File created: 6-Nov-91 14:51:00 **{DSK}<python>RELEASE>loops>2.0>src>LOOPS-FILEPKG.;3**

changes to: (FUNCTIONS _Super _Super? _SuperFringe Method)
(VARS LOOPS-FILEPKGCOMS)
(FNS INSTALL-METHOD-FN)

previous date: 6-Nov-91 14:34:36 {DSK}<python>RELEASE>loops>2.0>src>LOOPS-FILEPKG.;2

Read Table: INTERLISP

Package: INTERLISP

Format: XCCS

;;


```
(RPAQQ LOOPS-FILEPKGCOMS
       (
```

;;; To speed up loading files, we want to avoid doing a repeated work {like clearing caches or recomputing browsers}. We can find out if we are in a load
;;; because of the following advise.

```
        (ADVISE LOAD)
        (INITVARS (\BatchMethodDefs NIL)
               (\Loading-File NIL))
        (SPECVARS \BatchMethodDefs \Loading-File)
        (FUNCTIONS \Loading-File?)
        (FNS METHODSDelDef \BatchMethodDefs \BatchMethodDefs? \UnbatchMethodDefs)
```

;;; Set up the various Loops FILEPKGTYPES --- The order that things new filepkgtypes are declared is important: those items that should appear first in
;;; a new file should be defined last

;;; FilePackage type for instances

```
        (FNS ContainsOnlyInstances DEFINST DEFINSTANCES INSTANCESEditDef PrttyThese-Instances)
        (FILEPKGCOMS INSTANCES THESE-INSTANCES)
        (ADDVARS (FILEPKGTYPES (INSTANCE . INSTANCES)))
```

;;; FilePackage type for methods

```
        (DEFINE-TYPES METHOD-FNS)
        (FNS PARSE-METHOD-BODY PACK-METHOD-BODY INSTALL-METHOD-FN Method-Fns->Methods)
        [COMS (FUNCTIONS Method)
              (PROP ARGNAMES Method)
```

;; Macros formerly defined by MACROLET inside Method (above). Now done as separate macros, for performance reasons. -- JDS

```
              (FUNCTIONS _Super _Super? _SuperFringe)
```

;; Used in lots of new macros...

```
              (FUNCTIONS LOOPS-FUNCALL)
              (OPTIMIZERS LOOPS-FUNCALL)
```

;; JRB - Method-Fns->Methods must be at the end of MARKASCHANGEDFNS, because it aborts the marking of METHOD-FNS at
;; the last possible instant. Yecch....

```
              (P (SETQ MARKASCHANGEDFNS (NCONC (REMOVE 'Method-Fns->Methods MARKASCHANGEDFNS)
                                               (LIST 'Method-Fns->Methods]
        (FNS FnOrMethodChanged METHODSEditDef NoticeMethodChanged)
        (FILEPKGCOMS METHODS)
```

;;; FilePackage type for classes

```
        (FNS ContainsOnlyClasses CLASSESEditDef DEFCLASS DEFCLASSES UNDO-DEFCLASS UNDO-DEFCLASSES)
        (FILEPKGCOMS CLASSES)
        (ADDVARS (FILEPKGTYPES (CLASS . CLASSES)))
```

;; ObjectAlways PPFlag is bound to NIL in the advice to MAKEFILE so that recursion won't take place there unless the user explicitly sets the
;; value

```
        (ADVISE MAKEFILE)
        (DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY (FILES (LOADCOMP)
                                                                LOOPSDATATYPES LOOPSSTRUC))
        (DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS (ADDVARS (NLAMA DEFCLASSES DEFCLASS
                                                                                DEFINSTANCES DEFINST)
                                                                               (NLAML)
                                                                               (LAMA UNDO-DEFCLASSES)))
        (DECLARE%: DONTCOPY (PROP MAKEFILE-ENVIRONMENT LOOPS-FILEPKG)
               (PROP FILETYPE LOOPS-FILEPKG))))
```

;;; To speed up loading files, we want to avoid doing a repeated work {like clearing caches or recomputing browsers}. We can find out if we are in a load
;;; because of the following advise.

```
[XCL:REINSTALL-ADVICE 'LOAD :AROUND '((:LAST (PROG1 (LET ((\Loading-File T)
                                                          (\BatchMethodDefs NIL))
                                                     (DECLARE (SPECVARS \Loading-File))
                                                     *)
                                                   (UpdateClassBrowsers]

(READVISE LOAD)

(RPAQ? \BatchMethodDefs NIL)

(RPAQ? \Loading-File NIL)

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(SPECVARS \BatchMethodDefs \Loading-File)
)


(DEFMACRO \Loading-File? ()

;;; Are we loading a file

    '\Loading-File)

(DEFINEQ

(METHODSDelDef
  [LAMBDA (name type)                                          ; Edited 24-Nov-87 11:17 by Bane
    (LET* [(dotloc (STRPOS "." name))
           (classname (SUBATOM name 1 (SUB1 dotloc)))
           (methodname (SUBATOM name (ADD1 dotloc]
          (_ ($! classname)
             Delete
             'Method methodname T])


(\BatchMethodDefs
  [LAMBDA NIL                                                  ; Edited 27-May-87 17:43 by smL

          (* * Inform the system that method definitions should be batched.
          This is an efficiency hack. See the fn \UnbatchMethodDefs)

    (DECLARE (SPECVARS \BatchMethodDefs))
    (push \BatchMethodDefs 'T])


(\BatchMethodDefs?
  [LAMBDA NIL
    (DECLARE (SPECVARS \BatchMethodDefs))                      ; Edited  2-Jun-87 14:58 by smL
    \BatchMethodDefs])


(\UnbatchMethodDefs
  [LAMBDA NIL                                                  ; Edited 27-May-87 17:44 by smL

          (* * This unbatches any method defs produces by \BatchMethodDefs)

    (DECLARE (SPECVARS \BatchMethodDefs))
    (FlushMethodCache)
    (pop \BatchMethodDefs])

)


;;; Set up the various Loops FILEPKGTYPES --- The order that things new filepkgtypes are declared is important: those items that should appear first in
;;; a new file should be defined last

;;; FilePackage type for instances

(DEFINEQ

(ContainsOnlyInstances
  [LAMBDA (com name type)                                      (* smL "19-May-86 17:52")

          (* * This function is part of the implementation of INSTANCES as a file package type.)

    (SELECTQ type
        (INSTANCES [LET [(instances (if (EQ '* (CADR com))
                                        then (EVAL (CADDR com))
                                      else (CDR com]
                          (SELECTQ name
                              ((NIL T)
                                  instances)
                              (COND
                                 ((LITATOM name)
                                  (FMEMB name instances))
```

```
                                ((UIDP name)
                                 (for i in instances when (UIDP i) thereis (UIDEqual name i)))
                                (T (for n in name when (for i in instances when (UIDP i)
                                                          thereis (UIDEqual name i))
                                      collect n])
        NIL])
```

(DEFINST

```
  [NLAMBDA DEFINST% FORM                                          (* dgb%: " 4-OCT-83 10:07")
                                                                  (* Defining form for instances on a file)
    (_ [OR (GetClassRec (CAR DEFINST% FORM))
           (PROGN (printout T (CAR DEFINST% FORM)
                        " has no class defined for it" T "Defining one now:" T)
                  (_ ($ Class)
                     New
                     (CAR DEFINST% FORM]
       FileIn
       (CDR DEFINST% FORM])
```

(DEFINSTANCES

```
  [NLAMBDA Instances                                              (* dgb%: " 8-Oct-84 16:59")

          (* Read In the list of named instances, creating a new object with the named UID.
          Make sure it has other names as given in the list. This insures that other later references point to this same instance.)

    (MAPC Instances (FUNCTION (LAMBDA (classNameInstNames)
                                (PROG (instNames className UID obj)
                                      [COND
                                         [(NLISTP (SETQ className (CDR classNameInstNames)))
                                                                  (* Old format input. UID passed as NIL to NewObject)
                                          (SETQ instNames (LIST (CAR classNameInstNames]
                                         (T (SETQ className (CAR classNameInstNames))
                                            (SETQ instNames (REVERSE (CDR classNameInstNames)))
                                            (SETQ UID (pop instNames]
                                      (COND
                                         ((NULL className)
                                          (RETURN)))
                                      (NameObject (NewObject (GetObjectRec className)
                                                             UID)
                                            instNames])
```

(INSTANCESEditDef

```
  [LAMBDA (name type source editComs)                             (* smL "24-Apr-86 18:19")

          (* * Edit the named instance)

    (SELECTQ type
         (INSTANCES (LET ((obj ($! name)))
                         (if obj
                             then (_ obj Edit editComs)
                                  name
                           else NIL)))
         NIL])
```

(PrttyThese-Instances

```
  [LAMBDA (instances)                                             (* smL "15-Aug-86 10:24")

          (* * Dump out these instances on PRTTYFILE. Used by the FILEPKGCOM THESE-INSTANCES.
          Unlike the FILEPKGCOM INSTANCES, this does not include any instances pointed to by these instances.)

    (LET ((OutInstances))
         (for objName in instances bind obj do (SETQ obj (if (Object? objName)
                                                             then objName
                                                           else ($! objName)))
                                               (if obj
                                                   then (PrettyPrintInstance obj (OUTPUT))
                                                        (TERPRI)
                                                        (TERPRI)
                                                 else (HELPCHECK objName " not the name of an instance!" "Type OK
                                                                 to proceed."])

)

[PUTDEF 'INSTANCES 'FILEPKGCOMS '((COM MACRO [INSTANCES (E (PrttyInstances 'INSTANCES]
                                       CONTENTS ContainsOnlyInstances)
                                 (TYPE DESCRIPTION "instances" GETDEF InstanceNotMethod WHENCHANGED (
                                                                                                     FnOrMethodChanged
                                                                                                     )
                                       EDITDEF INSTANCESEditDef]

[PUTDEF 'THESE-INSTANCES 'FILEPKGCOMS '((COM MACRO [INSTANCES (E (PrttyThese-Instances 'INSTANCES]
                                       CONTENTS ContainsOnlyInstances]
```

```
(ADDTOVAR FILEPKGTYPES (INSTANCE . INSTANCES))

;;; FilePackage type for methods

(DEF-DEFINE-TYPE METHOD-FNS "Method functions")

(DEFINEQ

(PARSE-METHOD-BODY
  [LAMBDA (method-body)                                    ; Edited  1-Oct-87 11:19 by smL
    ;; Parse the method def'n body, which should look like (Method {prop value}* ((class-name selector) ...args) ...forms), returning 6 values:  the class
    ;; name, the selector, the arg list, a list of all declarations, the list of forms (less the declarations and doc string), the doc string, a p-list of
    ;; qualifier-value pairs, and the method type (i.e. the CAR of the def'n).
    (LET (method-type arg-list class-name selector doc forms declarations qualifier-plist)
         ;; Get the method type
         (SETQ method-type (pop method-body))
         ;; Collect the qualifier p-list
         [while (AND (NOT (NULL (CAR method-body)))
                     (LITATOM (CAR method-body)))
           do (SETQ qualifier-plist `(,.qualifier-plist ,(pop method-body)
                                                        ,(pop method-body]
         ;; Get the arg-list, class-name, and selector
         (SETQ arg-list (pop method-body))
         (SETQ class-name (CAAR arg-list))
         (SETQ selector (CADAR arg-list))
         (pop arg-list)
         (CL:CHECK-TYPE class-name CL:SYMBOL)
         (CL:CHECK-TYPE selector CL:SYMBOL)
         (CL:CHECK-TYPE arg-list LIST)
         ;; Now process the forms, collecting declarations and a possible doc-string
         (SETQ forms method-body)
         (CL:LOOP (if (NULL (CDR forms))
                      then                                 ; Last form in the list, so can't be a declaration or the comment
                          (RETURN)
                    elseif (LITATOM (CAR forms))
                      then                                 ; Well, it doesn't lead off with a comment
                          (RETURN)
                    elseif (STRINGP (CAR forms))
                      then                                 ; Could be a documentation string
                          (if (NULL doc)
                              then                         ; The doc string
                                   (SETQ doc (pop forms))
                            else                           ; Already have a doc string, so this must be a part of the body
                                 (RETURN))
                    elseif [OR (EDITDATE? (CAR forms))
                               (AND (EQ (CAR (CAR forms))
                                        COMMENTFLG)
                                    (MEMB (CADR (CAR forms))
                                          '(; ;; ;;;))
                                    (STRINGP (CADDR (CAR forms)))
                                    (STRPOS " " (CADDR (CAR forms)))
                                    (IDATE (SUBSTRING (CADDR (CAR forms))
                                                      (STRPOS " " (CADDR (CAR forms]
                      then                                 ; Skip over the edit date
                          (pop forms)
                    elseif (OR (EQ (CAR (CAR forms))
                                   'DECLARE%:)
                               (EQ (CAR (CAR forms))
                                   'DECLARE))
                      then                                 ; A declaration
                          [SETQ declarations `(,.declarations ,(pop forms]
                    elseif (EQ (CAR (CAR forms))
                               COMMENTFLG)
                      then                                 ; A leading comment
                          [LET ((comment (pop forms)))
                               (if doc
                                   then                    ; Already got a doc-string, so ignore this comment
                                        NIL
                                 elseif (AND (MEMB (CADR comment)
                                                   '(; ;; ;;;))
                                             (STRINGP (CADDR comment)))
                                   then                    ; A new-style comment
                                        (SETQ doc (CADDR comment))
                                 elseif (EQ (CADR comment)
                                            COMMENTFLG)
                                   then                    ; An old-style comment like (* * ...)
                                        (SETQ doc (SUBSTRING (CL:FORMAT NIL "~{ ~A~}" (CDDR comment))
                                                             2))
                                 else                      ; An old-style comment like (* ...)
                                      (SETQ doc (SUBSTRING (CL:FORMAT NIL "~{ ~A~}" (CDR comment))
```

```
                                                                    2]
                    else                                    ; A real form, so end the search for more declarations
                         (RETURN)))
          (CL:VALUES class-name selector arg-list declarations forms doc qualifier-plist method-type])
```

(PACK-METHOD-BODY

```
  [LAMBDA (class-name selector arg-list declarations forms doc qualifiers method-type)
                                                                    ; Edited  1-Oct-87 09:06 by smL
    `(,(OR method-type 'Method)
      ,@qualifiers
      ((,class-name ,selector)
       ,@arg-list)
      ,@(if doc
            then (LIST doc)
          else NIL)
      ,@declarations
      ,@forms])
```

(INSTALL-METHOD-FN

```
  [LAMBDA (class-name selector fn-name args doc)            ; Edited 21-Jun-88 17:22 by TAL
    (LET* ((class ($! class-name))
           (methObj (GetMethodObj class selector T))
           whereisClasses)
          (AddMethod class selector fn-name)
          (change (@ methObj method)
                  fn-name)
          (change (@ methObj args)
                  args)
          (change (@ methObj doc)
                  doc)
          (COND
             ((NOT (\Loading-File?))
              (SETQ LASTWORD fn-name)
              ;; TAL - Removed (COND ((AND (NULL (WHEREIS fn-name 'METHODS)) (SETQ whereisClasses (WHEREIS class-name
              ;; 'CLASSES))) (ADDTOFILE fn-name 'METHODS (CAR whereisClasses))))

              ))
          ;; JRB - Removed (MARKASCHANGED fn-name 'METHODS 'DEFINED); I think in the world of METHOD-FNS this marking is not needed
          ;; (this DOES worry me, though).
          (UNMARKASCHANGED fn-name 'FNS)
          (UNMARKASCHANGED fn-name 'INSTANCES)
          fn-name])
```

(Method-Fns->Methods

```
  [LAMBDA (name type reason)                                ; Edited  5-Apr-88 10:18 by bane
    ;; JRB - Punt now from marking a METHOD-FNS as changed.  We should have done everything else significant by now since this function is at the
    ;; end of MARKASCHANGEDFNS (at least it had better be!).
    (if (EQ type 'METHOD-FNS)
        then (MARKASCHANGED name 'METHODS reason)
             (RETFROM 'MARKASCHANGED])

)

(DEFDEFINER (Method [:NAME (CL:LAMBDA (method-body)
                                  (CL:MULTIPLE-VALUE-BIND (class-name selector)
                                      (PARSE-METHOD-BODY method-body)
                                    (MethName class-name selector))])
   METHOD-FNS (&WHOLE method-body)
  (CL:MULTIPLE-VALUE-BIND (class-name selector args declarations forms doc qualifiers method-type)
      (PARSE-METHOD-BODY method-body)
    (CL:ASSERT (Class? ($! class-name)))
    [LET (function-name function-type body)
         ;; Compute the name of the function
         (SETQ function-name (MethName class-name selector))
         ;; Compute the type of the function
         (SETQ function-type (OR (LISTGET qualifiers :FUNCTION-TYPE)
                                 :IL))
         (CL:CHECK-TYPE function-type (CL:MEMBER :CL :IL))
         ;; Get the body of the function, with the top level comments removed
         [SETQ body `(CL:COMPILER-LET [(*ArgsOfMethodBeingCompiled* ',args)
                                       (*ClassNameOfMethodOwner* ',class-name)
                                       (*SelectorOfMethodBeingCompiled* ',selector)
                                       (*SelfOfMethodBeingCompiled* ',(CAR args]
                             ,.forms]
         ;; Build the function definition form
         `(PROGN ,(CL:ECASE function-type
                     (:CL `(CL:DEFUN ,function-name ,args
                              ,@declarations ,body))
```

```
                        (:IL `(DEFINEQ [,function-name (LAMBDA ,args
                                                               ,@declarations
                                                               ,body])))
                    (INSTALL-METHOD-FN ',class-name ',selector ',function-name ',(CDR args)
                           ',doc)
                    ',function-name]))

(PUTPROPS Method ARGNAMES (#\( #\( CLASS-NAME SELECTOR #\) SELF #\{ VAR #\} #\) #\{ FORM #\}))

;; Macros formerly defined by MACROLET inside Method (above).  Now done as separate macros, for performance reasons. -- JDS


(DEFMACRO _Super (&REST Send-Super-Args)
   (DECLARE (CL:SPECIAL *ArgsOfMethodBeingCompiled* *ClassNameOfMethodOwner* *SelectorOfMethodBeingCompiled*
                    *SelfOfMethodBeingCompiled*))
   [COND
      [(NULL Send-Super-Args)                                          ; Args default to args of the method
       `(LOOPS-FUNCALL [FindSuperMethod ,*SelfOfMethodBeingCompiled* ',*SelectorOfMethodBeingCompiled*
                               (LOADTIMECONSTANT (OldClass ,*ClassNameOfMethodOwner*]
              ,.*ArgsOfMethodBeingCompiled*]
      ((NEQ *SelectorOfMethodBeingCompiled* (CADR Send-Super-Args))
                                                                       ; Selectors must match
       (ERROR "Selector to _Super does not match method selector" (CADR Send-Super-Args)))
      ((NEQ *SelfOfMethodBeingCompiled* (CAR Send-Super-Args))   ; Self must match
       (ERROR "Can't _Super to other than first arg of method" (CDR Send-Super-Args)))
      (T                                                               ; Args differ
         (APPEND `(LOOPS-FUNCALL [FindSuperMethod ,*SelfOfMethodBeingCompiled* '
                                                                                ,
                                                                                *SelectorOfMethodBeingCompiled*
                                       (LOADTIMECONSTANT (OldClass ,*ClassNameOfMethodOwner*]
                         ,*SelfOfMethodBeingCompiled*)
                 (CDDR Send-Super-Args]))


(DEFMACRO _Super? (&REST Send-Super-Args)
   (DECLARE (CL:SPECIAL *ArgsOfMethodBeingCompiled* *ClassNameOfMethodOwner* *SelectorOfMethodBeingCompiled*
                    *SelfOfMethodBeingCompiled*))
   [COND
      [(NULL Send-Super-Args)                                          ; Args default to args of the method
       `(LOOPS-FUNCALL (FindSuperMethod ,*SelfOfMethodBeingCompiled* ',*SelectorOfMethodBeingCompiled*
                               (LOADTIMECONSTANT (OldClass ,*ClassNameOfMethodOwner*))
                               (FUNCTION NILL))
              ,.*ArgsOfMethodBeingCompiled*]
      ((NEQ *SelectorOfMethodBeingCompiled* (CADR Send-Super-Args))
                                                                       ; Selectors must match
       (ERROR "Selector to _Super does not match method selector" (CADR Send-Super-Args)))
      ((NEQ *SelfOfMethodBeingCompiled* (CAR Send-Super-Args))   ; Self must match
       (ERROR "Can't _Super? to other than first arg of method" (CDR Send-Super-Args)))
      (T                                                               ; Args differ
         (APPEND `(LOOPS-FUNCALL (FindSuperMethod ,*SelfOfMethodBeingCompiled* '
                                                                                ,
                                                                                *SelectorOfMethodBeingCompiled*
                                       (LOADTIMECONSTANT (OldClass ,*ClassNameOfMethodOwner*))
                                       (FUNCTION NILL))
                         ,*SelfOfMethodBeingCompiled*)
                 (CDDR Send-Super-Args]))


(DEFMACRO _SuperFringe (&REST Send-Super-Args)
   (DECLARE (CL:SPECIAL *ArgsOfMethodBeingCompiled* *ClassNameOfMethodOwner* *SelectorOfMethodBeingCompiled*
                    *SelfOfMethodBeingCompiled*))
   [COND
      [(NULL Send-Super-Args)                                          ; Args default to args of the method
       `(for cls in [fetch localSupers of (LOADTIMECONSTANT (OldClass ,*ClassNameOfMethodOwner*]
           do (LOOPS-FUNCALL (OR (FetchMethod cls ',*SelectorOfMethodBeingCompiled*)
                                 (FUNCTION NILL))
                    ,.*ArgsOfMethodBeingCompiled*]
      ((NEQ *SelectorOfMethodBeingCompiled* (CADR Send-Super-Args))
                                                                       ; Selectors must match
       (ERROR "Selector to _Super does not match method selector" (CADR Send-Super-Args)))
      ((NEQ *SelfOfMethodBeingCompiled* (CAR Send-Super-Args))   ; Self must match
       (ERROR "Can't _SuperFringe to other than first arg of method" (CDR Send-Super-Args)))
      (T `(for cls in [fetch localSupers of (LOADTIMECONSTANT (OldClass ,*ClassNameOfMethodOwner*]
             bind (argList _ (MAPCAR '(,(CAR Send-Super-Args)
                                       ,.(CDDR Send-Super-Args))
                                     (FUNCTION EVAL)))
             do (APPLY (OR (FetchMethod cls ',*SelectorOfMethodBeingCompiled*)
                           (FUNCTION NILL))
                       argList])

;; Used in lots of new macros...

(DEFMACRO LOOPS-FUNCALL (FN &REST ARGS)
   ;; The optimizer for this doesn't make sure FN is evaluated first, but that's ok for loops and save a GENSYM and binding
```

```
   `(APPLY* ,FN ,@ARGS))

(DEFOPTIMIZER LOOPS-FUNCALL (FN &REST ARGS)
                                    ;; This version doesn't make sure FN is evaluated first, but that's ok for loops and save a GENSYM and
                                    ;; binding
                                    `((OPCODES APPLYFN)
                                      ,@ARGS
                                      ,(LENGTH ARGS)
                                      ,FN))

;; JRB - Method-Fns->Methods must be at the end of MARKASCHANGEDFNS, because it aborts the marking of METHOD-FNS at the last possible
;; instant.  Yecch....

[SETQ MARKASCHANGEDFNS (NCONC (REMOVE 'Method-Fns->Methods MARKASCHANGEDFNS)
                              (LIST 'Method-Fns->Methods]

(DEFINEQ

(FnOrMethodChanged
  [LAMBDA (name type reason)                                    (* smL "21-Aug-86 10:10")

          (* * Called when a FN or INSTANCE is marked as changed -
          see the WHENCHANGED field of a FILEPKGTYPEp 11.2 of the IRM)

          (* Marking a method fn or method instance as changed should really mark the method as changed)

    (LET ((obj ($! name)))
         (if (AND obj (_ obj InstOf! 'Method))
             then                                               (* mark the method as changed, too)
                  (MARKASCHANGED name 'METHODS reason)

          (* if the method is not on any file, then don't bother to mark the fn/instance as changed.
          This avoids having FILES? ask about the FN and the MEHTOD both)

                  (if [OR (\Loading-File?)
                          (NOT (WHEREIS name 'METHODS]
                      then (RETFROM 'MARKASCHANGED])

(METHODSEditDef
  [LAMBDA (name type source editComs)                           (* smL "24-Apr-86 18:19")

          (* * Edit the named method)

    (SELECTQ type
         (METHODS (LET ((methodObject ($! name)))
                       (if methodObject
                           then (_ methodObject Edit editComs)
                                name
                         else NIL)))
        NIL])

(NoticeMethodChanged
  [LAMBDA (name type reason)                                    ; Edited 13-Jul-88 09:14 by jrb:

;;; The method has changed.

    [LET ((methObj ($! name)))                                  ; Patch up the comment in the method object
         (COND
            ((AND methObj (NOT (\Loading-File?))
                  [NOT (MEMB reason '(LOAD DELETED]
                  (@ methObj className)
                  ($! (@ methObj className))
                  (@ methObj selector))
             (_ ($! (@ methObj className))
                CommentMethods
                (@ methObj selector)
                T]                                              ; Tell MasterScope that the function has changed.
    ;; JRB - No longer necessary for Lyric - Masterscope analyzes the METHOD-FNS directly. (MSMARKCHANGED name 'FNS reason)

    ])

)

[PUTDEF 'METHODS 'FILEPKGCOMS '((COM MACRO (X (COMS * (METHCOM . X)))
                                      CONTENTS TypeInMethods)
                                 (TYPE DESCRIPTION "methods" GETDEF GetMethodSource DELDEF METHODSDelDef
                                       WHENCHANGED (NoticeMethodChanged)
                                       EDITDEF METHODSEditDef]


;;; FilePackage type for classes
```

```
(DEFINEQ
```

(ContainsOnlyClasses

```
  [LAMBDA (com name type)                                          (* smL "19-May-86 17:53")
```

(* * This function is part of the implementation of CLASSES as a file package type.)

```
    (LET [(classList (COND
                        ((EQ (CADR com)
                             '*)
                         (EVAL (CADDR com)))
                        (T (CDR com]
         (SELECTQ type
             (CLASSES [SELECTQ name
                          ((NIL T)
                               classList)
                          (COND
                             ((LITATOM name)
                              (FMEMB name classList))
                             (T (for n in name when (FMEMB n classList) collect n])
             NIL])
```

(CLASSESEditDef

```
  [LAMBDA (name type source editComs)                              (* smL "24-Apr-86 18:19")
```

(* * Edit the named class)

```
    (SELECTQ type
         (CLASSES (LET ((class ($! name)))
                       (if class
                           then (_ class Edit editComs)
                                name
                         else NIL)))
         NIL])
```

(DEFCLASS

```
  [NLAMBDA FORM                                                    (* edited%: "20-Jun-86 11:16")
```

(* * Used by file package to define a class. DEFCLASS is CAR of defining form)

```
    (PROG ((className (CAR FORM))
           (source (CDR FORM))
           undoForm)
          (COND
             ((OR (NULL source)
                  (NULL className))                                (* Ignore empty class definitions)
              (RETURN NIL))
             ((NOT (LITATOM className))
              (HELPCHECK className " cannot be a class name.
                     Type OK to ignore.")
              (RETURN NIL)))
          [COND
             ((NULL (GetClassRec className))
              (NewClass className)
              (SETQ undoForm NIL))
             ((NULL (fetch metaClass of (GetClassRec className)))  (* No real class yet, so the undo form in NIL)
              (SETQ undoForm NIL))
             (T                                                    (* There is already a real class, so the undo form should
                                                                      reconstruct it)
                (SETQ undoForm (_ (GetClassRec className)
                                  MakeFileSource]
          (COND
             [(CheckClassSource source className)                  (* Dont't install the class if there are errors.)
              (COND
                 ((STKPOS 'EditClassSource)                        (* Bounce back to editor)
                  (ERROR className " not defined  -- bad form " T))
                 (T (PROMPTPRINT className "not installed because of error in source"]
             (T (InstallClassSource className source)
                (UNDOSAVE (LIST 'UNDO-DEFCLASS (CONS 'DEFCLASS FORM)
                                undoForm])
```

(DEFCLASSES

```
  [NLAMBDA CLASSES                                                 (* smL "18-Jun-86 17:25")
```

(* * Used by the file package. When a form (DEFCLASSES c1 c2 --) is read in, class records for c1, c2, -- are created. This allows the real class definitions to be read in in any order.)

```
    (UNDOSAVE (LIST 'UNDO-DEFCLASSES (for className in CLASSES when (NULL (GetClassRec className))
                                        collect (NewEntity (create class
                                                                   className _ className)
                                                           className)
                                      className])
```

```
(UNDO-DEFCLASS
  [LAMBDA (form restorePreviousClass)                    (* smL "18-Jun-86 17:15")

          (* * Undo a (DEFCLASS . form)%. The restorePreviousClass is a form that will restore the class that existed prior to the
          origical call to DEFCLASS)

    (if restorePreviousClass
        then

          (* There was a previous class, so all we need to do is re-install it by eval'ing the form restorePreviousClass, which should be
          a DEFCLASS form, which will end up making this call UNDOable)

             (EVAL restorePreviousClass)
      else

          (* There was no previous class, so we should just destroy this one.
          Of course, we also need to save enough info to UNDO this call)

          (UNDOSAVE form)
          (_ ($! (CADR form))
             Destroy])

(UNDO-DEFCLASSES
  [LAMBDA classes                                        (* smL "18-Jun-86 17:24")

          (* * The classes were created by DEFCLASSES, and should now be destroyed)

    (for className in classes do (_ ($! className)
                                    Destroy])

)

[PUTDEF 'CLASSES 'FILEPKGCOMS '((COM MACRO [CLASSLIST (P (DEFCLASSES . CLASSLIST))
                                                (E (MAPC 'CLASSLIST (FUNCTION PrettyPrintClass]
                                    CONTENTS ContainsOnlyClasses)
                                (TYPE DESCRIPTION "class definitions" GETDEF GetClassSource DELDEF
                                     RemoveClassDef EDITDEF CLASSESEditDef]

(ADDTOVAR FILEPKGTYPES (CLASS . CLASSES))

;; ObjectAlways PPFlag is bound to NIL in the advice to MAKEFILE so that recursion won't take place there unless the user explicitly sets the value

[XCL:REINSTALL-ADVICE 'MAKEFILE :AROUND '((:LAST (LET ((ObjectAlwaysPPFlag NIL))
                                                    *]

(READVISE MAKEFILE)

(DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY

(FILESLOAD (LOADCOMP)
       LOOPSDATATYPES LOOPSSTRUC)
)

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

(ADDTOVAR NLAMA DEFCLASSES DEFCLASS DEFINSTANCES DEFINST)

(ADDTOVAR NLAML )

(ADDTOVAR LAMA UNDO-DEFCLASSES)
)

(DECLARE%: DONTCOPY

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

(PUTPROPS LOOPS-FILEPKG FILETYPE :COMPILE-FILE)
)

(PUTPROPS LOOPS-FILEPKG COPYRIGHT ("Venue & Xerox Corporation" 1986 1987 1988 1990 1991))
```

FUNCTION INDEX

MACRO INDEX

VARIABLE INDEX

PROPERTY INDEX

ADVICE INDEX

DEFINE-TYPE INDEX

OPTIMIZER INDEX

DEFINER INDEX