (LOOPS-FUNCALL (COND
                                                 ((AND (Object? /\obj/\)
                                                       (EQ (\GETBASEPTR *LOOPS-INLINE-METHOD-CACHE* 0)
                                                           (fetch (OBJECT CLASS) of /\obj/\)))
                                                        ; A cache hit
                                                  (\GETBASEPTR *LOOPS-INLINE-METHOD-CACHE* 2))
                                                 (T                ; A cache miss
                                                    (Cached-FetchMethodOrHelp /\obj/\ ',selector
                                                           *LOOPS-INLINE-METHOD-CACHE*)))
                                             /\obj/\
                                             ,@args))
                        ;; '((OPENLAMBDA (/\obj/\ *LOOPS-INLINE-METHOD-CACHE*) (DECLARE (LOCALVARS . T)) (CL:FUNCALL (if
                        ;; (AND (Object? /\obj/\) (EQ (\GETBASEPTR *LOOPS-INLINE-METHOD-CACHE* 0) (fetch (OBJECT CLASS) of
                        ;; /\obj/\))) then ; A cache hit (\GETBASEPTR *LOOPS-INLINE-METHOD-CACHE* 2) else ; A cache miss
                        ;; (Cached-FetchMethodOrHelp /\obj/\ ',selector *LOOPS-INLINE-METHOD-CACHE*)) /\obj/\ ,@args)) ,object
                        ;; (LOADTIMECONSTANT (\Make-Method-Cache-Entry)))

                   elseif *BYTECOMPILER-IS-EXPANDING*
                     then `(LET ((/\obj/\ ,object))
                              (DECLARE (LOCALVARS /\obj/\))
                              (LOOPS-FUNCALL (FetchMethodOrHelp /\obj/\ ',selector)
                                     /\obj/\
                                     ,@args))
                   else
                   (LET* [(obj (if (LITATOM object)
                                   then object
                                 else (GENSYM)))
                          [bindings (if (EQ obj object)
                                        then NIL
                                      else `((,obj ,object]
                          (localVars (for binding in bindings collect (CAR binding]
                         (if *Compile-Local-Message-Cache*
                             then `(LET (,@bindings)
                                        ;; ,@(if localVars then '((DECLARE (LOCALVARS ,@localVars))) else NIL)
                                        (DECLARE (LOCALVARS . T))
                                        (LOOPS-FUNCALL (LET [(*LOOPS-INLINE-METHOD-CACHE* (LOADTIMECONSTANT
                                                                                                  (
                                                                                                 \Make-Method-Cache-Entry
                                                                                                   ]
                                        ;; This bogus SPECVARS stuff is here to prevent the compiler from thinking that the in-line cache is a
                                        ;; quoted constant.  Note that (almost) no user code gets called within this binding, so it is pretty safe.
                                        ;; (The potential exception is when the user has redefined the FetchMethodOrHelp method).
                                                           ;; (DECLARE (SPECVARS *LOOPS-INLINE-METHOD-CACHE*))
                                                           (if [AND (Object? ,obj)
                                                                    (EQ (\GETBASEPTR *LOOPS-INLINE-METHOD-CACHE*
                                                                                0)
                                                                        (fetch (OBJECT CLASS) of ,obj]
                                                               then
                                                                  ; A cache hit
                                                                   (\GETBASEPTR *LOOPS-INLINE-METHOD-CACHE* 2)
                                                             else
                                                                  ; A cache miss
                                                                   (Cached-FetchMethodOrHelp ,obj
                                                                          ',selector *LOOPS-INLINE-METHOD-CACHE*))
                                                           )
                                                  ,obj
                                                  ,@args))
                              elseif bindings
                                then `(LET (,@bindings)
                                           (DECLARE (LOCALVARS ,@localVars))
                                           (LOOPS-FUNCALL (FetchMethodOrHelp ,obj ',selector)
                                                  ,obj
                                                  ,@args))
                              else `(LOOPS-FUNCALL (FetchMethodOrHelp ,obj ',selector)
                                           ,obj
                                           ,@args])


(DEFOPTIMIZER SEND (object selector &REST args)
                  [if (AND *Compile-Local-Message-Cache* *BYTECOMPILER-IS-EXPANDING*)
                      then `(LET [(/\obj/\ ,object)
                                  (*LOOPS-INLINE-METHOD-CACHE* (LOADTIMECONSTANT (\Make-Method-Cache-Entry]
                                 (DECLARE (LOCALVARS /\obj/\ *LOOPS-INLINE-METHOD-CACHE*))
                                 (LOOPS-FUNCALL (COND
                                                    ((AND (Object? /\obj/\)
                                                          (EQ (\GETBASEPTR *LOOPS-INLINE-METHOD-CACHE* 0)
                                                              (fetch (OBJECT CLASS) of /\obj/\)))
                                                           ; A cache hit
                                                     (\GETBASEPTR *LOOPS-INLINE-METHOD-CACHE* 2))
```

```
                                                    (T          ; A cache miss
                                                       (Cached-FetchMethodOrHelp /\obj/\ ',selector
                                                              *LOOPS-INLINE-METHOD-CACHE*)))
                                                /\obj/\
                                                ,@args))
                                    ;; '((OPENLAMBDA (/\obj/\ *LOOPS-INLINE-METHOD-CACHE*) (DECLARE (LOCALVARS . T)) (CL:FUNCALL
                                    ;; (if (AND (Object? /\obj/\) (EQ (\GETBASEPTR *LOOPS-INLINE-METHOD-CACHE* 0) (fetch (OBJECT
                                    ;; CLASS) of /\obj/\))) then ; A cache hit (\GETBASEPTR *LOOPS-INLINE-METHOD-CACHE* 2) else ; A cache
                                    ;; miss (Cached-FetchMethodOrHelp /\obj/\ ',selector *LOOPS-INLINE-METHOD-CACHE*)) /\obj/\ ,@args))
                                    ;; ,object (LOADTIMECONSTANT (\Make-Method-Cache-Entry)))

                                elseif *BYTECOMPILER-IS-EXPANDING*
                                  then `(LET ((/\obj/\ ,object))
                                             (DECLARE (LOCALVARS /\obj/\))
                                             (LOOPS-FUNCALL (FetchMethodOrHelp /\obj/\ ',selector)
                                                    /\obj/\
                                                    ,@args))
                                else
                                (LET*
                                 [(obj (if (LITATOM object)
                                           then object
                                         else (GENSYM)))
                                  [bindings (if (EQ obj object)
                                                then NIL
                                              else `((,obj ,object]
                                  (localVars (for binding in bindings collect (CAR binding]
                                 (if *Compile-Local-Message-Cache*
                                     then `(LET (,@bindings)
                                                ;; ,@(if localVars then '((DECLARE (LOCALVARS ,@localVars))) else NIL)
                                                (DECLARE (LOCALVARS . T))
                                                (LOOPS-FUNCALL (LET [(*LOOPS-INLINE-METHOD-CACHE* (LOADTIMECONSTANT (
                                                                                                  \Make-Method-Cache-Entry
                                                                                                                    ]
                                                ;; This bogus SPECVARS stuff is here to prevent the compiler from thinking that the in-line cache is a
                                                ;; quoted constant.  Note that (almost) no user code gets called within this binding, so it is pretty safe.
                                                ;; (The potential exception is when the user has redefined the FetchMethodOrHelp method).
                                                               ;; (DECLARE (SPECVARS *LOOPS-INLINE-METHOD-CACHE*))
                                                               (if [AND (Object? ,obj)
                                                                        (EQ (\GETBASEPTR *LOOPS-INLINE-METHOD-CACHE* 0
                                                                                    )
                                                                            (fetch (OBJECT CLASS) of ,obj]
                                                                  then
                                                                     ; A cache hit
                                                                       (\GETBASEPTR *LOOPS-INLINE-METHOD-CACHE* 2)
                                                                 else
                                                                     ; A cache miss
                                                                      (Cached-FetchMethodOrHelp ,obj
                                                                             ',selector *LOOPS-INLINE-METHOD-CACHE*))
                                                               )
                                                         ,obj
                                                         ,@args))
                                   elseif bindings
                                     then `(LET (,@bindings)
                                                (DECLARE (LOCALVARS ,@localVars))
                                                (LOOPS-FUNCALL (FetchMethodOrHelp ,obj ',selector)
                                                         ,obj
                                                         ,@args))
                                   else `(LOOPS-FUNCALL (FetchMethodOrHelp ,obj ',selector)
                                                ,obj
                                                ,@args])
```

```
(DEFMACRO SubclassResponsibility ()
   (DECLARE (CL:SPECIAL *ArgsOfMethodBeingCompiled* *ClassNameOfMethodOwner* *SelectorOfMethodBeingCompiled*
                    *SelfOfMethodBeingCompiled*))
   `(HELPCHECK (CONCAT "Method " ,*SelectorOfMethodBeingCompiled* " of class " ,*ClassNameOfMethodOwner* " needs
                     to be defined in class ")
           (_ ,*SelfOfMethodBeingCompiled* ClassName)))

(DEFINEQ

(AddMethod
  [LAMBDA (class selector method)                                    ; Edited 14-Aug-90 16:53 by jds

          (* * Adds a method to a class, or replaces the function named if selecor is already local to class)

    (LET ((index (FindSelectorIndex class selector)))
         (COND
            (index                                                   (* already in class)
                   (PutMethodNth class index method))
            (T

          (* This is an efficiency hack. If we flush the cache on new method read in from a file, we can waste a great deal of time.
          This counts on the filepackage def for methods dumping out an explicit call to FlushMethodCache at the end of a block of
```

methods.)

```
(if (NOT (\BatchMethodDefs?))
    then (FlushMethodCache))
(UNINTERRUPTABLY
    (LET* ((sels (fetch (class selectors) of class))
           (freePos (\FreeEntryIndex sels)))
          (replace (class selectors) of class with (\AddBlockEntry sels selector freePos))
          (replace (class methods) of class with (\AddBlockEntry (fetch (class methods) of class)
                                                                 method freePos)))))])
```

(ApplyMethod

```
[LAMBDA (object selector argList class)                    ; Edited 14-Aug-90 16:53 by jds
                                                           (* Apply the selected method to the already evaluated args in
                                                           argList.)
  (if (OR (type? instance object)
          (type? class object))
      then (APPLY [OR (FetchMethod (OR class (fetch (OBJECT CLASS) of object))
                                   selector)
                      (ERROR selector (CONCAT "not a selector in " (OR class (fetch (OBJECT CLASS) of object]
                  (CONS object argList))
    else (HELPCHECK object " not an instance or class"])
```

(ApplyMethodInTtyProcess

```
[LAMBDA (object selector argList class)                    ; Edited 27-May-87 17:06 by smL
```

(* Apply the selected method to the already evaluated args in argList as a tty process.)

```
(EVAL.IN.TTY.CONTEXT `(ApplyMethod ,object ',selector ',argList ,class)
       selector])
```

(DefMethObj

```
[LAMBDA (cName sel fn arg dcm methodProps otherIVs)        ; Edited 17-Jun-87 16:09 by smL
```

;; Creates the method object and fills in its IVs If the UID of the object is there, then use it else create a new object

```
(LET ((uid (LISTGET methodProps 'UID))
      (methName (AND cName (MethName cName sel)))
      (methClass (OR ($! (LISTGET methodProps 'methodClass))
                     ($ Method)))
      oldSelf self)
     (SETQ oldSelf ($! methName))                          ; If there is already an instance with the same name and class,
                                                           ; use it
     [SETQ self (COND
                   ((AND oldSelf (OR (NULL uid)
                                     (NOT (HasUID? oldSelf))
                                     (UIDEqual uid (UID oldSelf)))
                         (EQ methClass (Class oldSelf)))   ; already have the method instance, so re-use it
                    oldSelf)
                   (T (NewObject methClass (COND
                                             ((NULL uid)
                                              (Make-UID))
                                             (T uid]
     [COND
        (methName (NameObject self (CONS methName]
     (PutValueOnly self 'className cName)
     (PutValueOnly self 'selector sel)
     (PutValueOnly self 'args arg)
     (PutValueOnly self 'doc dcm)
     (PutValueOnly self 'method fn)
     (for p on methodProps do (PutValueOnly self 'method (CADR p)
                                            (CAR p))
        by (CDDR p))
```

;; method is filled by fn. all the other IVs which the method can have are in oth, and will be filled into the instance

```
     (COND
        (otherIVs (FillInst otherIVs self)))
     (InstallMethod self)
     self])
```

(DefineMethod

```
[LAMBDA (class selector args expr file methodType)         ; Edited 23-Nov-87 16:58 by Bane
```

;;; Define a new method (or replace an old one). If expr is NIL then args should be a list of arguments, and expr should be the function definition. File is ;;; the place where this method should be stored. methodType can be a method defining macro or NIL.

```
  (if (NOT (LITATOM selector))
      then (ERROR selector "is not a LIATOM, so cannot be a selector"))
  (if (NOT (LITATOM file))
      then (ERROR file "is not a LITATOM, so can't be a fileName"))
  (if (AND args (LITATOM args))
      then                                                 ; Naming a function to be used as a method - this is no longer
                                                           ; allowed
          (ERROR "Can't explicitly name the method function"))
```

```
    (if (NOT (Class? class))
        then (ERROR class "is not a Class object"))
    (LET* [(className (GoodClassName class))
           (methName (MethName className selector))
           (doc (if [AND (LISTP expr)
                         (LISTP (CDR expr))
                         (OR (AND (LISTP (CAR expr))
                                  (EQ COMMENTFLG (CAAR expr)))
                             (STRINGP (CAR expr]
                    then (pop expr)
                  else (CONCAT "Method documentation"]
```

;; Save the method on a file

```
          (COND
             ([OR file (AND (NULL (WHEREIS methName 'METHODS))
                            (SETQ file (CAR (WHEREIS className 'CLASSES]
              (ADDTOFILE methName 'METHODS file)))
```

;; Build the method

```
          (EVAL (PACK-METHOD-BODY className selector (CONS 'self args)
                      NIL
                      (COND
                         [(NULL expr)                              ; No Method Body Given
                          (COPY `((SubclassResponsibility]
                         ([AND (LISTP expr)
                               (NOT (LISTP (CAR expr]              ; This is a single expression, not an implicit PROGN
                          (LIST expr))
                         (T expr))
                      doc NIL methodType))
```

;; Edit it if no body was provided

```
          (COND
             ((NOT (OR args expr))
              (_ class EditMethod selector)))
```

;; Return the name

```
          methName])
```

(DeleteMethod

```
  [LAMBDA (class selector prop)                                    ; Edited 14-Aug-90 16:53 by jds
```

;; If prop is NIL or T this means delete the method from the class. Otherwise delete the method property. If prop is T then also delete the function definition

```
    (PROG (methObj (methName (MethName class selector))
                index pl fn freePos sel file)
      TRYAGAIN
          (SETQ class (GetClassRec class))
          (SETQ index (FindSelectorIndex class selector))
          (COND
             ((NULL class)
              (SETQ class (HELPCHECK class " is not a known class. Type
                                  RETURN 'className
                                  to try again"))
              (GO TRYAGAIN))
             ((NULL index)
              (SETQ selector (HELPCHECK class " does not contain the selector " selector "Type
                                     RETURN 'selectorName
                                     to try again"))
              (GO TRYAGAIN)))
          (SETQ fn (GetMethod class selector))
          (COND
             (prop (APPLY* (FUNCTION UNBREAK)
                           fn)))
          [COND
             ((EQ prop T)                                          ; T is special Flag for deleteing the function definition too
              (SETQ prop NIL)
              (CLEARW PROMPTWINDOW)
              (printout PROMPTWINDOW (CHARACTER 7)
                     "Deleting function definition for " fn T)
              (\PUTD fn))
             [prop                                                 ; This deletes a real property of a method
                   (SETQ methObj (GetMethodObj class selector))
                   (RETURN (COND
                              ((FMEMB prop (GetClassValue methObj 'ivProperties))
                               (PutValueOnly methObj prop NotSetValue))
                              (T (DeleteIV methObj 'method prop]
             ((SETQ file (WHEREIS (SETQ fn (GetMethod class selector))
                                  'METHODS))                       ; Remember to save fn
              (ADDTOFILE fn 'FNS (CAR file]
```

;; \DeleteNthEntry requires knowing the freePos. Must compute it from selectors because it checks for occurrence of NIL in block to mark end

```
          (MARKASCHANGED methName 'METHODS 'DELETED)
          (FlushMethodCache)
          (AND ($! methName)
               (_ ($! methName)
                  Destroy))
```

```
          (UNINTERRUPTABLY
              [SETQ freePos (\FreeEntryIndex (SETQ sel (fetch (class selectors) of class]
              (\DeleteNthEntry sel index freePos)
              (\DeleteNthEntry (fetch (class methods) of class)
                     index freePos))])
```

(DoFringeMethods

```
  [NLAMBDA |obj selector ..args|                                    ; Edited 14-Aug-90 16:53 by jds
```

(* * This calls all the methods of an object in the immediate supers of the object.
selector is evaluated.)

```
    (PROG [selector object objClass fn (argList (MAPCAR |obj selector ..args| (FUNCTION EVAL]
          (DECLARE (LOCALVARS . T))
          (SETQ object (CAR argList))
          (SETQ selector (CADR argList))
          (SETQ argList (CONS object (CDDR argList)))
          (SETQ objClass (Class object))
          (COND
             ((NULL objClass)
              (ERROR object "has no class"))
             ((SETQ fn (FetchMethodLocally objClass selector))
              (APPLY fn argList))
             (T (for cls in (fetch (class localSupers) of objClass) do (COND
                                                                          ((SETQ fn (FetchMethod cls selector))
                                                                           (APPLY fn argList])
```

(DoMethod

```
  [NLAMBDA |obj selector class ..args|                              (* smL "29-May-86 17:55")
                                                                    (* Function for macro so that args are known)
    (DECLARE (LOCALVARS . T)
           (SPECVARS classForMethod))
    (LET (classForMethod allArgs oBj)
         (SETQ allArgs (MAPCAR |obj selector class ..args| (FUNCTION EVAL)))
         (SETQ oBj (pop allArgs))
         (if (type? class (OR (CADR allArgs)
                              (Class oBj)))
             then (APPLY [OR (FetchMethod (OR (CADR allArgs)
                                              (Class oBj))
                                    (CAR allArgs))
                             (ERROR (CAR allArgs)
                                    (CONCAT "not a selector for " (OR (CADR allArgs)
                                                                      (Class oBj]
                         (CONS oBj (CDDR allArgs)))
          else (ERROR (OR (CADR allArgs)
                          (Class oBj))
                      "not a class"])
```

(FindSuperMethod

```
  [LAMBDA (object selector classOfSendingMethod noError?)           (* smL " 5-Jun-86 14:29")
                                                                    (* Searches for an selector up the supers chain.
                                                                    If none found, calls LISP HELP)
    (OR [for class in [LET ((class (Class object)))
                           (COND
                              ((EQ classOfSendingMethod class)
                               (Supers class))
                              (T (CDR (FMEMB classOfSendingMethod (Supers class]
          bind index do (COND
                           ((SETQ index (FindSelectorIndex class selector))
                                                                    (* There is a response in this class)
                            (RETURN (GetNthMethod class index]
        noError?
        (_ object SuperMethodNotFound selector classOfSendingMethod])
```

(IVFunction

```
  [LAMBDA (obj ivName)                                              (* edited%: " 3-Apr-86 17:36")
    (LET ((fnName (GetValue obj ivName)))
         (COND
            ((DEFINEDP fnName)
             fnName)
            (T (LoopsHelp "No iv function" obj ivName fnName])
```

(BootInstallMethod

```
  [LAMBDA (self)                                                    (* edited%: "21-Nov-85 14:18")
```

(* Used in kernel system to add methods. Replaced by FullInstallMethod after LOOPSKERNEL is loaded by LOADLOOPS)

```
    (AddMethod (GetObjectRec (GetIVHere self 'className))
           (GetIVHere self 'selector)
           (GetIVHere self 'method])
```

(FullInstallMethod

```
[LAMBDA (self)                                          (* dgb%: " 1-NOV-83 08:02")
                                                        (* Used after kernel is installed. Calls a method to install a
                                                        method)
   (_ self OldInstance])
```

(InstanceNotMethod

```
[LAMBDA (name)                                          (* edited%: " 6-Feb-85 17:44")
                                                        (* test if this instance is not a method)
   (LET ((inst ($! name)))
        (COND
           ([AND (type? instance inst)
                 (NOT (_ inst InstOf! 'Method]
            (GetInstanceSource name])
```

(LoopsHelp

```
[LAMBDA msgs                                            (* smL "25-Apr-86 15:36")

          (* * The standard way of generating an error in Loops. -
          If LoopsDebugFlg is set, go into a continuable error, otherwise generate an un-continuable error.)

   (LET [(msg (APPLY (FUNCTION CONCAT)
                     (CDR (for i from 1 to msgs join (LIST " " (ARG msgs i]
        (if LoopsDebugFlg
            then (HELP "LoopsHelp:" msg)
          else (ERROR "LoopsHelp:" msg])
```

(METH

```
[NLAMBDA methDescr                                      (* smL "19-Aug-86 11:43")

          (* * Put out by the class method. Contains in order the (className selector methName
          (if different from className.selector) args doc . other-properties))

   (LET (self cName sel fnName methName args doc methodProps (descr methDescr))
        (SETQ cName (pop descr))
        (SETQ sel (pop descr))
        (SETQ methName (MethName cName sel))
        (COND
           ((AND (SETQ fnName (CAR descr))
                 (LITATOM fnName))                      (* Method name which is not identical to methName)
            (SETQ descr (CDR descr)))
           (T (SETQ fnName methName)))
        (SETQ args (pop descr))
        (SETQ doc (pop descr))
        [COND
           ((EQ 'method (CAAR descr))                   (* There are methodProps or there is a funny function name)
            (SETQ methodProps (pop descr))
            (SETQ fnName (CADR methodProps))
            (SETQ methodProps (CDDR methodProps]
        (DefMethObj cName sel fnName args doc methodProps descr])
```

(METHOBJ

```
[NLAMBDA (methodInfo doc otherIVs methodType uid)       (* kmk%: "22-Feb-85 17:16")

          (* * Input form is -- (METHOBJ2 ((selector argListSpec)%. argList) doc otherIVs methodType uid))

   (DefMethObj (CADR (CAR methodInfo))
          (CAAR methodInfo)
          (MethName (CADR (CAR methodInfo))
                 (CAAR methodInfo))
          (CDR (CDR methodInfo))
          doc
          (LIST 'methodClass methodType 'UID uid)
          otherIVs])
```

(MessageAuthor

```
[LAMBDA NIL                                             (* smL "17-Mar-85 18:53")
   (LET [(currentMessage (STKSCAN 'self]
        (AND currentMessage (EVALV 'self (STKNTH 1 currentMessage])
```

(MethName

```
[LAMBDA (classOrName selector)                          (* dgb%: " 5-Apr-84 08:14")
                                                        (* Make name of form className.selector)
   (PACK* (COND
             ((type? class classOrName)
              (ClassName classOrName))
             (T classOrName))
          "." selector])
```

(MoveMethod

```
[LAMBDA (oldClassName newClassName selector newSelector files) ; Edited 16-Mar-88 14:50 by jrb:

;;; Move a method from oldClassName to newClassName, renaming function if appropriate

    (SETQ oldClassName (GoodClassName oldClassName NIL T))
    (OR newClassName (SETQ newClassName oldClassName))
    (SETQ newClassName (GoodClassName newClassName NIL T))
    (OR newSelector (SETQ newSelector selector))
    (PROG (oldDef newLocalFn delFnFlg (oldClass (GetClassRec oldClassName))
                (newClass (GetClassRec newClassName))
                (localFn (FindLocalMethod (GetClassRec oldClassName)
                                selector)))
     ;; Punt now for null moves
          (if (AND (EQ oldClass newClass)
                   (EQ selector newSelector))
              then (RETURN NIL))
          (COND
             ((NULL localFn)
              (printout T selector " not found in " oldClassName)
              (RETURN NIL))
             [(STRPOS oldClassName localFn)
              (OR (SETQ oldDef (GETDEF localFn 'METHOD-FNS))
                  (ERROR "No defintion found for " localFn))     ; Remember to delete fn def Dont use DELDEF since it bitches.
              (SETQ delFnFlg T)                                  ; Define the method
              (SETQ newLocalFn (EVAL (CL:MULTIPLE-VALUE-BIND (cname sel args decls formsd doc quals method-type)
                                         (PARSE-METHOD-BODY oldDef)
                                       (PACK-METHOD-BODY newClassName newSelector args decls formsd doc quals
                                            method-type))]
             (T (AddMethod newClass newSelector localFn)))
          (for prop in (DREMOVE 'RuleSet (_ oldClass ListAttribute 'Method selector))
             do (PutMethodOnly newClass newSelector (GetMethodOnly oldClass selector prop)
                       prop))
          (DeleteMethod oldClass selector delFnFlg)
          (RETURN (OR newLocalFn localFn])
```

(RenameMethod

```
[LAMBDA (classOrName oldSelector newSelector)                    ; Edited 17-Nov-87 16:18 by jrb:

;;; Rename selector in class, and rename method also if it is composite.  If oldClassName is given, then class has been renamed, and not selector
;;; changed.

    (PROG (className class newLocalFn oldDef oldMethName newMethName file)
          (COND
             ((NULL classOrName)
              (printout T "NIL is not a class" T)
              (RETURN))
             ((OR (NULL oldSelector)
                  (NULL newSelector))
              (printout T "NIL is not a valid selector" T)
              (RETURN)))
          [COND
             ((type? class classOrName)
              (SETQ class classOrName)
              (SETQ className (ClassName class)))
             (T (SETQ className (GoodClassName classOrName NIL T))
                (SETQ class (GetClassRec className]
          (SETQ oldMethName (FindLocalMethod class oldSelector))
          (COND
             ((NULL oldMethName)
              (printout T oldSelector " not found in " className T)
              (RETURN)))
          (SETQ oldDef (GETDEF oldMethName 'METHOD-FNS))
          (COND
             ((NULL oldDef)
              (ERROR oldMethName " defn cannot be found for RenameMethod")))
          [SETQ file (CAR (WHEREIS oldMethName 'METHODS]
          (_ (GetMethodObj class oldSelector T)
             ChangeName oldMethName (MethName className newSelector)
             newSelector)
          (DeleteMethod class oldSelector T)
          [SETQ newMethName (EVAL (CL:MULTIPLE-VALUE-BIND (oldClassName oldSelector arg-list decls forms doc
                                                                  qualifiers method-type)
                                      (PARSE-METHOD-BODY oldDef)
                                    (PACK-METHOD-BODY className newSelector arg-list decls forms doc qualifiers
                                          method-type))]
          (COND
             (file (ADDTOFILE newMethName 'METHODS file)))
          (RETURN newMethName])
```

(\ApplyMethod

```
[LAMBDA (selector argList class)                                 (* dgb%: "16-Nov-84 16:36")

          (* * Apply the selected method to the already evaluated args in argList.
          argList includes the object as first item)
```

```
    (APPLY [OR (FetchMethod (OR class (Class (CAR argList)))
                       selector)
               (ERROR selector (CONCAT "not a selector in " (OR class (Class (CAR argList]
           argList])
```

(FindLocalMethod

```
  [LAMBDA (class selector)                                          (* smL "16-Dec-85 16:25")
                                                                    (* Return function handling method in this class, or NIL if there is
    none)
    (LET ((index (FindSelectorIndex class selector)))
         (AND index (GetNthMethod class index])
```

(FindSelectorIndex

```
  [LAMBDA (class selector)                                          ; Edited 14-Aug-90 16:53 by jds
    (PROG NIL

          (* Prog is only so one can bomb out in case of NIL selectors of class)

          (RETURN (\FindEntryIndex selector (OR (fetch (class selectors) of class)
                                                (RETURN])
```

(FetchMethod

```
  [LAMBDA (classRec selector)                                       ; Edited 14-Aug-90 16:53 by jds
                                                                    (* Returns the function for selector or NIL)
    (PROG ((pos (LLSH (LOGAND 1023 (LOGXOR (\LOLOC classRec)
                                           (\LOLOC selector)))
                      3))
           (class classRec)
           meth index supers)
          (DECLARE (LOCALVARS . T))
          [COND
             ((AND (EQ class (\GETBASEPTR *Global-Method-Cache* pos))
                   (EQ selector (\GETBASEPTR (\ADDBASE *Global-Method-Cache* 2)
                                             pos)))
              (RETURN (\GETBASEPTR (\ADDBASE *Global-Method-Cache* 4)
                                   pos]
          (SETQ supers (fetch (class supers) of classRec))
      LP  (COND
             ((SETQ index (FindSelectorIndex class selector))
              (SETQ meth (GetNthMethod class index))
              (\PUTBASEPTR *Global-Method-Cache* pos classRec)
              (\PUTBASEPTR (\ADDBASE *Global-Method-Cache* 2)
                     pos selector)
              (\PUTBASEPTR (\ADDBASE *Global-Method-Cache* 4)
                     pos meth)
              (RETURN meth))
             ((SETQ class (pop supers))
              (GO LP))
             (T (RETURN NIL])
```

(FetchMethodOrHelp

```
  [LAMBDA (self selector)                                           (* smL "17-Sep-85 17:51")
    ;; Searches for method corresponding to selector up the supers chain.  If successful returns the name of the lisp function.
    (OR (FetchMethod (Class self)
                selector)
        (_ self MethodNotFound selector])
```

(GetCallerClass

```
  [LAMBDA (object selector fromCaller)                              (* smL "10-Oct-85 15:24")
                                                                    (* Get the class of the caller for use by _Super)
    (PROG (class fn index supersList stkPos (callerName fromCaller))
      SETCALLER
          (OR (SETQ stkPos (REALSTKNTH -1 (OR stkPos callerName)
                                  NIL stkPos))
              (LoopsHelp selector "No caller found in " (OR fromCaller "_Super")))
          (SETQ callerName (STKNAME stkPos))
      CALLERSET
          (SETQ class (Class object))
          (SETQ supersList (Supers class))
      LP  [COND
             ((SETQ index (FindSelectorIndex class selector))       (* There is a method in this class)
              (COND
                 ((EQ callerName (SETQ fn (GetNthMethod class index)))

          (* Fn here is the same one I am in. Return class as callerName of DoSuperMethods)

                  (RELSTK stkPos)
                  (RETURN class]
          (COND
             ((SETQ class (pop supersList))                         (* Try next superClass)
```

```
             (GO LP))
            (T                                                    (* Never found containing method --
                                                                  Move Back one callerName)
               (GO SETCALLER])
```

(GetNthMethod

```
  [LAMBDA (class n)                                               ; Edited 14-Aug-90 16:53 by jds
    (LET ((meths (fetch (class methods) of class)))
         (COND
            ((LISTP meths)
             (GetNth meths n))
            (T (\GetNthEntry meths n])
```

(GetSuperMethod

```
  [LAMBDA (object selector callerName noError?)                   (* smL " 5-Jun-86 14:30")

          (* * THIS IS OBSOLETE AND IS RETAINED BECAUSE COMPILED CODE CALLS IT.
          IT HAS BEEN SUPERSEEDED BY FindSuperMethod)
                                                                  (* Searches for an selector up the supers chain.
                                                                  If none found, calls LISP HELP.)
    (PROG (index fn flg supersList class stkPos)
          [COND
             (callerName                                          (* first time could be set by interpreted _Super)
                    (GO CALLERSET))
             (T                                                   (* Initialization for SETCALLER)
                (SETQ callerName 'GetSuperMethod]
      SETCALLER
          (OR (SETQ stkPos (REALSTKNTH -1 (OR stkPos callerName)
                                       NIL stkPos))
              (PROGN (RELSTK stkPos)
                     (LoopsHelp "No caller found in _Super for:
                            " selector)))
          (SETQ callerName (STKNAME stkPos))
      CALLERSET
          (SETQ class (Class object))
          (SETQ supersList (Supers class))
      LP  [COND
             ((SETQ index (FindSelectorIndex class selector))     (* There is a response in this class)
              (COND
                 ((EQ callerName (SETQ fn (GetNthMethod class index)))

          (* Fn here is the same one I am in. Mark caller as found by seting flg)

                  (SETQ flg T))
                 (flg                                             (* found a response and have previously seen caller of _Super)
                      (RELSTK stkPos)
                      (RETURN fn]
          (COND
             ((SETQ class (pop supersList))                       (* Try next superClass)
              (GO LP))
             (flg                                                 (* No super found, though method was found)
                  (if noError?
                      then (RETURN noError?)
                    else (LoopsHelp selector "not understood in _Super")))
             (T                                                   (* Never found containing method --
                                                                  Move Back one caller)
                (GO SETCALLER])
```

(PutMethodNth

```
  [LAMBDA (class n fn)                                            ; Edited 14-Aug-90 16:53 by jds
    (PROG ((meths (fetch (class methods) of class)))
          (RETURN (COND
                     ((NULL meths)
                      (ERROR class "no methods in class"))
                     ((LISTP meths)
                      (PutNth meths n fn))
                     (T (\PutNthEntry meths n fn])
```

(DCM

```
  [LAMBDA (className selector)                                    ; Edited 27-May-87 17:06 by smL
                                                                  (* Combine methods from supers with _SuperFringe)
    (PROG [(argList (ARGLIST (GetMethod (OR (GetClassRec className)
                                            (ERROR className "not a defined class"))
                                        selector]
          (RETURN (_ (GetClassRec className)
                     DefMethod selector argList (SUBST (CDR argList)
                                                       'argL
                                                       `(        (* Combined method for (\, selector))
                                                         (_SuperFringe
                                                          self
                                                          ,selector . argL])
)
```

```
(MOVD? 'BootInstallMethod 'InstallMethod)
```

;;; Method lookup caching stuff

```
(DEFINEQ

(Cached-FetchMethodOrHelp
  [LAMBDA (self selector classCache)                    (* smL "20-Jun-86 13:13")

          (* * Get the method for this instance and selector, and update the cache.)

    (DECLARE (LOCALVARS class method))
    (LET* ((class (Class self))
           (method (FetchMethod class selector)))
          (COND
             (method (\PUTBASEPTR classCache 0 class)
                     (\PUTBASEPTR classCache 2 method))
             (T (_ self MethodNotFound selector])

)

;;; Other stuff ???

(DECLARE%: EVAL@COMPILE

(PUTPROPS MapSupersForm MACRO ((mappingForm classRec . progArgs)
                                                        (* dgb%: "12-JAN-82 14:55")

          (* Maps through a class and its supers in order. Returns if form has return statement, or NIL when finished %.
          form can use class as free variable)

                                (PROG (supers (class classRec) . progArgs)
                                      (COND
                                         ((NULL class)
                                          (RETURN NIL)))
                                      (SETQ supers (Supers class))
                                  LP
                                  mappingForm
                                                        (* this is where the substitution goes)
                                  ON  (COND
                                         ((SETQ class (pop supers))
                                                        (* If there is a Super, iterate around the Loop)
                                          (GO LP)))     (* Returns NIL if not found)
                                      (RETURN NIL))))

(PUTPROPS MapSupersUnlessBadList MACRO ((badList mappingForm classRec . progArgs)
                                                        (* dgb%: "12-JAN-82 14:55")

          (* Maps through a class and its supers in order. Returns if form has return statement, or NIL when finished %.
          form can use class as free variable)

                                 (PROG (supers (class classRec) . progArgs)
                                       (COND
                                          ((NULL class)
                                           (RETURN NIL)))
                                       (SETQ supers (Supers class))
                                   LP                   (* Skip if super is on badList.)
                                       (OR (FMEMB (ClassName class)
                                                  badList)
                                           mappingForm) (* this is where the substitution goes)
                                   ON  (COND
                                          ((SETQ class (pop supers))
                                           (GO LP)))    (* Returns NIL if not found)
                                       (RETURN NIL))))

(PUTPROPS NextSuperClass MACRO [NIL (COND
                                       ((SETQ class (pop supers))

          (* * This code assumes that LP is a defined PROG label and supers and class are bound)
                                                        (* If there is a Super, iterate around the Loop)
                                        (GO LP])
)

(DEFINEQ

(GetMethodSource
  [LAMBDA (obj)                                         (* edited%: "20-Dec-84 08:18")
    (LET [(m (COND
                ((type? instance obj)
                 obj)
                ((LITATOM obj)
                 ($! obj]
          (COND
             ((AND m (_ m InstOf! 'Method))
```

```
            (GetInstanceSource m])
```

(CheckMethodChanged

```
  [LAMBDA NIL                                                    (* smL "21-Dec-84 12:49")

          (* * clean up the mark as changed info for methods)

                                                                 (* if a method instance is changed, so is the method)
    [for fn in (INFILECOMS? NIL 'INSTANCES (FILEPKGCHANGES)) do (if (AND ($! fn)
                                                                        (_ ($! fn)
                                                                           InstOf!
                                                                           'Method))
                                                                   then (UNMARKASCHANGED fn 'INSTANCES)
                                                                        (MARKASCHANGED fn 'METHODS]
                                                                 (* if a method fn is changed, so is the method)
    [for fn in (INFILECOMS? NIL 'FNS (FILEPKGCHANGES)) do (if (AND ($! fn)
                                                               (_ ($! fn)
                                                                  InstOf!
                                                                  'Method))
                                                          then (MARKASCHANGED fn 'METHODS]

          (* if a method is marked, the instance shouldn't be and the fn (probably) should be)

    (for fn in (INFILECOMS? NIL 'METHODS (FILEPKGCHANGES)) do (if (WHEREIS fn 'METHODS)
                                                                  then (MARKASCHANGED fn 'FNS)
                                                               else (UNMARKASCHANGED fn 'FNS])
```

(CheckMethodForm

```
  [LAMBDA (class selector method)                                ; Edited 17-Jun-87 16:28 by smL

;;; Check if method is in current Method form, and convert if necessary

    ;; This used to be (LET* ((methName (OR method (MethName class selector))) (methodDef (GETDEF methName 'FNS NIL '(NOCOPY
    ;; NOERROR NODWIM)))) (COND ((EQ (CAR methodDef) 'LAMBDA) (RPLACA methodDef 'Method) (RPLACA (CDR methodDef) (CONS (LIST
    ;; (ClassName class) selector) (CADR methodDef))) (COND ((AND methodDef (CCODEP methName)) ; This caches the definition in core so that it
    ;; won't go out to file to edit (PUTPROP methName 'EXPR methodDef) (UNSAVEDEF methName 'EXPR))) (MARKASCHANGED methName
    ;; 'FNS))) methodDef), but we don't need all that any more, because all methods are assumed to be in the new format

    (GETDEF (OR method (MethName class selector))
           'METHOD-FNS NIL '(NOCOPY NOERROR NODWIM])

)

(RPAQ? LoopsDebugFlg T)

(ADDTOVAR NLAMA METH DoMethod DoFringeMethods)

(DECLARE%: DONTCOPY

(PUTPROPS LOOPSMETHODS FILETYPE :COMPILE-FILE)

(PUTPROPS LOOPSMETHODS MAKEFILE-ENVIRONMENT (:PACKAGE "IL" :READTABLE "INTERLISP" :BASE 10))
)

(DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS

(ADDTOVAR NLAMA METH DoMethod DoFringeMethods)

(ADDTOVAR NLAML METHOBJ)

(ADDTOVAR LAMA LoopsHelp)
)

(PUTPROPS LOOPSMETHODS COPYRIGHT ("Venue & Xerox Corporation" 1984 1985 1986 1987 1988 1990 1991))
```

FUNCTION INDEX

MACRO INDEX

VARIABLE INDEX

OPTIMIZER INDEX

PROPERTY INDEX