File created: 15-Jun-88 18:46:10 **{POGO:AISNORTH:XEROX}<LOOPSCORE>LYRIC>USERS>LOOPSBACKWARDS.;11**

changes to: (CLASSES PathBrowser)
(METHODS PathBrowser.Recompute PathBrowser.LeftShiftSelect PathBrowser.GetSubs PathBrowser.Clear
PathBrowser.AddToPath PathBrowser.AddAndBox LatticeBrowser.ListObjects)
(ADVICE $AV $& DEFINST)
(VARS LOOPSBACKWARDSCOMS)
(FNS GetSuperClassValue NewMethodFormat)
(MACROS ObjRealValue)

previous date: 15-Jun-88 17:47:34 {POGO:AISNORTH:XEROX}<LOOPSCORE>LYRIC>USERS>LOOPSBACKWARDS.;10

Read Table: INTERLISP

Package: INTERLISP

Format: XCCS

;;


```
(RPAQQ LOOPSBACKWARDSCOMS
       ((FILES (FROM VALUEOF LOOPSLIBRARYDIRECTORY)
               LOOPSVCOPY)
        (DECLARE%: DONTCOPY (PROP MAKEFILE-ENVIRONMENT LOOPSBACKWARDS))
```

;;; Maintain backwards compatibility with old versions of Loops. If not using old files, this need not be loaded by Loops. Indeed, it is recommended that
;;; the file not be used.

```
        (COMS
```

;;; Random stuff goes here

```
              (FNS GetLclStateInClass GetLocalStateInClass))
        (COMS
```

;;; Path browser

```
              (CLASSES PathBrowser)
              (METHODS LatticeBrowser.ListObjects PathBrowser.AddAndBox PathBrowser.AddToPath PathBrowser.Clear
                     PathBrowser.GetSubs PathBrowser.LeftShiftSelect PathBrowser.Recompute))
```

;;; Convert Koto Loops files to Lyric/Medley format

```
        (FNS ConvertLoopsFiles)
```

;;; Do as much renaming as possible --- This takes care of the change from List messages to ListAttribute messages

```
        (FNS ConvertChangedNames)
```

;;; Advise the reader to allow reading in old style read macros

```
        (FNS HashRead$ HashRead& HashReadLeftParen)
        (PROPS (%( HASHREADMACRO)
               (& HASHREADMACRO)
               ($ HASHREADMACRO))
```

;;; Some variables that will go away

```
        (VARS (TTY \TopLevelTtyWindow))
```

;;; Deal with old style UIDs

```
        (FNS ConvertUIDs ConvertOldUID \Convert-DEFINST)
        (ADVISE $& DEFINST $AV)
```

;;; Used to convert from Arcade release Loops files to Buttress release and beyond

```
        (FNS NewMethodFormat)
```

;;; Used to convert old '?' to '#.NotSetValue'

```
        (FNS ConvertNotSetValue ConvertClassNotSetValue ConvertInstanceNotSetValue ConvertValueNotSetValue)
```

;;; These are old names. Due to some possible conflicts, the names have been changed. Old code might still refer to these. The new messages are
;;; ListAttribute and ListAttribute!

```
        (METHODS Class.List Class.List! Object.List Object.List!)
```

;;; Old access functions. Use CHANGETRAN forms instead of these

```
        (FNS PushClassValue PushNewValue PushValue)
```

;;; Some old, unused methods

```
        (METHODS Object.At)
```

;;; Old window stuff

```
        (FNS GetLispWindow PutLispWindow InitLoopsWindow LoopsIconWindow)
```

;;; This stuff is needed to provide backwards compatability with old-style ActiveValues

```
        (RECORDS activeValue)
        (VARS ImplicitReplaceFns)
        (FNS GetActiveValueGetFn GetActiveValueLocalState GetActiveValuePutFn PutActiveValueGetFn
             PutActiveValueLocalState PutActiveValuePutFn)
        (METHODS ActiveValue.FetchGetFn ActiveValue.FetchPutFn ActiveValue.ReplaceGetFn ActiveValue.ReplacePutFn
               ExplicitFnActiveValue.FetchGetFn ExplicitFnActiveValue.FetchPutFn
               ExplicitFnActiveValue.ReplaceGetFn ExplicitFnActiveValue.ReplacePutFn
               ExplicitFnActiveValue.UpdateAV)
        (FNS ALISTUNION AttachListAP AtCreation CopyAnnotatedValue DataType FirstFetch GetFromIV GetIndirect
             GetLocalState! GetSuperClassValue GettingBrokenVariable GettingTracedVariable HasAV HasActiveGetFn
             HasActivePutFn MakeActiveValue NoUpdatePermitted PutIndirect PutLocalState PutLocalState!
             PutLocalStateOnly RemoteCall RemoveListAP ReplaceActiveValue ReplaceMe SendAVMessage
             SettingBrokenVariable SettingTracedVariable SnapLink StoreUnmarked SubstInAV Temporary UnSnapLink
             UnionSuperValue)
```

;;; Handle old-style forms like $class or @mumble

```
                                                                ; Reading in
        (FNS TRANS@$ Fix@$ AtMacro AtMacroConstruct DollarMacro)
        (ADDVARS (DWIMUSERFORMS (TRANS@$)))
                                                                ; Printing out
        (FNS DollarPrintOut)
```

;;; Old editor stuff

```
        (FNS EC EditClassSource EI EM SplitAtom)
        (MACROS @@ _@@)))

(FILESLOAD (FROM VALUEOF LOOPSLIBRARYDIRECTORY)
       LOOPSVCOPY)

(DECLARE%: DONTCOPY

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

;;; Maintain backwards compatibility with old versions of Loops. If not using old files, this need not be loaded by Loops. Indeed, it is recommended that ;;; the file not be used.

;;; Random stuff goes here

```
(DEFINEQ
```

(GetLclStateInClass

```
  [LAMBDA (av self varName propName type)                       (* smL "27-May-86 14:12")

          (* * Returns the local state of an activeValue)

    (ExtractRealValue self varName (GetIVHere av 'localState)
           propName type])
```

(GetLocalStateInClass

```
  [LAMBDA (av self varName propName type)                       (* smL "27-May-86 14:12")

          (* * Returns the local state of an activeValue)

    (ExtractRealValue self varName (GetIVHere av 'localState)
           propName type])

)
```

;;; Path browser

```
(DEFCLASSES PathBrowser)

(DEFCLASS PathBrowser (MetaClass Class doc "A mixin which shows paths and their locl context in a browser"
                           Edited%:                          (* dgb%: " 7-FEB-84 17:52"))
       (Supers LatticeBrowser)
       (InstanceVariables (pathObjects NIL doc "objects which have been put on the path")))

(\BatchMethodDefs)

(METH LatticeBrowser ListObjects (object)
      "return a list of the objects that are being displayed in the browser"
      (category (LatticeBrowser)))

(METH PathBrowser AddAndBox (obj objName)
      "Add selected object to path, and box it."
      (category (PathBrowser)))

(METH PathBrowser AddToPath (object)
      "Add a new object to the path"
      (category (PathBrowser)))

(METH PathBrowser Clear NIL "Empty PathBrowser" (category (Window)))

(METH PathBrowser GetSubs (obj)
      "Only get subs for elements on the path"
      (category (LatticeBrowser)))

(METH PathBrowser LeftShiftSelect (obj objName)
      "Add selected object to path, and box it"
      (category (LatticeBrowser)))

(METH PathBrowser Recompute (dontReshapeFlg)
      "Flip nodes which are on path"
      (category (LatticeBrowser)))

(Method ((LatticeBrowser ListObjects) self object)              ; dgb: 24-NOV-82 10:47
   "return a list of the objects that are being displayed in the browser"
   (AND (for node in (fetch GRAPHNODES of (WINDOWPROP (@ window)
                                                      'GRAPH))
           collect (CAR node))))

(Method ((PathBrowser AddAndBox) self obj objName)              ; smL 17-Sep-86 15:23
   "Add selected object to path, and box it."
   (_ self AddToPath obj)
   (AND obj (_ self BoxNode obj)))

(Method ((PathBrowser AddToPath) self object)                   ; smL 17-Sep-86 15:23
   "Add a new object to the path"
   (PROG NIL
         (COND
            ((NULL object)
             (RETURN (_ self Clear)))
            ((FMEMB object (@ pathObjects))
             (RETURN NIL)))
         (PushNewValue self 'pathObjects object)
         (_ self Recompute 'dontReshape))
   object)

(Method ((PathBrowser Clear) self)                              ; smL 17-Sep-86 15:23
   "Empty PathBrowser"
   [COND
      ((OR (@ pathObjects)
           (@ lastSelectedObject))
       (_@
        pathObjects NIL)
       (_@
        lastSelectedObject NIL)
       (_ self Recompute 'dontReshape])

(Method ((PathBrowser GetSubs) self obj)                        ; smL 17-Sep-86 15:23
   "Only get subs for elements on the path"
   (COND
      ((FMEMB obj (@ pathObjects))
       (_Super
        self GetSubs obj))))

(Method ((PathBrowser LeftShiftSelect) self obj objName)        ; smL 17-Sep-86 15:23
   "Add selected object to path, and box it"
   'AddAndBox)

(Method ((PathBrowser Recompute) self dontReshapeFlg)           ; smL 17-Sep-86 15:23
   "Flip nodes which are on path"
   (_Super
    self Recompute dontReshapeFlg)
   (for pathObj in (@ pathObjects) do (_ self FlipNode pathObj))
```

```
   self)

(\UnbatchMethodDefs)
```

;;; Convert Koto Loops files to Lyric/Medley format

```
(DEFINEQ

(ConvertLoopsFiles
  [LAMBDA (files load? cleanup? preButtress?)                    (* smL "10-Apr-87 11:07")

          (* * Convert old Loops files to new format)

    (for file in (OR (MKLIST files)
                     FILELST)
       bind badFiles goodFiles
       do (if [NULL (NLSETQ (PROGN (if load?
                                       then (LOAD file))
                                   (if preButtress?
                                       then (NewMethodFormat file))
                                   (ConvertNotSetValue file)
                                   (ConvertUIDs file)
                                   (ConvertChangedNames file)
                                   (if cleanup?
                                       then (MAKEFILE file '(NEW C ST]
              then (push badFiles file)
            else (push goodFiles file))
       finally (if goodFiles
                   then (printout NIL "Converted files " goodFiles T))
               (if badFiles
                   then (printout NIL "Could not convert files " badFiles T))
               (RETURN (LIST goodFiles badFiles])

)
```

;;; Do as much renaming as possible --- This takes care of the change from List messages to ListAttribute messages

```
(DEFINEQ

(ConvertChangedNames
  [LAMBDA (file)                                                 (* smL "17-Apr-87 13:57")

          (* * Use Masterscope to do any name changes required -
          For example, change any List messages to ListAttribute messages.)

    (LET ((fns (FILECOMSLST file)))
         (if fns
             then (printout NIL T "Analyzing functions on " file "...")
                  [MASTERSCOPE `(ANALYZE ANY IN ',fns]
                  (printout NIL T "Changing vars...")
                  [for var in '((TTY . \TopLevelTtyWindow))
                     do (MASTERSCOPE `(EDIT WHERE ANY IN ',fns USE ',(CAR var)
                                                                  -
                                                                  (R ,(CAR var)
                                                                     ,(CDR var]
                  (printout NIL T "Changing selectors...")
                  [for selector in '((At . Get)
                                     (List . ListAttribute)
                                     (List! . ListAttribute!))
                     do (MASTERSCOPE `(EDIT WHERE ANY IN ',fns SEND List - (R ,(CAR selector)
                                                                              ,(CDR selector]
                  (printout NIL T "Checking for direct use of Loops PromptWindows...")
                  (MASTERSCOPE `(EDIT WHERE ANY IN ',fns SEND GetPromptWindow))
                  (printout NIL T "Replacing calls to PushValue & others with CHANGETRAN calls...")
                  [for pushFn in '((PushValue push GetValue)
                                   (PushNewValue pushnew GetValue)
                                   (PushClassValue push GetClassValue))
                     do

          (* The IF edit command is needed because we may be looking at a macro that expands to a call to the pushFn)

                        (MASTERSCOPE `(EDIT WHERE ANY IN ',fns CALLS ',(CAR pushFn)
                                                                  -
                                                                  (IF (EQ ',(CAR pushFn)
                                                                          (CAR (%##)))
                                                                      (COMSQ (MOVE 4 TO N)
                                                                             (REPLACE 1 BY ,(CADDR pushFn))
                                                                             (INSERT ,(CADR pushFn)
                                                                                    BEFORE 1)
                                                                             (BI 2 -2))
                                                                    NIL]
                  (printout NIL T "Replacing calls to _@ with CHANGETRAN calls...")
                  [for setFn in '((_@
                                   change @))
                     do (MASTERSCOPE `(EDIT WHERE ANY IN ',fns CALLS ',(CAR setFn)
```

```
                                        -
                                        (IF (EQ ',(CAR setFn)
                                            (CAR (%##)))
                                        (COMSQ (REPLACE 1 BY ,(CADDR setFn))
                                               (INSERT ,(CADR setFn)
                                                       BEFORE 1)
                                               (BI 2 -2))
                                        NIL]
```

(* * Check for any direct use of fields that have been changed)

```
        [for fieldName in '(iDescrs)
           do (LET [(dirtyFns (MASTERSCOPE `(WHO USES THE FIELD ',fieldName]
                   (if dirtyFns
                       then (printout NIL T "The following functions use the field " fieldName ", and
                                    will need to be fixed by hand: " dirtyFns]
```

(* * Check for use or specialization of any method that has changed)

(* * That's all, folks)

```
        [MASTERSCOPE `(FORGET ANY IN ',fns]
        (printout NIL T])
```

)

;;; Advise the reader to allow reading in old style read macros

```
(DEFINEQ
```

(HashRead$

```
  [LAMBDA (fileHandle readTable)                                    (* edited%: "19-Feb-86 15:10")
```

(* %#$FOO causes the unit named FOO to be read in right now, and a pointer to it inserted in the list being read)

```
                                                                    (* Skip $)
    (LET ((name (READ fileHandle readTable)))
         (OR (GetObjectRec name)
             (COND
                ((STRINGP name)
                 (NewObject ($ Object)
                        name))
                ((LITATOM name)
                 (_ ($ Class)
                    New name])
```

(HashRead&

```
  [LAMBDA (fileHandle readTable)                                    (* smL "16-Jan-87 10:23")
```

(* * Read in an old-style Loops instance reference, of the form "#&(className uid)")

```
    (LET ((instDef (READ fileHandle readTable)))
         (ModifyInstance (LET ((className (CAR instDef))
                               (uid (CADR instDef)))
                          `(,className (,(if (STRINGP uid)
                                             then (ConvertOldUID uid)
                                         else uid])
```

(HashReadLeftParen

```
  [LAMBDA (fileHandle readTable)                                    (* edited%: "19-Feb-86 14:19")
    (APPLY (FUNCTION $A)
           (READ fileHandle readTable])
```

)

```
(PUTPROPS %( HASHREADMACRO HashReadLeftParen)

(PUTPROPS & HASHREADMACRO HashRead&)

(PUTPROPS $ HASHREADMACRO HashRead$)
```

;;; Some variables that will go away

```
(RPAQ TTY \TopLevelTtyWindow)
```

;;; Deal with old style UIDs

```
(DEFINEQ
```

(ConvertUIDs

```
  [LAMBDA (files load? cleanup?)                                    (* smL "19-May-86 18:47")
```

(* * Try to convert any instances on the file to the new style UIDs)

```
    (LET ((files (OR (MKLIST files)
                     FILELST)))
         (if load?
             then (for file in files do (LOAD file)))
         (for file in files do (for i in (FILECOMSLST file 'INSTANCES) when (STRINGP i)
                                  do (ADDTOFILE (ConvertOldUID i)
                                                'INSTANCES file i)
                                     (DELFROMFILE i 'INSTANCES file))
                               (if cleanup?
                                   then (APPLY* (FUNCTION CLEANUP)
                                                file])
```

(ConvertOldUID

```
  [LAMBDA (uid)                                                    (* smL "19-May-86 16:29")

          (* * Convert an old string UID into a new UID)

    (LET* [(sepr (CONSTANT (CHCON1 ".")))
           (idNumber (match (CHCON uid)
                            with
                            ($ =sepr $ =sepr $ =sepr *_ --]
          (if idNumber
              then (create UID
                           sessionID _ [SUBATOM uid 1 (DIFFERENCE (NCHARS uid)
                                                                  (ADD1 (LENGTH idNumber]
                           uidNumber _ (PACKC idNumber])
```

(\Convert-DEFINST

```
  [LAMBDA (DEFINST% FORM)                                          (* smL "15-Aug-86 11:20")

          (* * Some advise that allows us to read in old DEFINST forms)

    (for i in (CDDR DEFINST% FORM) do (for p on i by (CDDR p) when (EQ (CADR p)
                                                                       '?)
                                         do (change (CADR p)
                                                    NotSetValue)))
    (if [STRINGP (CAR (LAST (CADR DEFINST% FORM]
        then (CONS (CAR DEFINST% FORM)
                   (CONS [DREVERSE (CONS [ConvertOldUID (CAR (LAST (CADR DEFINST% FORM]
                                         (CDR (REVERSE (CADR DEFINST% FORM]
                         (CDDR DEFINST% FORM)))
      else DEFINST% FORM])

)

[XCL:REINSTALL-ADVICE '$& :BEFORE
       '((:LAST (COND
                   ([STRINGP (CAR (LAST (CADR nameOrUID]
                    (SETQ nameOrUID (CONS (CAR nameOrUID)
                                          (CONS [DREVERSE (CONS [ConvertOldUID (CAR (LAST (CADR nameOrUID]
                                                                (CDR (REVERSE (CADR nameOrUID]
                                                (CDDR nameOrUID]

[XCL:REINSTALL-ADVICE 'DEFINST :BEFORE '((:LAST (SETQ DEFINST% FORM (\Convert-DEFINST DEFINST% FORM]

[XCL:REINSTALL-ADVICE '$AV :BEFORE
       '((:LAST (COND
                   ([STRINGP (CAR (LAST (CADR L]
                    (SETQ L (CONS (CAR L)
                                  (CONS [DREVERSE (CONS [ConvertOldUID (CAR (LAST (CADR L]
                                                        (CDR (REVERSE (CADR L]
                                        (CDDR L]

(READVISE $& DEFINST $AV)


;;; Used to convert from Arcade release Loops files to Buttress release and beyond

(DEFINEQ
```

(NewMethodFormat

```
  [LAMBDA (files load? cleanup?)                                   ; Edited 15-Jun-88 16:49 by raf

          (* * Change method of storing methods on file so that methods are independent entities on file and methods will now be
          headed by a (Method ((ClassName selector) |...|)))

    (CLRHASH CLISPARRAY)
    (SETQ files (if (NULL files)
                    then FILELST
                  else (MKLIST files)))                            (* Load the file if we should)
    (if load?
        then (for file in files do (LOAD file)))
    (for file in files
       do [for C in (FILECOMSLST file 'CLASSES)
```

```
            do                                                    (* Make sure the methods and functions are on the same file)
               (for sel methName in (_ ($! C)
                                       ListAttribute
                                       'Methods)
                  do (SETQ methName (MethName C sel))
                     (COND
                        ((NULL (WHEREIS methName 'METHODS))
                         (LET [(fnFile (CAR (WHEREIS methName 'FNS]
                               (COND
                                  (fnFile (DELFROMFILE methName 'FNS fnFile)
                                          (ADDTOFILE methName 'METHODS fnFile]
                                                              (* Make sure each method has the correct format)
         (for methName in (FILECOMSLST file 'METHODS) bind methObj fnName args when (SETQ methObj ($! methName))
            do (SETQ fnName (GetValue methObj 'method))
               (SETQ args (CDR (ARGLIST fnName)))
               (change (@ methObj args)
                       args)                                  (* Make sure the function name is correct)
               (if (NEQ methName fnName)
                   then (PRINT (LIST 'Defining [CAR (DEFINE `((,methName (LAMBDA (self ,.args)
                                                                 (,fnName self ,.args]
                               'to
                               'call fnName))
                        (change (@ methObj method)
                                methName))                    (* Make sure the function format is correct)
               (CheckMethodForm ($! (GetValue methObj 'className))
                      (GetValue methObj 'selector)
                      methName))                              (* Make sure that the name of the method function is what is
                                                              should be)
         (if cleanup?
             then (APPLY* (FUNCTION CLEANUP)
                          file])

)


;;; Used to convert old '?' to '#.NotSetValue'

(DEFINEQ

(ConvertNotSetValue
  [LAMBDA (files load? cleanup?)                              (* edited%: "23-Apr-86 09:46")

          (* * Convert references to "?" in instances and classes to "#.NotSetValue")

    (DECLARE (SPECVARS ValuesBeingConverted))
    (LET ((files (OR (MKLIST files)
                     FILELST))
          (ValuesBeingConverted NIL))
         (if load?
             then (for file in files do (LOAD file)))
         (for file in files do (for type in '(CLASSES INSTANCES)
                                  do (for name in (FILECOMSLST file type) when [ConvertValueNotSetValue
                                                                                (COND
                                                                                   ((Object? name)
                                                                                    name)
                                                                                   (T ($! name]
                                        do (MARKASCHANGED name type)))
                                  (if cleanup?
                                      then (APPLY* (FUNCTION CLEANUP)
                                                   file])

(ConvertClassNotSetValue
  [LAMBDA (class)                                             (* smL "22-Apr-86 18:26")

          (* * Convert references to "?" in a class to "#.NotSetValue")

    (for ivName in (_ class ListAttribute 'IVs) bind changed?
       do [for prop in (CONS NIL (_ class ListAttribute 'IVProps ivName)) bind value
             do (SETQ value (GetClassIV class ivName prop))
                (if (EQ value '?)
                    then (PutClassIV class ivName NotSetValue prop)
                         (SETQ changed? T)
                  else (change changed? (OR DATUM (ConvertValueNotSetValue value]
       finally [for ivName in (_ class ListAttribute 'CVs)
                  do (for prop in (CONS NIL (_ class ListAttribute 'CVProps ivName)) bind value
                        do (SETQ value (GetCVHere class ivName prop))
                           (if (EQ value '?)
                               then (PutClassValueOnly class ivName NotSetValue prop)
                                    (SETQ changed? T)
                             else (change changed? (OR DATUM (ConvertValueNotSetValue value]
          (RETURN changed?])

(ConvertInstanceNotSetValue
  [LAMBDA (instance)                                          (* smL "22-Apr-86 18:21")
```

```
        (* * Convert references to "?" in an instance to "#.NotSetValue")

    (for ivName in (_ instance ListAttribute 'IVs) bind changed?
       do [for prop in (CONS NIL (_ instance ListAttribute 'IVProps ivName)) bind value
             do (SETQ value (GetIVHere instance ivName prop))
                (if (EQ value '?)
                    then (PutValueOnly instance ivName NotSetValue prop)
                         (SETQ changed? T)
                  else (change changed? (OR DATUM (ConvertValueNotSetValue value]
       finally (RETURN changed?])
```

(ConvertValueNotSetValue

```
  [LAMBDA (value)                                          (* edited%: "23-Apr-86 09:46")

        (* * Convert references to "?" in a value to "#.NotSetValue" -
        Returns T iff the value was changed.)

    (DECLARE (SPECVARS ValuesBeingConverted))
    (if (MEMB value ValuesBeingConverted)
        then NIL
      else (LET ((ValuesBeingConverted (CONS value ValuesBeingConverted)))
                (if (type? class value)
                    then (ConvertClassNotSetValue value)
                  elseif (type? annotatedValue value)
                    then (ConvertInstanceNotSetValue (fetch annotatedValue of value))
                  elseif (type? instance value)
                    then (ConvertInstanceNotSetValue value)
                  else NIL])

)


;;; These are old names.  Due to some possible conflicts, the names have been changed.  Old code might still refer to these.  The new messages are
;;; ListAttribute and ListAttribute!

(\BatchMethodDefs)

(METH Class List (type name)
      "Fn to list local parts of a class."
      (category (Object)))

(METH Class List! (type name verboseFlg)
      "Recursive version of List message. Omits things inherited from Object and Class unless verboseFlg is T.
      Sets it to T for Class and Object"
      (category (Object)))

(METH Object List (type name)
      "For type= IVs, list the iv names in instance. For IVProps lists IV properties for name found in instance.
      Otherwise lists properties inherited from class"
      (category (Object)))

(METH Object List! (type name verboseFlg)
      "Recursive form of List for objects. Omits things inherited from Object unless verboseFlg is T."
      (category (Object)))

(Method ((Class List) self type name)                      ; smL 10-Apr-87 11:10
   "Fn to list local parts of a class."
   (_ self ListAttribute type name))

(Method ((Class List!) class type name verboseFlg)         ; smL 10-Apr-87 11:19
   "Recursive version of List message. Omits things inherited from Object and Class unless verboseFlg is T. Sets
   it to T for Class and Object"
   (_ class ListAttribute! type name verboseFlg))

(Method ((Object List) self type name)                     ; smL 10-Apr-87 11:09
   "For type= IVs, list the iv names in instance. For IVProps lists IV properties for name found in instance.
   Otherwise lists properties inherited from class"
   (_ self ListAttribute type name))

(Method ((Object List!) self type name verboseFlg)         ; smL 10-Apr-87 11:11
   "Recursive form of List for objects. Omits things inherited from Object unless verboseFlg is T."
   (_ self ListAttribute! type name verboseFlg))

(\UnbatchMethodDefs)


;;; Old access functions.  Use CHANGETRAN forms instead of these

(DEFINEQ

## (PushClassValue

  [LAMBDA (self varName newValue propName)                 (* mjs%: "30-JUN-82 17:41")

        (* * Add new value to list that is value of a class variable.)

    (PutClassValue self varName (CONS newValue (GetClassValue self varName propName))
```

```
          propName])
```

(PushNewValue

```
  [LAMBDA (self varName newValue prop)                    (* dgb%: "17-JUN-82 15:37")
```

(* * Push new value onto list that is prop or value of instance variable.)

```
    (PROG ((oldValues (GetValue self varName prop)))
          (RETURN (OR (FMEMB newValue oldValues)
                      (PutValue self varName (CONS newValue oldValues)
                             prop])
```

(PushValue

```
  [LAMBDA (self varName item prop)                        (* dgb%: "14-MAR-83 16:02")
```

(* * Push new value onto list that is value of an instance variable or property.)

```
    (PutValue self varName (CONS item (LISTP (GetValue self varName prop)))
          prop])

)


;;; Some old, unused methods

(\BatchMethodDefs)

(METH Object At (varName prop)
      "Internal form of getValue"
      (category (Object)))

(Method ((Object At) self varName prop)                   ; smL 28-May-86 13:48
   "Internal form of getValue"
   (GetValue self varName prop))

(\UnbatchMethodDefs)


;;; Old window stuff

(DEFINEQ
```

(GetLispWindow

```
  [LAMBDA (self varName localSt propName activeVal type)  (* dgb%: "26-Apr-84 08:23")
```

(* This is a getFn for Window to insure a window is a real Lisp window.)

```
    (COND
       ((NOT (WINDOWP localSt))                           (* replace the local state with a window, associating this object
                                                          with the window)
        (SETQ localSt (PutLocalState activeVal (InitLoopsWindow self (_ self CreateWindow))
                             self varName propName type)))
       (T                                                 (* Ensure one is always using a Loops connected window)
          (WINDOWPROP localSt 'LoopsWindow self)))
    localSt])
```

(PutLispWindow

```
  [LAMBDA (self varName newValue propName activeVal type)  (* dgb%: "17-Apr-84 15:04")
                                                          (* Initializes the window with the correct button fns)
    (PutLocalState activeVal (InitLoopsWindow self (OR (WINDOWP newValue)
                                                       (ERROR newValue "should be window")))
          self varName propName type])
```

(InitLoopsWindow

```
  [LAMBDA (self window pos)                               (* smL "16-Apr-86 13:24")
```

(* * Initialize the Lisp window with correct EventFns)

```
    (AND pos (MOVEW window pos))
    (WINDOWPROP window 'LoopsWindow self)
    (_ self SetOuterRegion (WINDOWPROP window 'REGION)
       'NoUpdate)
    (WINDOWPROP window 'RIGHTBUTTONFN 'WindowRightButtonFn)
    (WINDOWPROP window 'BUTTONEVENTFN 'WindowButtonEventFn)
    (WINDOWPROP window 'AFTERMOVEFN 'WindowAfterMoveFn)
    (WINDOWPROP window 'RESHAPEFN 'WindowReshapeFn 'SimpleWindowReshapeFn)
    window])
```

(LoopsIconWindow

```
  [LAMBDA (self varName localSt propName activeVal type)  (* smL "18-Apr-86 13:43")
```

(* This is a getFn. The value of this getFn is returned as the value of the enclosing GetValue.)

```
    [OR (WINDOWP localSt)
        (PutValueOnly self varName (LET ((avObj (_ ($ ExplicitFnActiveValue)
                                                   New)))
                                         [SETQ localSt (InitLoopsWindow self (ICONW BlackLoopsIconBM
                                                                                    LoopsIconShadow
                                                                                    (create POSITION
                                                                                           XCOORD _
                                                                                           (OR (@ left)
                                                                                               LASTMOUSEX)
                                                                                           YCOORD _
                                                                                           (OR (@ bottom)
                                                                                               LASTMOUSEY]
                                         (change (@ avObj getFn)
                                                 'LoopsIconWindow)
                                         (change (@ avObj putFn)
                                                 'NoUpdatePermitted)
                                         (change (@ avObj localState)
                                                 localSt)
                                         (create annotatedValue
                                                 annotatedValue _ avObj]
    localSt])

)

;;; This stuff is needed to provide backwards compatability with old-style ActiveValues

(DECLARE%: EVAL@COMPILE

(ACCESSFNS activeValue ((localState GetActiveValueLocalState PutActiveValueLocalState)
                        (getFn GetActiveValueGetFn PutActiveValueGetFn)
                        (putFn GetActiveValuePutFn PutActiveValuePutFn))
       (CREATE (APPLY* (FUNCTION $A)
                       localState getFn putFn))
       (TYPE? (type? annotatedValue DATUM))
       (SYSTEM))
)

(RPAQQ ImplicitReplaceFns (AtCreation FirstFetch))

(DEFINEQ

(GetActiveValueGetFn
  [LAMBDA (activeValue)                                    (* smL "15-Apr-86 15:08")

          (* * The access fn for activeValues getFn)

    (_ (COND
          ((type? annotatedValue activeValue)
           (fetch annotatedValue of activeValue))
          (T activeValue))
       FetchGetFn])

(GetActiveValueLocalState
  [LAMBDA (activeValue)                                    (* smL " 6-May-86 15:13")

          (* * The access fn for activeValues localState)

    (_ (COND
          ((type? annotatedValue activeValue)
           (fetch annotatedValue of activeValue))
          (T activeValue))
       GetWrappedValueOnly])

(GetActiveValuePutFn
  [LAMBDA (activeValue)                                    (* smL "15-Apr-86 15:09")

          (* * The access fn for activeValues putFn)

    (_ (COND
          ((type? annotatedValue activeValue)
           (fetch annotatedValue of activeValue))
          (T activeValue))
       FetchPutFn])

(PutActiveValueGetFn
  [LAMBDA (activeValue newValue)                           (* smL "15-Apr-86 15:10")

          (* * The replace fn for activeValues getFn)

    (_ (COND
          ((type? annotatedValue activeValue)
           (fetch annotatedValue of activeValue))
```

```
          (T activeValue))
       ReplaceGetFn newValue])
```

(PutActiveValueLocalState

```
  [LAMBDA (activeValue newvalue)                          (* smL " 6-May-86 15:15")

          (* * The replace fn for activeValues localState)

    (_ (COND
          ((type? annotatedValue activeValue)
           (fetch annotatedValue of activeValue))
          (T activeValue))
       PutWrappedValueOnly newvalue])
```

(PutActiveValuePutFn

```
  [LAMBDA (activeValue newvalue)                          (* smL "15-Apr-86 15:09")

          (* * The replace fn for activeValues putFn)

    (_ (COND
          ((type? annotatedValue activeValue)
           (fetch annotatedValue of activeValue))
          (T activeValue))
       ReplacePutFn newvalue])

)

(\BatchMethodDefs)

(METH ActiveValue FetchGetFn NIL "Return the getFn for the active value, if there is one" (category (ActiveValue
                                                                                                     )))

(METH ActiveValue FetchPutFn NIL "Return the putFn for the active value, if there is one" (category (ActiveValue
                                                                                                     )))

(METH ActiveValue ReplaceGetFn (newGetFn)
      "Replace the getFn for the active value, if there is one"
      (category (ActiveValue)))

(METH ActiveValue ReplacePutFn (newPutFn)
      "Replace the putFn for the active value, if there is one"
      (category (ActiveValue)))

(METH ExplicitFnActiveValue FetchGetFn NIL "Specialization" (category (ActiveValue)))

(METH ExplicitFnActiveValue FetchPutFn NIL "Specialization" (category (ActiveValue)))

(METH ExplicitFnActiveValue ReplaceGetFn (newGetFn)
      "Specialization"
      (category (ActiveValue)))

(METH ExplicitFnActiveValue ReplacePutFn (newPutFn)
      "Specialization"
      (category (ActiveValue)))

(METH ExplicitFnActiveValue UpdateAV (propList)
      "New method template"
      (category (ExplicitFnActiveValue)))

(Method ((ActiveValue FetchGetFn) self)                   ; smL 25-Apr-86 17:34
   "Return the getFn for the active value, if there is one"
   NIL)

(Method ((ActiveValue FetchPutFn) self)                   ; smL 25-Apr-86 17:34
   "Return the putFn for the active value, if there is one"
   NIL)

(Method ((ActiveValue ReplaceGetFn) self newGetFn)        ; smL 25-Apr-86 17:37
   "Replace the getFn for the active value, if there is one"
   NIL)

(Method ((ActiveValue ReplacePutFn) self newPutFn)        ; smL 25-Apr-86 17:37
   "Replace the putFn for the active value, if there is one"
   NIL)

(Method ((ExplicitFnActiveValue FetchGetFn) self)         ; smL 20-Nov-85 14:37
   "Specialization"
   (@ getFn))

(Method ((ExplicitFnActiveValue FetchPutFn) self)         ; smL 20-Nov-85 14:47
   "Specialization"
   (@ putFn))

(Method ((ExplicitFnActiveValue ReplaceGetFn) self newGetFn)   ; smL 20-Nov-85 14:38
   "Specialization"
```

```
   (change (@ getFn)
           newGetFn))

(Method ((ExplicitFnActiveValue ReplacePutFn) self newPutFn)    ; smL 20-Nov-85 14:48
   "Specialization"
   (change (@ putFn)
           newPutFn))

(Method ((ExplicitFnActiveValue UpdateAV) self propList)        ; smL  5-Dec-85 11:37
   "New method template"
   [for p on propList by (CDDR p) do (SELECTQ (CAR p)
                                             (localState (change (@ localState)
                                                                 (LET ((newValue (CADR p)))
                                                                      (SELECTQ newValue
                                                                           (T NIL)
                                                                           (NIL DATUM)
                                                                           newValue))))
                                             (putFn (change (@ putFn)
                                                            (LET ((newValue (CADR p)))
                                                                 (SELECTQ newValue
                                                                      (T NIL)
                                                                      (NIL DATUM)
                                                                      newValue))))
                                             (getFn (change (@ getFn)
                                                            (LET ((newValue (CADR p)))
                                                                 (SELECTQ newValue
                                                                      (T NIL)
                                                                      (NIL DATUM)
                                                                      newValue))))
                                             (HELP "Unknown AV property" (CAR p]
   self)

(\UnbatchMethodDefs)

(DEFINEQ
```

(ALISTUNION

```
  [LAMBDA (L1 L2)                                                  (* smL " 8-May-86 14:10")
```

(* * Does not include any element on list L1 which is either on L2 or which has its car as elt of L2)

```
    (LET ((newList (CONS)))
         (for e in L1 when [AND (NOT (FMEMB e L2))
                                (OR (ATOM e)
                                    (NOT (FASSOC (CAR e)
                                                 L2]
           do (TCONC newList e))
         (CAR (LCONC newList L2])
```

(AttachListAP

```
  [LAMBDA (object ivName avProc prop receiver selector otherArgs)
                                                                   (* dgb%: " 5-APR-83 16:22")
```

(* Puts an active value put procedure avProc on the instance variable value exactly once, and adds value to the list on ivname%:,prop)

```
    (PROG (propVals currentVal)
          (SETQ propVals (LISTP (GetIVHere object ivName prop)))
          (SETQ currentVal (GetValueOnly object ivName))
          [COND
             ((NOT (HasActivePutFn currentVal avProc))
              (MakeActiveValue object ivName NIL avProc 'EMBED]
          (PutValueOnly object ivName (CONS (CONS receiver (CONS selector otherArgs))
                                            propVals)
                 prop)
          (RETURN (GetValue object ivName])
```

(AtCreation

```
  [LAMBDA (self varName locState propName av type)                 (* dgb%: "22-SEP-82 00:02")
```

(* * For active values -
will evaluate form in localState at first fetch request, and replace it as value of variable.
This will only happen for new variables, since old ones had value created at creation.)

```
    (ReplaceActiveValue av (COND
                              ((LISTP locState)
                               (EVAL locState))
                              (T (AVApply* locState self varName)))
          self varName propName type])
```

(CopyAnnotatedValue

```
  [LAMBDA (annotatedValue)                                         (* smL " 6-May-86 14:22")
```

(* * Copy an annotated value, being careful to copy any nested ones)

```
    (_AV
     annotatedValue CopyActiveValue annotatedValue])
```

(DataType

```
  [LAMBDA (self varName localSt propName activeVal type)        (* dgb%: "13-DEC-82 16:24")
                                                                (* Dummy to cause dumping of noncircular datatype in
                                                                instances)
    localSt])
```

(FirstFetch

```
  [LAMBDA (self varName locState propName av type)              (* dgb%: "22-SEP-82 00:04")

          (* * For active values -
          will evaluate form in localState at first fetch. Replaces active value with new value.
          locState will be evaled if it is a list, ele applied as a function name.)

    (ReplaceActiveValue av (COND
                             ((LISTP locState)
                              (EVAL locState))
                             (T (AVApply* locState self varName)))
           self varName propName type])
```

(GetFromIV

```
  [LAMBDA (self varName localSt propName activeVal type)        (* mjs%: "22-JUL-83 10:51")

          (* This is a getFn. The value of this getFn is returned as the value of the enclosing GetValue.)

    (GetValue self localSt])
```

(GetIndirect

```
  [LAMBDA (self varName ls propName activeVal type)             (* dgb%: "17-JUN-82 09:33")

          (* Expects the local state of the activeValue to be a list of the form
          (object varname propName) where propName is optional)

    (APPLY 'GetIt ls])
```

(GetLocalState!

```
  [LAMBDA (activeValObject)                                     (* smL "24-Sep-85 11:09")

          (* * Return the innermost local state of an activeValue)

    (LET [(ls (GetValueOnly activeValObject 'localState]
         (COND
            ((type? annotatedValue ls)
             (GetLocalState! (fetch annotatedValue of ls)))
            (T ls])
```

(GetSuperClassValue

```
  [LAMBDA (self varName propName activeVal)                     ; Edited 15-Jun-88 17:27 by raf
    (LET* [(class (COND
                      ((type? instance self)
                       (Class self))
                      (T self)))
           (superList (for sup on (CONS class (Supers class))
                         do (if (AND (_ (CAR sup)
                                        HasCV varName)
                                     (HasAV (GetCVHere (CAR sup)
                                                       varName propName)
                                            activeVal))
                                then (RETURN (CDR sup]
          (for c in superList bind value do [if [AND (_ c HasCV varName)
                                                     (NOT (NotSetValue (SETQ value (GetCVHere c varName propName]
                                                then (RETURN (ExtractRealValue c varName value propName
                                                                               'CV]
            finally (RETURN NotSetValue])
```

(GettingBrokenVariable

```
  [LAMBDA (self varName value propName av type)                 (* dgb%: "30-JUL-82 09:19")
    (BREAK1 value T GettingValue (?= (self varName propName value])
```

(GettingTracedVariable

```
  [LAMBDA (self varName value propName av)                      (* dgb%: "29-JUL-82 15:12")
    (BREAK1 value T TracedValue (TRACE ?= (self varName propName)
                                       GO])
```

(HasAV

```
[LAMBDA (value av)                                          (* smL " 8-May-86 15:08")

          (* * Returns T if activeVal is contained in value else NIL)

    (if (type? annotatedValue value)
        then (_AV
              value HasAV? av)
      elseif (AND (type? instance value)
                  (_ value Understands 'HasAV?))
        then (_ value HasAV? av)
      else NIL])
```

(HasActiveGetFn

```
[LAMBDA (value getF)                                        (* smL " 6-May-86 15:18")

          (* * Returns active Value containing getFn or NIL)

    (AND (type? annotatedValue value)
         (EQ getF (_AV
                   value FetchGetFn))
         value])
```

(HasActivePutFn

```
[LAMBDA (value putF)                                        (* smL " 6-May-86 15:19")

          (* * Returns active Value containing PutFn or NIL)

    (AND (type? annotatedValue value)
         (EQ putF (_AV
                   value FetchPutFn))
         value])
```

(MakeActiveValue

```
[LAMBDA (self varOrSelector newGetFn newPutFn newLocalSt propName type)
                                                            (* smL " 9-May-86 17:01")

          (* Makes the slot named varOrSelector of self be an active value, and puts in getfn and newPutFn if given.
          If the getFn, putFn, or localState as given are NIL, uses the old value.
          If newGetFn or newPutFn is T, then makes corresponding part of active value be NIL.
          Uses old active value if one was there, unless newLocalSt=EMBED.
          -
          This function is outdated. The newLocalSt arg is ignored, and assumed to be EMBED.)

    (LET ((newAV (_ ($ ExplicitFnActiveValue)
                    New)))
         (change (@ newAV getFn)
                 newGetFn)
         (change (@ newAV putFn)
                 newPutFn)
         (_ newAV AddActiveValue self varOrSelector propName type)
         newAV])
```

(NoUpdatePermitted

```
[LAMBDA (self entry value propName av)                      (* dgb%: " 4-MAR-82 16:34")
  (ERROR "No entry permitted on " (LIST self entry propName])
```

(PutIndirect

```
[LAMBDA (self varName newValue propName activeVal type)     (* dgb%: "17-JUN-82 09:54")

          (* Expects the local state of the activeValue to be a list of the form
          (object varname propName) where propName is optional)

    (PROG ((ls (GetLocalState activeVal)))
          (RETURN (PutIt (CAR ls)
                         (CADR ls)
                         newValue
                         (CADDR ls)
                         (CADDDR ls])
```

(PutLocalState

```
[LAMBDA (av newValue self varName propName type)            (* smL " 6-May-86 14:23")

          (* * Replaces the local state of an active value. Checks to see if the local state of the activeValue is itself active.
          -
          Works if av is an ActiveValue object or an annotatedValue datatype)

    (LET* ((avObject (if (AND (type? instance self)
                              (MEMB type '(NIL IV))
                              (NotSetValue (GetIVHere self varName propName)))
                         then
```

(* It is an active Value inherited from the class, copy it in the instance itself.)

```
                        (LET* ((classValue (GetClassIV (Class self)
                                                       varName propName))
                               (avCopy (_AV
                                        classValue CopyActiveValue classValue)))
                              (PutValueOnly self varName avCopy propName)
                              (fetch annotatedValue of avCopy))
                    elseif (type? annotatedValue av)
                      then (fetch annotatedValue of av)
                    else av))
          (localState (_ avObject GetWrappedValueOnly)))
         (if (type? annotatedValue localState)
             then (_ (fetch annotatedValue of localState)
                     PutWrappedValue self varName newValue propName type)
           else (_ avObject PutWrappedValueOnly newValue))
         newValue])
```

(PutLocalState!

```
[LAMBDA (activeValObject newls)                                    (* AAA "17-Feb-86 15:41")
```

(* * Replace the bottom level local state)

```
   (if (type? annotatedValue activeValObject)
       then (SETQ activeValObject (fetch annotatedValue of activeValObject)))
   (LET [(oldls (GetValueOnly activeValObject 'localState]
        (COND
           ((type? annotatedValue oldls)
            (PutLocalState! (fetch annotatedValue of oldls)
                   newls))
           (T (PutValueOnly activeValObject 'localState newls)
              activeValObject])
```

(PutLocalStateOnly

```
[LAMBDA (av newValue)                                              (* smL " 5-May-86 17:19")
```

(* * replaces the local state of an activeValue. Does not check whether the localState is active.)

```
   (_ (if (type? annotatedValue av)
          then (fetch annotatedValue of av)
        else av)
      PutWrappedValueOnly newValue])
```

(RemoteCall

```
[LAMBDA (call)                                                     (* smL " 4-Apr-86 19:02")
```

(* * A Remote call received here. Get object from UID, and apply method.
-
call is of form (objUID selector arg1 arg2 |...|))

```
   (DECLARE (LOCALVARS . T)
            (SPECVARS classForMethod))
   (LET [classForMethod (oBj (GetObjFromUID (CAR call]
        (APPLY (FetchMethodOrHelp oBj (CADR call))
               (CONS oBj (CDDR call])
```

(RemoveListAP

```
[LAMBDA (object ivName avProc prop value noErrorFlg)              (* smL "10-Oct-85 13:19")
```

(* Removes value on the list on ivname%:,prop for an active value, put procedure avProc on the instance.
Removes avProc if there are no more necessary attachments)

```
   (PROG (propVals av)
         (PutValueOnly object ivName [SETQ propVals (DELASSOC value (LISTP (GetIVHere object ivName prop]
                prop)
         (AND propVals (RETURN T))                                 (* still others left)
         (COND
            ((SETQ av (HasActivePutFn (GetValueOnly object ivName)
                                      avProc))                     (* none left. Remove the active value)
             (ReplaceActiveValue av (GetValueOnly av 'localState)
                    object ivName))
            (T (OR noErrorFlg (HELPCHECK avProc "not found on object"])
```

(ReplaceActiveValue

```
[LAMBDA (av newVal self varName propName type)                     (* smL " 6-May-86 14:18")
```

(* * Used to replace an active value in a potentially nested set of active values associated with some value or property.)

```
   (_ (if (type? annotatedValue av)
          then (fetch annotatedValue of av)
        else av)
      ReplaceActiveValue newVal self varName propName type])
```

(ReplaceMe

```
[LAMBDA (self varName newValue propName activeVal type)        (* dgb%: "21-SEP-82 22:28")
                                                               (* This is a putFn for ---)
   (ReplaceActiveValue activeVal newValue self varName propName type])
```

(SendAVMessage

```
[LAMBDA (self varName newValue propName activeVal)             (* dgb%: "17-FEB-83 18:52")

        (* This is a putFn for changing attached instruments. msg is of form
        (obj selector otherArgs)%. -
        newValue is filled in before otherArgs)

   (for msg in (GetIVHere self varName 'avMessages) bind (nv _ (PutLocalState activeVal newValue self varName
                                                                    propName))
                                                          classForMethod oBj
      do [APPLY (FetchMethodOrHelp (SETQ oBj (CAR msg))
                       (CADR msg))
                (CONS oBj (CONS nv (CDDR msg]
      finally (RETURN nv])
```

(SettingBrokenVariable

```
[LAMBDA (self varName newValue propName activeVal type)        (* dgb%: " 3-AUG-82 22:55")

        (* Go into a lisp break when a variable is to be set. OK will cause variable to be given value)

   (BREAK1 (PutLocalState activeVal newValue self varName propName type)
          T SettingVariable (?= (self varName propName newValue])
```

(SettingTracedVariable

```
[LAMBDA (self varName newValue propName activeValue)           (* dgb%: "29-JUL-82 15:17")
  (PROG ((oldValue (GetLocalState activeValue self varName newValue propName)))
        (RETURN (BREAK1 (PutLocalState activeValue newValue self varName propName)
                        T SettingVariable (TRACE ?= (self varName propName oldValue)
                                                 GO])
```

(SnapLink

```
[LAMBDA (self varName locState propName av type)               (* smL "24-Sep-85 17:40")

        (* * Will be used as a getFn for values which will not be fetched from the database until referenced)

   (PutValueOnly self varName (GetObjectRec (GetValueOnly (fetch annotatedValue of av)
                                                 'localState])
```

(StoreUnmarked

```
[LAMBDA (self varName newValue propName activeVal type)        (* smL "24-Sep-85 17:35")

        (* This is a putFn which replaces its value without marking the object as changed, or calling any active values)

   (PutValueOnly activeVal 'localState newValue])
```

(SubstInAV

```
[LAMBDA (oldAVObj newLS av)                                    (* smL "26-Feb-86 16:57")
                                                               (* Substitutes, copying, newLS for oldLS in a nested set of
                                                               active values.)
   (LET* [(avObj (fetch annotatedValue of av))
          (ls (GetValueOnly avObj 'localState]
         (COND
            ((EQ ls (@ oldAVObj localState))
             newLS)
            (T                                                 (* Nested value not found. This is just a copy of av)
               (LET ((newAV (_ avObj CopyShallow)))
                    (change (@ newAV localState)
                           (if (type? annotatedValue ls)
                               then (SubstInAV oldAVObj newLS ls)
                             else ls))
                    (create annotatedValue
                           annotatedValue _ newAV])
```

(Temporary

```
[LAMBDA (self varName newValue propName activeVal type)        (* smL "24-Sep-85 17:35")

        (* This is a putFn which replaces its value without marking the object as changed, or calling any active values)

   (PutValueOnly activeVal 'localState newValue])
```

(UnSnapLink

```
[LAMBDA (self varname propName)                                (* smL "24-Sep-85 10:09")
```

```
    (LET ((value (GetValueOnly self varname propName)))
         (COND
            ((type? instance value)
             (PutValueOnly self varname (APPLY* (FUNCTION $A)
                                                (UID value)
                                                'SnapLink
                                                'PutInValue)
                   propName)
             (UID value))
            (T NIL])
```

(UnionSuperValue

```
  [LAMBDA (self varName localSt propName activeVal type)        (* sm%: " 7-NOV-83 18:19")

          (* This is a getFn. The value of this getFn is returned as the value of the enclosing GetValue.)

    (ALISTUNION (LISTP (GetSuperClassValue self varName propName activeVal))
           (LISTP localSt])

)


;;; Handle old-style forms like $class or @mumble
;; Reading in

(DEFINEQ
```

(TRANS@$

```
  [LAMBDA NIL                                                   (* smL "20-May-86 14:05")

          (* * Fix atoms which start with @ and $ to be list form)

    (AND (LITATOM FAULTX)
         (Fix@$ FAULTX TAIL])
```

(Fix@$

```
  [LAMBDA (atom tail)                                           (* dgb%: " 3-JUN-83 15:01")
    (PROG (FORM ATOM (FIRSTCHAR (NTHCHAR atom 1)))
          (SELECTQ FIRSTCHAR
               (($ @)
                    [COND
                       ((SETQ ATOM (SUBATOM atom 2))            (* Used form $loopsName or @ivName)
                        (SETQ FORM (LIST FIRSTCHAR ATOM))
                        (AND (LISTP tail)
                             (RPLACA tail FORM)))
                       ((AND (EQ FIRSTCHAR '@)
                             (LISTP tail)
                             (LISTP (CAR tail)))                (* Tried to use old form @ (FOO FIE)%.
                                                                Put @ inside parens)
                        (SETQ FORM (CONS FIRSTCHAR (CAR tail)))
                        (RPLACA tail FORM)
                        (RPLACD tail (CDDR tail])
               NIL)
          (RETURN FORM])
```

(AtMacro

```
  [LAMBDA (fileHandle readTable)                                (* dgb%: "17-SEP-82 02:28")

          (* Causes the following transformations by a read-in macro. -
          @FOO -> (GetValue self (QUOTE FOO)) or (@ FOO) -
          @@FOO -> (GetClassValue self (QUOTE FOO)) or (@@ FOO) -
          @FOO_exp -> (PutValue self exp) or (_@ FOO exp) -
          @FOO_+exp -> (PushValue self exp) -
          @@FOO_ exp -> (PutClassValue self exp) -
          @@FOO_+ exp -> (PushClassValue self exp) -
          @ (X foo) -> (GetValue X (QUOTE foo)) or (@ X foo) -
          @ (X foo prop) -> (GetValue X (QUOTE foo) (QUOTE prop)) or
          (@ X foo prop) -
          @@ (x foo) -> (GetClassValue x (QUOTE foo)) or (@@ X foo prop) -
          @@ (x foo prop) -> (GetClassValue x (QUOTE foo) (QUOTE prop)) -
          or (@@ X foo prop) -
          |...| and similarly for the puts using _@ and _@@ with the newValue always last)
                                                                (* Also performs transformations for getting and putting
                                                                properties of class variables and instance variables.)
    (PROG (classVarFlg varName propName storeFlg newValueForm temp (objName 'self))
          (COND
             ((EQ '@ (PEEKC fileHandle))

          (* If next character is an "@" then this is a reference to a class variable)

              (SETQ classVarFlg T)
              (READC fileHandle)))
          (COND
             ((EQ (PEEKC fileHandle readTable)
```

```
                ’%()
              (SETQ temp (READ fileHandle readTable))
              (SETQ objName (CAR temp))
              (SETQ varName (CADR temp))
              (SETQ propName (CADDR temp)))
             [(NEQ ’OTHER (GETSYNTAX (PEEKC fileHandle)
                                  readTable))                    (* Quit if character after the At looks like LISP usage.)
              (RETURN (COND
                         (classVarFlg ’@@)
                         (T ’@]
             (T                                                  (* Here unless varName already ready with propName.)
                                                                 (* Temporarily make LeftArrow a BREAK character before
                                                                 reading expression.)
                (RESETSAVE (SETSYNTAX ’_ ’BREAKCHAR readTable)
                       (LIST ’SETBRK (GETBRK readTable)))
                (SETQ varName (RATOM fileHandle readTable))
                (SETSYNTAX ’_ ’OTHER readTable)))
          [COND
             ((EQ ’_ (PEEKC fileHandle readTable))
              (READC fileHandle readTable)
              (COND
                 ((EQ (PEEKC fileHandle readTable)
                      ’+)                                        (* This means to push value on front)
                  (READC fileHandle)
                  (SETQ storeFlg 1))
                 (T (SETQ storeFlg T)))
              (SETQ newValueForm (READ fileHandle readTable]
          (RETURN (AtMacroConstruct objName varName propName newValueForm classVarFlg storeFlg])
```

(AtMacroConstruct

```
  [LAMBDA (objName varName propName newValue classVarFlg storeFlg)
                                                                 (* dgb%: "17-SEP-82 02:37")
                                                                 (* Constructs the form needed for atMacro.
                                                                 See translation table in AtMacro)
    (COND
       [(NULL storeFlg)
        (NCONC [LIST (COND
                        (classVarFlg ’@@)
                        (T ’@]
               (COND
                  ((OR propName (NEQ ’self objName))
                   (LIST objName)))
               (LIST varName)
               (COND
                  (propName (LIST propName]
       [(EQ storeFlg 1)
        (NCONC (LIST (COND
                        (classVarFlg ’PushClassValue)
                        (T ’PushValue))
                     objName
                     (KWOTE varName))
               (LIST newValue)
               (COND
                  (propName (LIST (KWOTE propName]
       (T (NCONC [LIST (COND
                          (classVarFlg ’_@@)
                          (T ’_@]
                 (COND
                    ((OR propName (NEQ ’self objName))
                     (LIST objName)))
                 (LIST varName)
                 (COND
                    (propName (LIST propName)))
                 (LIST newValue])
```

(DollarMacro

```
  [LAMBDA (fileHandle readTable)                                 (* dgb%: " 8-Oct-84 13:28")

          (* Causes $FOO to be translated at READ time into (GetObjectRec
          (QUOTE FOO))%. The localState of this expression is the objectRec itself)

    (COND
       ((EQ (GETSYNTAX (PEEKC fileHandle readTable))
            ’OTHER)
        (LIST ’$ (READ fileHandle readTable)))
       (T ’$])

)

(ADDTOVAR DWIMUSERFORMS (TRANS@$))

;; Printing out

(DEFINEQ
```

(DollarPrintOut

```
  [LAMBDA (X)                                                    (* dgb%: "19-NOV-81 23:27")

          (* * This is called by PRETTYPRINT when it sees a form X starting with the atom GetClassRec.
          It causes this form to be printed out as "$name" in just those cases where the form is
          (GetObjectRec (QUOTE name)))

    (COND
       ((AND (LISTP (CDR X))
             (NULL (CDDR X)))
        (PRIN1 "$")
        (PRIN1 (CADR X))
        NIL)
       (T X])

)


;;; Old editor stuff

(DEFINEQ

(EC
  [LAMBDA (className coms)                                       (* mjs%: "21-JUL-82 08:11")
                                                                 (* Edit the symbolic class definition)
    (PROG ((name className))
          (COND
             ((NULL name)
              (SETQ name LASTCLASS)))
          [COND
             ((LISTP name)
              (SETQ name (CAR name]
          (COND
             ((NULL (SETQ name (GoodClassName name)))
              (ERROR className "not editable")))
          (RETURN (_ (GetClassRec name)
                     Edit coms])

(EditClassSource
  [LAMBDA (EXP COMS NAME)                                        (* dgb%: "29-NOV-82 13:38")
                                                                 (* Edit source description for class)
    (DECLARE (SPECVARS CHANGEDFLG NAME))
    (PROG (CHANGEDFLG)
      LP  [SETQ EXP (EDITE EXP COMS NAME 'CLASSES (FUNCTION (LAMBDA (ATM EXPR TYPE FLG)
                                                                (COND
                                                                   (FLG (SETQ CHANGEDFLG T))
                                                                   (T (RETFROM 'EC NAME]
          [COND
             (CHANGEDFLG                                         (* Here if the source has changed at all)
                     (COND
                        ((NULL (ERSETQ (EVAL EXP)))

          (* Evaluate form to install class. If there was an error, then value of ERSETQ was NIL.
          This implies a syntax error, so loop back to editor)

                         (SETQ COMS NIL)
                         (GO LP))
                        (T                                       (* Evaluation was succesful, class was changed.
                                                                 PutClass will mark the class object as changed.)
                           (PutClass (GetClassRec NAME)
                                  (EDITDATE NIL INITIALS)
                                  'Edited%:]
          (RETURN (SETQ LASTCLASS NAME])


(EI
  [LAMBDA (INST commands)                                        (* dgb%: "12-OCT-82 23:47")
    (_ (COND
          ((type? instance INST)
           INST)
          ((LITATOM INST)
           (GetObjectRec INST))
          (T (EVAL INST)))
       Edit commands])


(EM
  [LAMBDA (className selector commands)                          (* dgb%: " 6-DEC-83 11:19")
    (PROG (temp (cn className))
      LP  [COND
             ([NULL (SETQ cn (CAR (NLSETQ (GoodClassName cn NIL T]
              (OR (AND (NULL selector)
                       (SETQ temp (SplitAtom className '%.))
                       (SETQ cn (CAR temp))
                       (SETQ selector (CDR temp))
                       (GO LP))
```

```
                (ERROR cn "not a class name"]
          (RETURN (_ (GetClassRec cn)
                     EditMethod selector commands])
```

(SplitAtom

```
  [LAMBDA (atom splitChar)                                    (* dgb%: "22-JUN-82 11:14")
```

(* * Used to split method names etc at splitChar. Takes an atom and a char and returns a list of two atoms. e.g. SplitAtom (A.B %.) -> (A B) Returns NIL if splitChar is not in atom)

```
    (PROG ((pos (STRPOS splitChar atom)))
          (RETURN (COND
                     ((NULL pos)
                      NIL)
                     (T (CONS (SUBATOM atom 1 (SUB1 pos))
                              (SUBATOM atom (ADD1 pos])

)

(DECLARE%: EVAL@COMPILE

(PUTPROPS @@ MACRO (arg (Parse@ arg 'CV)))

(PUTPROPS _@@ MACRO (arg (ParsePut@ arg 'CV)))
)

(PUTPROPS LOOPSBACKWARDS COPYRIGHT ("Xerox Corporation" 1986 1987 1988))
```

FUNCTION INDEX

PROPERTY INDEX

VARIABLE INDEX

ADVICE INDEX

MACRO INDEX

RECORD INDEX