File created: 6-Jul-2022 15:03:42 **{DSK}<home>larry>loops>src>LOOPSSTRUC.;2**

changes to: (MACROS Class)

previous date: 15-Jun-93 13:59:37 {DSK}<home>larry>loops>src>LOOPSSTRUC.;1

Read Table: INTERLISP

Package: INTERLISP

Format: XCCS

;;
;; Copyright (c) 1983-1988, 1990-1991, 1993 by Venue & Xerox Corporation.

```
(RPAQQ LOOPSSTRUCCOMS
       (
```

;;; Basic records, operations and macros for dealing with implementation of classes and instances

;; Variables and constants used by LOOPS

```
(GLOBALVARS ErrorOnNameConflict ListMixinNames DefaultComment LASTCLASS LispClassTable NotSetValue
      NoValueFound ObjNameTable)
(ADDVARS (GLOBALVARS ErrorOnNameConflict ListMixinNames DefaultComment LASTCLASS LispClassTable
                NotSetValue NoValueFound ObjNameTable))
(INITVARS (NoValueFound NIL)
       (ListMixinNames)
       (DefaultComment)
       (OutInstances)
       (PPDefault T)
       (LASTCLASS)
       (ErrorOnNameConflict)
       (ObjNameTable (HASHARRAY 8000))
       (LispClassTable (HASHARRAY 16)))
                                                          ; Bootstrap value for NotSetValue.  This gets fixed later on in the
                                                          ; loadup.
(INITVARS (NotSetValue (create annotatedValue)))
```

;; Access macros for instances

```
(DECLARE%: EVAL@COMPILE DONTCOPY (FILES (LOADCOMP)
                                        LOOPSDATATYPES)
       (EXPORT (FUNCTIONS Once-Only)
              (FUNCTIONS WithIVValue ChangeIVValue)
              (RECORDS IVPropDescr)
              (MACROS FetchIVPropDescr)
              (FUNCTIONS MakeIVPropDescr WithIVPropDescr WithIVPropDescr!)
              (MACROS InstGetProp InstPutProp InstRemProp)
              (MACROS ExtractRealValue))
       DONTEVAL@COMPILE DOCOPY (INITRECORDS IVPropDescr))
```

;; Access macros for classes and the like

```
(DECLARE%: EVAL@COMPILE DONTCOPY (EXPORT (RECORDS IVDescr))
       DONTEVAL@COMPILE DOCOPY (INITRECORDS IVDescr))
(COMS (MACROS /PutNth Class ClassVariables FetchCIVDescr FetchCVDescr FetchIVDescr FindIndex GetCIVNth
             GetClassDescr GetInstanceIVValue GetNth GetVarNth NDescrs NotSetValue NoValueFound
             ValueFound ObjGetProp ObjPutProp ObjRemProp ObjSetValue PutIVDescr PutNth PutVarNth Supers)
      (FUNCTIONS GetInitialValue))
```

;; Access expressions

```
(MACROS $ @ @* _@)
(FNS Parse@ Parse@* ParseAccess ParseBang ParseExpr ParsePut@)
(PROP ARGNAMES @ _@)
(PROP INFO $ @ @* _@)
```

;; Interface Functions

```
(FNS CreateEntity DeleteObjectName FastClassInitialize GetIVHere GetValueOnly GetClassRec GetLispClass
     GetObjectName GetObjectRec MakeMixinClass NameEntity NewEntity PutObjectName PutValueOnly $!)
```

;; IV Lookup cache stuff

```
(DECLARE%: EVAL@COMPILE DONTCOPY (EXPORT (MACROS FindVarIndex FastFindIndex)))
```

;; Functions which build and change structure

```
(FNS BlankInstance BootNameObject FullNameObject FillInst FillIVs IVSource ModifyInstance NewClass
     NewObject UpdateClassIVs UpdateIVDescrs)
(P (MOVD? 'BootNameObject 'NameObject))
(ADDVARS (NLAMA DEFINST DEFINSTANCES DEFCLASS DEFCLASSES @ _@))
(I.S.OPRS IN-SUPERS-OF)))
```

;;; Basic records, operations and macros for dealing with implementation of classes and instances

;; Variables and constants used by LOOPS

```
(DECLARE%: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS ErrorOnNameConflict ListMixinNames DefaultComment LASTCLASS LispClassTable NotSetValue NoValueFound
```

```
       ObjNameTable)
)

(ADDTOVAR GLOBALVARS ErrorOnNameConflict ListMixinNames DefaultComment LASTCLASS LispClassTable NotSetValue
                          NoValueFound ObjNameTable)

(RPAQ? NoValueFound NIL)

(RPAQ? ListMixinNames )

(RPAQ? DefaultComment )

(RPAQ? OutInstances )

(RPAQ? PPDefault T)

(RPAQ? LASTCLASS )

(RPAQ? ErrorOnNameConflict )

(RPAQ? ObjNameTable (HASHARRAY 8000))

(RPAQ? LispClassTable (HASHARRAY 16))
```

;; Bootstrap value for NotSetValue. This gets fixed later on in the loadup.

```
(RPAQ? NotSetValue (create annotatedValue))
```

;; Access macros for instances

```
(DECLARE%: EVAL@COMPILE DONTCOPY

(FILESLOAD (LOADCOMP)
       LOOPSDATATYPES)
```

;; FOLLOWING DEFINITIONS EXPORTED

```
(DEFMACRO Once-Only (vars &BODY body)
   [LET ((Gensym-Var (GENSYM))
         (Run-time-Vars (GENSYM))
         (Run-time-Vals (GENSYM))
         (Expand-time-Val-Forms NIL))
        [SETQ Expand-time-Val-Forms (for var in vars
                                      collect `(COND
                                                  ([OR (LITATOM ,var)
                                                       (NUMBERP ,var)
                                                       (AND (LISTP ,var)
                                                            (MEMB (CAR ,var)
                                                                  ''FUNCTION]
                                                   ,var)
                                                  (T (LET ((,Gensym-Var (GENSYM)))
                                                          (push ,Run-time-Vars ,Gensym-Var)
                                                          (push ,Run-time-Vals ,var)
                                                          ,Gensym-Var]
        `(LET* (,Run-time-Vars ,Run-time-Vals (WRAPPED-BODY ([LAMBDA ,vars ., body]
                                                             ., Expand-time-Val-Forms)))
               (COND
                  ((NULL ,Run-time-Vars)
                   WRAPPED-BODY)
                  (T `([LAMBDA ,(DREVERSE ,Run-time-Vars)
                               (DECLARE (LOCALVARS . T))
                               ,WRAPPED-BODY]
                       .,
                       (DREVERSE ,Run-time-Vals])


(DEFMACRO WithIVValue (self ivName ifFound ifNotFound &OPTIONAL (cache NIL))
   [Once-Only (self ivName cache)
          (LET [(value (GENSYM 'value))
                (varDescr (GENSYM 'varDescr]
               (if cache
                   then `[if [AND (\GETBASEPTR ,cache 0)
                                  (EQ (\GETBASEPTR ,cache 0)
                                      (fetch (OBJECT VARNAMES) of ,self]
                              then (,ifFound ,self ,ivName (GetVarNth ,self (\GETBASEPTR ,cache 2))
                                             (\GETBASEPTR ,cache 2))
                             else (PROG (,value ,varDescr)
                                        (DECLARE (LOCALVARS . T))
                                        (SETQ ,varDescr (FindVarIndex ,ivName ,self))
                                        (COND
                                           (,varDescr (\PUTBASEPTR ,cache 0 (fetch (OBJECT VARNAMES)
                                                                                   of ,self))
                                                      (\PUTBASEPTR ,cache 2 ,varDescr)
                                                      (SETQ ,value (GetVarNth ,self ,varDescr))
                                                      (GO IVFound)))
```

```
                                    [SETQ ,varDescr (ASSOC ,ivName (fetch (instance otherIVs)
                                                                       of ,self]
                                    (COND
                                       (,varDescr (SETQ ,value (CDR ,varDescr))
                                              (GO IVFound)))
                                    (COND
                                       ((FIXP ,ivName)
                                        (SETQ ,value (FetchNthDescr ,self ,ivName))
                                        (GO IVFound)))
                                    (RETURN (,ifNotFound ,self ,ivName))
                                  IVFound
                                    (RETURN (,ifFound ,self ,ivName ,value ,varDescr]
                 else `(PROG (,value ,varDescr)
                             (DECLARE (LOCALVARS . T))
                             (SETQ ,varDescr (FindVarIndex ,ivName ,self))
                             (COND
                                (,varDescr (SETQ ,value (GetVarNth ,self ,varDescr))
                                       (GO IVFound)))
                             [SETQ ,varDescr (ASSOC ,ivName (fetch (instance otherIVs) of ,self]
                             (COND
                                (,varDescr (SETQ ,value (CDR ,varDescr))
                                       (GO IVFound)))
                             (COND
                                ((FIXP ,ivName)
                                 (SETQ ,value (FetchNthDescr ,self ,ivName))
                                 (GO IVFound)))
                             (RETURN (,ifNotFound ,self ,ivName))
                           IVFound
                             (RETURN (,ifFound ,self ,ivName ,value ,varDescr])


(DEFMACRO ChangeIVValue (self varName ivLoc newValue)
   [Once-Only (self varName ivLoc newValue)
          `(PROGN (COND
                     ((FIXP ,ivLoc)
                      (PutVarNth ,self ,ivLoc ,newValue))
                     ((LISTP ,ivLoc)
                      (RPLACD ,ivLoc ,newValue))
                     (T (SHOULDNT)))
                  ,newValue])

(DECLARE%: EVAL@COMPILE

(RECORD IVPropDescr (IVName . IVPropList)
       (SYSTEM))
)

(DECLARE%: EVAL@COMPILE

(PUTPROPS FetchIVPropDescr MACRO ((self ivName)
                                  (ASSOC ivName (fetch instIVProps of self))))
)


(DEFMACRO MakeIVPropDescr (self varName)
   `[CAR (push (fetch (instance instIVProps) of ,self)
               (create IVPropDescr
                       IVName _ ,varName])


(DEFMACRO WithIVPropDescr (self ivName ifFound ifIVNotFound &OPTIONAL (cache NIL))
   `(WithIVValue ,self ,ivName [LAMBDA (self ivName)
                               (,ifFound self ivName (FetchIVPropDescr self ivName]
           ,ifIVNotFound
           ,cache))


(DEFMACRO WithIVPropDescr! (self ivName ifFound ifIVNotFound &OPTIONAL (cache NIL))
   `(WithIVValue ,self ,ivName [LAMBDA (self ivName)
                               (,ifFound self ivName (OR (FetchIVPropDescr self ivName)
                                                         (MakeIVPropDescr self ivName]
           ,ifIVNotFound
           ,cache))

(DECLARE%: EVAL@COMPILE

(PUTPROPS InstGetProp MACRO [OPENLAMBDA (descr propName)
                            (COND
                               ((NULL descr)
                                NotSetValue)
                               (T (for tail on (fetch IVPropList of descr) declare%: (LOCALVARS . T)
                                     by (CDDR tail) do [COND
                                                          ((EQ propName (CAR tail))
                                                           (RETURN (CADR tail]
                                     finally (RETURN NotSetValue])

(PUTPROPS InstPutProp MACRO (OPENLAMBDA (descr propName value)
```

```
                                (if (NULL (fetch IVPropList of descr))
                                    then (change (fetch IVPropList of descr)
                                                 (LIST propName value))
                                         value
                                  else (LISTPUT (fetch IVPropList of descr)
                                                propName value))))

(PUTPROPS InstRemProp MACRO [OPENLAMBDA (descr propName)
```

(* * descr in an IVPropDescr with fields IVName and IVPropList Removes a property from that list.
RETURNS NIL if not found, propname otherwise)

```
                                (AND descr (LET ((propList (fetch IVPropList of descr)))
                                                (if (NULL propList)
                                                    then NIL
                                                  elseif (EQ propName (CAR propList))
                                                    then (replace IVProps of descr with (CDDR propList))
                                                         propName
                                                  else (for tail on (CDR propList) by (CDDR tail)
                                                          do (if (NULL (CDR tail))
                                                                 then (RETURN NIL)
                                                               elseif (EQ propName (CADR tail))
                                                                 then (RPLACD tail (CDDDR tail))
                                                                      (RETURN propName])
)

(DECLARE%: EVAL@COMPILE

(PUTPROPS ExtractRealValue MACRO (OPENLAMBDA (self varName value propName type)
```

(* Called by Fetches and Gets which want to notice activeValues.
type is one of NIL for instance variables, CLASS for class properties, METHOD for method properties and CV for class variables and properties. Returns either the value found or the result of evaluating the GETFN)

```
                                   (COND
                                      ((type? annotatedValue value)
                                       (_ (fetch annotatedValue of value)
                                          GetWrappedValue self varName propName type))
                                      (T value))))
)
DOCOPY)
```

;; END EXPORTED DEFINITIONS

;; Access macros for classes and the like

```
(DECLARE%: EVAL@COMPILE DONTCOPY
```

;; FOLLOWING DEFINITIONS EXPORTED

```
(DECLARE%: EVAL@COMPILE

(RECORD IVDescr (IVValue . IVProps)
       IVValue _ NotSetValue IVProps _ NIL (SYSTEM))
)
DOCOPY)
```

;; END EXPORTED DEFINITIONS

```
(DECLARE%: EVAL@COMPILE

(PUTPROPS /PutNth MACRO ((list index entry)
                         (/RPLACA (FNTH list index)
                                  entry)))

(PUTPROPS Class MACRO [OPENLAMBDA (self)
                        (COND
                           ((Object? self)
                            (fetch (OBJECT CLASS) of self))
                           (T (GetLispClass self])

(PUTPROPS ClassVariables MACRO ((self)
                                (APPEND (fetch (class cvNames) of self))))

(PUTPROPS FetchCIVDescr MACRO [(self varName)                    (* dgb%: "25-JAN-82 15:48")
```

(* * Find the description list for the named variable in a class, returning NIL if none is there.)

```
                                (PROG NIL                        (* Short circuit GetNth with embedded RETURN if no index found)
                                      (RETURN (GetNth (fetch VARDESCRS of self)
                                                      (OR (FindIndex varName (fetch VARNAMES of self))
                                                          (RETURN NIL])

(PUTPROPS FetchCVDescr MACRO [(classRec varName)
                              (PROG [(index (FindIndex varName (fetch cvNames of classRec]
```

```
                                    (RETURN (COND
                                               (index (GetNth (fetch cvDescrs of classRec)
                                                              index])

(PUTPROPS FetchIVDescr MACRO [OPENLAMBDA (self varName)
```

(* * Find the IVDescr for the named variable)

```
                                (LET (descr (varIndex (FindVarIndex varName self)))
                                     (DECLARE (LOCALVARS varIndex descr))
                                     (COND
                                        (varIndex (OR (GetVarNth self varIndex)
                                                      (create IVDescr)))
                                        [(SETQ descr (ASSOC varName (fetch otherIVs of self)))
```

(* * non standard instance variables are stored on an ALIST in otherIVs)

```
                                         (OR (CDR descr)
                                             (CDR (NCONC1 descr (create IVDescr]
                                        ((FIXP varName)
                                         (FetchNthDescr self varName])

(PUTPROPS FindIndex MACRO (OPENLAMBDA (entry table)
                            (for item in table as pos from 0 declare%: (LOCALVARS . T)
                               do (if (EQ item entry)
                                      then (RETURN pos))
                               finally (RETURN NIL))))

(PUTPROPS GetCIVNth MACRO (OPENLAMBDA (obj n)
                            (GetNth (fetch VARDESCRS of obj)
                                   n)))

(PUTPROPS GetClassDescr MACRO (OPENLAMBDA (class)
                                (CONS (fetch metaClass of class)
                                      (fetch otherClassDescription of class))))

(PUTPROPS GetInstanceIVValue MACRO [OPENLAMBDA (object ivName)
                                    (WithIVValue object ivName [LAMBDA (self varName value)
                                                                  (ExtractRealValue self varName value NIL
                                                                                    'IV]
                                          (LAMBDA (self varName)
                                            (_ self IVMissing varName NIL 'GetValue])

(PUTPROPS GetNth MACRO [OPENLAMBDA (table index)
                        (CAR (FNTH table (ADD1 index])

(PUTPROPS GetVarNth MACRO (OPENLAMBDA (obj n)
                            (\GETBASEPTR (fetch VARDESCRS of obj)
                                   (LLSH n 1))))

(PUTPROPS NDescrs MACRO (OPENLAMBDA (n)
                          (\ALLOCBLOCK n T)))

(PUTPROPS NotSetValue MACRO ((arg)
                             (EQ NotSetValue arg)))

(PUTPROPS NoValueFound MACRO ((arg)
                              (EQ NoValueFound arg)))

(PUTPROPS ValueFound MACRO ((arg)
                            (NEQ NoValueFound arg)))

(PUTPROPS ObjGetProp MACRO [OPENLAMBDA (descr propName)
```

(* Called by all fetch fns. Gets value in description list. Does not check for activeValues.)

```
                           (COND
                              ((NULL descr)
                               NotSetValue)
                              ((NULL propName)
                               (fetch IVValue of descr))
                              (T (for tail on (fetch IVProps of descr) by (CDDR tail)
                                    do [COND
                                          ((EQ propName (CAR tail))
                                           (RETURN (CADR tail]
                                    finally (RETURN NotSetValue])

(PUTPROPS ObjPutProp MACRO (OPENLAMBDA (descr propName value)
```

(* * Called to put a new value on a decr list by all the Store fns.
Adds property if no value there already.)

```
                           (COND
                              ((NULL propName)
                               (replace IVValue of descr with value))
                              (T (change (fetch IVProps of descr)
                                         (if (NULL DATUM)
```

```
                                        then (LIST propName value)
                                      else (LISTPUT DATUM propName value)
                                           DATUM))
                                value))))

(PUTPROPS ObjRemProp MACRO [OPENLAMBDA (descr propName)
```

(* * descr in an IVPropDescr with fields IVName and IVPropList Removes a property from that list.
RETURNS NIL if not found, propname otherwise)

```
                          (LET ((propList (fetch IVPropList of descr)))
                               (if (NULL propList)
                                   then NIL
                                 elseif (EQ propName (CAR propList))
                                   then (replace IVProps of descr with (CDDR propList))
                                        propName
                                 else (for tail on (CDR propList) by (CDDR tail)
                                         do (if (NULL (CDR tail))
                                                then (RETURN NIL)
                                              elseif (EQ propName (CADR tail))
                                                then (RPLACD tail (CDDDR tail))
                                                     (RETURN propName])

(PUTPROPS ObjSetValue MACRO [(self varName newValue descr aValue propName type)
```

(* Called by anyone who wants to set a value of a variable or property of any kind.
Does the checking for active values. The argument type is NIL for InstanceVariables, and otherwise is one of CV, CLASS, METHOD)

```
                        (COND
                           ((type? annotatedValue aValue)
                            (_ (fetch annotatedValue of aValue)
                               PutWrappedValue self varName newValue propName type))
                           (T (ObjPutProp descr propName newValue])

(PUTPROPS PutIVDescr MACRO [OPENLAMBDA (obj ivName ivDescr)
                            [WithIVValue obj ivName [LAMBDA (self varName oldValue loc)
                                                      (ChangeIVValue self varName loc (CAR ivDescr]
                                   (LAMBDA (self varName)
                                     (AddIV self varName (CAR ivDescr]
                            (if (CDR ivDescr)
                                then (WithIVPropDescr! obj ivName [LAMBDA (self varName propDescr)
                                                                    (replace IVPropList of propDescr
                                                                       with (CDR ivDescr]
                                           (LAMBDA (self varName)
                                             (HELPCHECK])

(PUTPROPS PutNth MACRO ((list index entry)
                        (RPLACA (FNTH list (ADD1 index))
                                entry)))

(PUTPROPS PutVarNth MACRO (OPENLAMBDA (obj n desc)
                            (\RPLPTR (fetch VARDESCRS of obj)
                                   (LLSH n 1)
                                   desc)))

(PUTPROPS Supers MACRO ((classRec)
                        (fetch supers of classRec)))
)

(DEFMACRO GetInitialValue (self varName ivDescr)

;;; Compute the initial value to store in the self iv

    `[for p on (fetch (IVDescr IVProps) of ,ivDescr) by CDDR declare%: (LOCALVARS . T)
        thereis (EQ (CAR p)
                    '%:initForm)
        finally (RETURN (if (NULL $$VAL)
                            then NotSetValue
                          else (LET ((self ,self)
                                     (varName ,varName))
                                    (DECLARE (SPECVARS self varName))
                                    (EVAL (CADR $$VAL])

;; Access expressions

(DECLARE%: EVAL@COMPILE

(PUTPROPS $ MACRO [name (LIST 'GetObjectRec (KWOTE (CAR name])

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

(PUTPROPS @* MACRO (accessPath (Parse@* accessPath)))

(PUTPROPS _@ MACRO (arg (ParsePut@ arg 'IV)))
)
```

```
(DEFINEQ
```

(Parse@

```
  [LAMBDA (exp type)                                        (* smL "13-Apr-87 16:05")

          (* exp is a list with 1, 2 or 3 elements. If 1, then it is a reference to a variable of self determined by type.
          If 2 elements, then the first one is an expression to be evaluated to give an object and the second like the previous single
          element. If 3 then it is like 2 but with a property of that variable specified)

    (LET (n val (v (CAR exp))
           (obj 'self))
         [COND
            ((CDR exp)
             (SETQ obj v)
             (SETQ v (CADR exp]
         (COND
            ((AND (NLISTP v)
                  (SETQ n (STRPOSL (CONSTANT (CHCON '_))
                                   v)))                     (* This has an embedded assignment)
             (ParsePut@ [LIST obj (SUBATOM v 1 (SUB1 n))
                              (COND
                                 ((EQ n (NCHARS v))         (* _ at end)
                                  (AND [NEQ v (SETQ val (CAR (LAST exp]
                                       (OR (AND (ATOM val)
                                                (ParseAccess val 'IV 'self 1))
                                           val)))
                                 (T (ParseAccess v 'IV 'self (ADD1 n]
                        type))
            (T (LET ((trans (ParseAccess v type obj 1))
                     (v (CDDR exp)))
                    [COND
                       (v                                   (* Now take care of property)
                          (SETQ trans (ParseAccess (CAR v)
                                                   'PROP trans 1]
                    (if (GREATERP (LENGTH trans)
                                  4)
                        then (ERROR "Invalid access expression" exp))
                    trans])
```

(Parse@*

```
  [LAMBDA (path objectExpr)                                 ; Edited 27-May-87 15:33 by smL

          (* * Parse@* parses (@* X ivName1 ivName2 ivName3) into (GetValue
          (GetValue (GetValue X (QUOTE ivName1)) (QUOTE ivName2))
          (QUOTE ivName3)) to)

    (COND
       ((NULL path)
        objectExpr)
       ((NULL objectExpr)
        (Parse@* (CDR path)
                 (CAR path)))
       (T (Parse@* (CDR path)
                   `(GetValue ,objectExpr ',(CAR path])
```

(ParseAccess

```
  [LAMBDA (input exprType currentExpr start)                ; Edited 27-May-87 15:33 by smL

          (* * Parse an expression containing %: (iv) |::| cv %:, prop and those followed by !.
          expr type is IV CV PROP MSG PERSP.)

    (DECLARE (SPECVARS input))
    (PROG [a e (n start)
           (nt exprType)
           (charList (CONSTANT (CHCON '|,.:|]
          (COND
             ((NULL input)                                  (* Dont Parse NIL)
              (RETURN))
             ((NOT (OR (ATOM input)
                       (STRINGP input)))                    (* needs to be evaluated, and not parsed)
              (SETQ a input)
              (GO MKEXPR)))
      BANGLOOP

          (* The following SELECTQ used to contain a branch ((\, (SETQ nt
          (QUOTE PERSP)))) (\, but) that got removed with the goings of perspectives)

          (SELECTQ (NTHCHAR input n)
               ($ (SETQ nt 'LOOPSNAME))
               (! [COND
                     ((EQ nt 'MSG)
                      (SETQ nt 'MSG!]
                  (SETQ e (ParseBang input 'IV currentExpr (ADD1 n)
                                     a))
```

```
                (SETQ a (CAR e))
                (SETQ n (CDR e))
                (GO MKEXPR))
            (\ (SETQ nt 'LISP))
            (%: (SELECTQ (NTHCHAR input (add n 1))
                     (%: (SETQ nt 'CV))
                     (%, (SETQ nt 'PROP))
                     (PROGN (SETQ nt 'IV)
                            (GO BANGLOOP))))
            (%. (SETQ nt 'MSG))
            (GO ENDSYMBOL))
        (add n 1)
        (GO BANGLOOP)
    ENDSYMBOL
        [SETQ a (SUBATOM input n (COND
                                    ((SETQ n (STRPOSL charList input n))
                                                          (* there is some infix operator)
                                     (SUB1 n))
                                    (T                    (* through the end of the input)
                                       NIL]               (* Create next level of nesting)
        (OR (EQ nt 'LISP)
            (SETQ a (KWOTE a)))
    MKEXPR
        (SETQ e (ParseExpr nt currentExpr a))
        (COND
           ((NULL n)                                      (* No further nesting)
            (RETURN e)))

        (* * Get new exprType. n will end up pointing to first char of operator.
        type is irrelevant)

        (RETURN (ParseAccess input NIL e n])
```

(ParseBang

```
  [LAMBDA (input exprType currentExpr start)                    ; Edited 27-May-87 15:33 by smL

        (* Parse an expression after a bang containing %: (iv) |::| cv %:, prop and those followed by !.
        expr type is IV CV PROP MSG PERSP.)

    (DECLARE (SPECVARS input))
    (PROG [a (n start)
           (nt exprType)
           (charList (CONSTANT (CHCON '|,.:|]
        (SELECTQ (NTHCHAR input n)
            (\ (SETQ nt 'LISP))
            (%: (SELECTQ (NTHCHAR input (add n 1))
                     (%: (SETQ nt 'CV))
                     (%, (ERROR "Can't have a property at top level"))
                     (PROGN (SETQ nt 'IV)
                            (GO ENDSYMBOL))))
            (%, (ERROR "Can't use perspective as name"))
            (%. (SETQ nt 'MSG))
            (GO ENDSYMBOL))
        (add n 1)
    ENDSYMBOL
        [SETQ a (SUBATOM input n (COND
                                    ((SETQ n (STRPOSL charList input n))
                                                          (* there is some infix operator)
                                     (SUB1 n))
                                    (T                    (* through the end of the input)
                                       NIL]               (* Create next level of nesting)
    MKEXPR
        (RETURN (CONS [ParseExpr nt 'self (COND
                                              ((EQ nt 'LISP)
                                               a)
                                              (T (KWOTE a]
                      n])
```

(ParseExpr

```
  [LAMBDA (type currentExpr firstAtom)                          ; Edited 27-May-87 15:33 by smL

        (* * Sub-function of ParseAccess)

    (DECLARE (SPECVARS input))

        (* The following SELECTQ used to contain a branch (PERSP (LIST
        (QUOTE _) currentExpr (QUOTE GetPersp) firstAtom)) (\, but) that got removed with the goings of perspectives)

    (SELECTQ type
        (LISP (COND
                  ((NEQ currentExpr 'self)
                   (ERROR "\ must follow ! or appear at beginning" input)))
              firstAtom)
        (LOOPSNAME (COND
                       ((NEQ currentExpr 'self)
```

```
                        (ERROR "$ must appear at beginning" input)))
                   (LIST 'GetObjectRec firstAtom))
        (IV (LIST 'GetValue currentExpr firstAtom))
        (CV (LIST 'GetClassValue currentExpr firstAtom))
        (PROP (NCONC1 currentExpr firstAtom))
        (MSG! (LIST '_!
                    currentExpr firstAtom))
        (MSG (LIST '_ currentExpr (CADR firstAtom)))
        (ERROR "Invalid form in access expression:" firstAtom])
```

(ParsePut@

```
  [LAMBDA (arg type)                                          (* smL "13-Apr-87 16:04")
                                                              (* Produce translation for _@ and _@@)
    (PROG (first trans (last (LAST arg)))
          (SETQ trans (Parse@ (LDIFF arg last)
                              type))
          (RPLACA trans (SELECTQ (SETQ first (CAR trans))
                                 (GetValue 'PutValue)
                                 (GetClassValue
                                     'PutClassValue)
                                 (GO CHECKSEND)))
          (RPLACD (CDDR trans)
                  (CONS (CAR last)
                        (CDDDR trans)))
          (RETURN trans)
      CHECKSEND
          (COND
             ((AND (EQ first '_)
                   (EQ (CADDR trans)
                       'GetPersp))
              (RPLACA (CDDR trans)
                      'AddPersp)
              (RPLACD (CDDDR trans)
                      (CONS (CAR last)
                            (CDDDDR trans)))
              (RETURN trans)))
          (SHOULDNT "Bad translation in Parse@"])

)

(PUTPROPS @ ARGNAMES ({object} accessPath))

(PUTPROPS _@ ARGNAMES ({object} accessPath newValue))

(PUTPROPS $ INFO (NOEVAL))

(PUTPROPS @ INFO (NOEVAL))

(PUTPROPS @* INFO (NOEVAL))

(PUTPROPS _@ INFO (NOEVAL))

;; Interface Functions

(DEFINEQ
```

(CreateEntity

```
  [LAMBDA (obj uid)                                           ; Edited 15-Aug-90 13:01 by jds

          (* * Creates an entity for the object, putting in uid in entity and obj.)

    (OR (Object? obj)
        (HELPCHECK obj "not object for CreateEntity"))
    (SETQ uid (if (NULL uid)
                  then (Make-UID)
                else uid))
    (PutObjectUID uid obj)
    (replace (OBJECT OBJUID) of obj with uid)
    uid])
```

(DeleteObjectName

```
  [LAMBDA (obj name)                                          (* smL " 9-Apr-87 17:10")

          (* * In ObjNameTable Delete name as name of object, or all names for object if name is NIL)

    (LET [(allNames (GETHASH obj ObjNameTable))
          (type (if (Class? obj)
                    then 'CLASSES
                  else 'INSTANCES]
         (COND
            ((NULL allNames)
             NIL)
            ((NULL name)
             (PUTHASH obj NIL ObjNameTable)
             (for n inside allNames do (PUTHASH n NIL ObjNameTable)
```

```
                                    (SELECTQ type
                                        (CLASSES (if (LISTP n)
                                                     then          (* Remove the dynamic mixin name)
                                                          (change ListMixinNames (REMOVE n DATUM))))
                                        NIL))                      (* Return nonNIL so marking of change will take place)
            allNames)
           ((EQ name allNames)
            (PUTHASH obj NIL ObjNameTable)
            (PUTHASH name NIL ObjNameTable)
            name)
           ((OR (LITATOM allNames)
                (NOT (MEMBER name allNames)))
            (ERROR name "Not a name for object"))
           (T (PUTHASH obj (REMOVE name allNames)
                      ObjNameTable)
              (PUTHASH name NIL ObjNameTable)
              (SELECTQ type
                   (CLASSES (if (LISTP name)
                                then                               (* Remove the dynamic mixin name)
                                     (change ListMixinNames (REMOVE name DATUM))))
                   NIL)
              name])
```

(FastClassInitialize

```
[LAMBDA (class self)                                               ; Edited 15-Aug-90 13:00 by jds
```

(* * Fill in the IVs with initial values)

```
    (DECLARE (LOCALVARS . T))
    (for varName in (fetch (class ivNames) of class) as descr in (fetch (class ivDescrs) of class)
       do (WithIVValue self varName [LAMBDA (self varName oldValue loc)
                                    (if (NotSetValue oldValue)
                                        then (for p on (fetch (IVDescr IVProps) of descr)
                                                by (CDDR p) when (EQ (CAR p)
                                                                     '%:initForm)
                                                do (ChangeIVValue self varName loc (EVAL (CADR p)))
                                                   (RETURN]
                (LAMBDA (self varName)
                  NIL])
```

(GetIVHere

```
[LAMBDA (self varName propName)                                    (* smL "27-May-86 18:36")
```

(* * Gets the value found in the instance, without invoking activeValues.
Returns NotSetValue if not found in instance)

```
    (COND
       ((NOT (type? instance self))
        (_ (GetLispClass self)
           GetIVHere self varName propName))
       [propName (WithIVPropDescr self varName [LAMBDA (self varName propDescr)
                                                  (InstGetProp propDescr propName]
                         (LAMBDA (self varName)
                           NotSetValue]
       (T (WithIVValue self varName [LAMBDA (self varName value)
                                    value]
                 (LAMBDA (self varName)
                   NotSetValue])
```

(GetValueOnly

```
[LAMBDA (self varName propName)                                    (* smL "10-Apr-87 10:48")
```

(* * Like GetValue except that it ignores the special status of ActiveValues and just returns them as a data structure without activating any procedures)

```
    (COND
       ((type? class self)
        (GetClassIV self varName propName))
       ((NOT (type? instance self))
        (GetItOnly self varName propName 'IV))
       ((FIXP varName)                                             (* Here for indexed variables.)
        (FetchNthValueOnly self varName propName))
       [propName                                                   (* Get an IV prop from an instance)
              (WithIVPropDescr self varName [LAMBDA (self varName propDescr)
                                              (LET ((value (InstGetProp propDescr propName)))
                                                   (COND
                                                      ((OR (NULL propDescr)
                                                           (NotSetValue value))
                                                       (_ self IVValueMissing varName propName 'GetValueOnly))
                                                      (T value]
                     (LAMBDA (self varName)
                       (_ self IVMissing varName propName 'GetValueOnly]
       (T                                                          (* usual case)
          (WithIVValue self varName [LAMBDA (self varName value)
```

```
                                    (COND
                                       ((NotSetValue value)
                                        (_ self IVValueMissing varName NIL 'GetValueOnly))
                                       (T value]
                 (LAMBDA (self varName)
                   (_ self IVMissing varName NIL 'GetValueOnly])
```

(GetClassRec

```
  [LAMBDA (className)                                   (* dgb%: "29-Feb-84 15:35")
```

(* Given an atom, returns the class which is named by that atom.
If there is no such definition, returns NIL)

```
    (COND
       ((type? class className)
        className)
       (T (PROG ((classRec (GetObjectRec className)))
                (COND
                   ((AND classRec (NOT (type? class classRec)))
                    (LoopsHelp className " is a defined object, but is not a class.")))
                (RETURN classRec])
```

(GetLispClass

```
  [LAMBDA (obj)                                         (* smL " 6-May-86 14:46")
```

(* * Gets the class corresponding to the Lisp object as specified in LispClassTable)

```
    (LET ((convertForm (GETHASH (TYPENAME obj)
                                LispClassTable)))
         (if (NULL convertForm)
             then ($ Tofu)
          elseif (type? class convertForm)
             then convertForm
          else (APPLY* convertForm obj])
```

(GetObjectName

```
  [LAMBDA (object)                                      (* smL " 2-Apr-86 15:03")
                                                        (* Returns the name of an object if it has one other than its UID)
    (COND
       ((NULL object)
        NIL)
       ((type? class object)
        (ClassName object))
       (T (LET ((names (GETHASH object ObjNameTable)))
               (COND
                  ((LITATOM object)                     (* This is the name of an object)
                   (AND names object))
                  ((LISTP names)                        (* If it has more than one, then return first)
                   (CAR names))
                  (T names])
```

(GetObjectRec

```
  [LAMBDA (name)                                        ; Edited 25-Nov-87 14:20 by Bane
```

;;; Given a name (or UID) returns the object which is named by that atom. else returns NIL.

```
    (COND
       ((Object? name)
        name)
       ((UIDP name)
        (GetObjFromUID name))
       (T ;; If name is a list Return a class which has this list as supers provided all are classes.
          (OR (GETHASH name ObjNameTable)
              (AND (LISTP name)
                   (MakeMixinClass name])
```

(MakeMixinClass

```
  [LAMBDA (nameList)                                    (* smL " 8-Apr-87 19:22")
```

(* * Make a class with supers specified by nameList, and with canonical name also specified by nameList)

```
    (DECLARE (GLOBALVARS ListMixinNames))
    (if [for c in nameList always (AND (LITATOM c)
                                       (type? class ($! c]
        then (LET [(canonicalClassName (CAR (MEMBER nameList ListMixinNames]
                  (COND
                     ((NOT canonicalClassName)          (* We have never seen this dynamic class, so create it anew)
```

(* Be careful that we name a new list for the canonical name, so the user can't smash it by accident)

```
                      (LET* ((nameListCopy (COPYALL nameList))
```

```
                        (class (NewClass nameListCopy)))
                       (InstallSupers class nameList)
                       (push ListMixinNames nameListCopy)
                       class))
                (($! canonicalClassName))
                (T                                          (* We thought we had a class by that name already, but we
                                                            didn't)
                   (change ListMixinNames (REMOVE canonicalClassName ListMixinNames))
                   (MakeMixinClass nameList])
```

(NameEntity

```
[LAMBDA (self name)                                         (* smL "18-Sep-86 10:17")
```

(* * Associate a name with entity in current environment An object can have more than one name. RETURN NIL if already has name)

```
(COND
   ((NULL name)
    (ERROR "Can't name object NIL" self))
   ((NOT (LITATOM name))
    (ERROR name "should be an atom to be a name"))
   (T (LET ((oldObj (GetObjectRec name)))
           (COND
              ((EQ self oldObj)                             (* already has name)
               NIL)
              (T (COND
                    ((AND oldObj (NEQ self oldObj))
                     (AND ErrorOnNameConflict (HELPCHECK name "is already used as a name for an object.  To
                                                          continue type OK."))
                     (_ oldObj UnSetName name)))
                 (PutObjectName name self)
                 self])
```

(NewEntity

```
[LAMBDA (facts names)                                       (* dgb%: "26-DEC-83 15:00")
```

(* * Creates a new entity and names it if name given)

```
[COND
   ((NULL names)                                            (* No UID given so just create the entity)
    (CreateEntity facts))
   ((LITATOM names)                                         (* Only a single name given)
    (CreateEntity facts)
    (PutObjectName names facts))
   (T (PROG ((allNames (REVERSE names)))                    (* Now uid is first on allNames)
            (CreateEntity facts (pop allNames))
            (for name in allNames do (PutObjectName name facts]
facts])
```

(PutObjectName

```
[LAMBDA (name obj)                                          (* smL " 8-Apr-87 19:18")
```

(* * Puts a new name for object in nameTable)

```
(PROG (newNames oldNames)                                   (* First make sure object has a UID if it is to be named)
      (UID obj)
      (SETQ oldNames (GETHASH obj ObjNameTable))
      [SETQ newNames (COND
                        ((NULL oldNames)
                         (if (LISTP name)
                             then (LIST name)
                           else name))
                        ((AND (LITATOM oldNames)
                              (LISTP name))
                         (LIST name oldNames))
                        ((LITATOM oldNames)
                         (COND
                            ((EQ name oldNames)             (* already has name)
                             (RETURN NIL)))
                         (LIST name oldNames))
                        ((MEMBER name oldNames)             (* already has name)
                         (RETURN NIL))
                        (T (CONS name oldNames]
      (PUTHASH name obj ObjNameTable)
      (PUTHASH obj newNames ObjNameTable])
```

(PutValueOnly

```
[LAMBDA (self varName newValue propName)                    (* smL " 9-Jun-86 19:18")
```

(* * Puts newValue in an varNmae value or property. Overwrites any existing value, even if it is an activeValue.)

```
(COND
   ((type? class self)                                      (* Error check for class)
```

```
        (PutClassIV self varName newValue propName))
       [(type? instance self)
        (COND
           [propName (WithIVPropDescr! self varName [LAMBDA (self varName propDescr)
                                                      (InstPutProp propDescr propName newValue]
                           (LAMBDA (self varName)
                             (_ self IVMissing varName propName 'PutValueOnly newValue]
           (T (WithIVValue self varName [LAMBDA (self varName oldValue loc)
                                         (ChangeIVValue self varName loc newValue]
                     (LAMBDA (self varName)
                       (_ self IVMissing varName NIL 'PutValueOnly newValue]
       (T (_ (GetLispClass self)
             PutValueOnly self varName newValue propName])
```

($!

```
  [LAMBDA (name)                                                    (* smL "19-May-86 14:43")
```

(* * Given a name (or UID) returns the object which is named by that atom.
else returns NIL.)

```
    (GetObjectRec name])

)

;; IV Lookup cache stuff

(DECLARE%: EVAL@COMPILE DONTCOPY

;; FOLLOWING DEFINITIONS EXPORTED

(DECLARE%: EVAL@COMPILE

(PUTPROPS FindVarIndex MACRO (OPENLAMBDA (name obj)
                               (FastFindIndex name (fetch VARNAMES of obj))))

(PUTPROPS FastFindIndex MACRO [OPENLAMBDA (ivName ivList)
```

(* cacheBlockPos is an even postion on the page, computed from the xor of the ivName and the IVList addresses)

```
                               (PROG ((ivIndex 0)
                                      (IVList ivList)
                                      (cacheBlockPos (LLSH (LOGAND 1023 (LOGXOR (\LOLOC ivList)
                                                                                (\LOLOC ivName)))
                                                           3)))
                                    (DECLARE (LOCALVARS . T))
                                    [COND
                                       ((AND (EQ IVList (\GETBASEPTR *Global-IV-Cache-Block* cacheBlockPos))
                                             (EQ ivName (\GETBASEPTR (\ADDBASE *Global-IV-Cache-Block* 2)
                                                                     cacheBlockPos)))
                                        (RETURN (\GETBASEPTR (\ADDBASE *Global-IV-Cache-Block* 4)
                                                             cacheBlockPos]
                                 TryNextIV
                                    (COND
                                       ((EQ ivName (CAR IVList))
                                        (\PUTBASEPTR *Global-IV-Cache-Block* cacheBlockPos ivList)
                                        (\PUTBASEPTR (\ADDBASE *Global-IV-Cache-Block* 2)
                                               cacheBlockPos ivName)
                                        (\PUTBASEPTR (\ADDBASE *Global-IV-Cache-Block* 4)
                                               cacheBlockPos ivIndex)
                                        (RETURN ivIndex))
                                       ((NULL (SETQ IVList (CDR IVList)))
                                        (RETURN NIL))
                                       (T (add ivIndex 1)
                                          (GO TryNextIV])
)
)

;; END EXPORTED DEFINITIONS
;; Functions which build and change structure

(DEFINEQ
```

(BlankInstance

```
  [LAMBDA (class obj unmodifiedFlg)                                 ; Edited 15-Aug-90 13:02 by jds
```

(* * Make this instance be a blank with structure determined from the class.
If unmodifiedFlg=T then a newly read in instance.)

```
    (DECLARE (LOCALVARS . T))
    (LET* ((objUID (AND obj (fetch (instance instUnitRec) of obj)))
           (obj (OR obj (create instance)))
           [class (OR (AND class (OR (GetClassRec class)
                                     (LoopsHelp class "Specified but has no class to create object")))
                      (OR (fetch (OBJECT CLASS) of obj)
```

```
                        ($ Object]
          (ivnms (fetch (class ivNames) of class))
          (numberIVs (FLENGTH ivnms)))
         (replace (instance class) of obj with class)
         (replace (instance iNames) of obj with ivnms)
         (replace (instance instMiscField) of obj with NIL)
         (replace (instance iDescrs) of obj with (NDescrs numberIVs))
         (for i from 0 to (SUB1 numberIVs) do (PutVarNth obj i NotSetValue))
         (if objUID
             then (replace (instance instUnitRec) of obj with objUID))
        obj])
```

(BootNameObject

```
[LAMBDA (object names)                                    (* smL "19-May-86 19:12")
```

(* * This function gets replaced by the function FullNameObject during the Loops loadup.)

```
  [COND
     ((NLISTP names)
      NIL)
     (T (for name in names when (NOT (UIDP name)) do (PutObjectName name object]
   object])
```

(FullNameObject

```
[LAMBDA (object names)                                    (* smL "19-May-86 19:12")
  (COND
     ((NLISTP names)
      NIL)
     (T (for name in names when (NOT (UIDP name)) do (_ object SetName name])
```

(FillInst

```
[LAMBDA (ivSource obj)                                    ; Edited 15-Jun-93 11:29 by sybalsky:mv:envos
  (for v in ivSource do ;; Insert new values and properties for each variable given
                        (COND
                           ((AND (LISTP v)
                                 (CDR v))
                            (PutIVDescr obj (CAR v)
                                   (create IVDescr
                                           IVValue _ (CAR (CDR v))
                                           IVProps _ (CDR (CDR v])
```

(FillIVs

```
[LAMBDA (self class descrList unmodifiedFlg)              ; Edited 27-May-87 15:33 by smL
```

(* Given an instance, a class and a list of (name . IVdescription)
\, fill) in the instance appropriately. unmodifiedFlg=T in just those cases in which the object is being read in from a file)

```
  (SETQ self (BlankInstance class self unmodifiedFlg))
  [for d in descrList do (PutIVDescr self (CAR d)
                                (OR (CDR d)
                                    (LIST NotSetValue]
  self])
```

(IVSource

```
[LAMBDA (self dropDontSavesFlg)                           ; Edited 15-Aug-90 12:58 by jds
```

(* * Create a list structure form for editing or dumping an instance.)

```
  (for ivName in (_ self ListAttribute 'IVs) bind ivValue ivProps exceptions ivSourceList
     eachtime [SETQ ivProps (WithIVPropDescr self ivName [LAMBDA (self ivName propDescr)
                                                           (AND propDescr (fetch (IVPropDescr IVPropList)
                                                                             of propDescr]
                                                         (LAMBDA (self ivName)
                                                           NIL]
              [SETQ ivValue (WithIVValue self ivName [LAMBDA (self ivName value)
                                                       value]
                                                     (LAMBDA (self ivName)
                                                       NotSetValue]
              [SETQ exceptions (AND dropDontSavesFlg (LET [(val (GetValueOnly self ivName 'DontSave]
                                                           (if (NoValueFound val)
                                                               then NIL
                                                             else val]
     when (NEQ exceptions 'Any) collect [SETQ ivSourceList
                                         (COND
                                            ((OR (NULL dropDontSavesFlg)
                                                 (NULL exceptions))
                                             (CONS ivValue ivProps))
                                            (T (CONS (COND
                                                        ((EQMEMB 'Value exceptions)
                                                                  (* value is to be omitted)
                                                         NotSetValue)
```

```
                                        (T ivValue))
                                    (for pair on ivProps by (CDDR pair)
                                       when (NOT (EQMEMB (CAR pair)
                                                         exceptions))
                                       join (LIST (CAR pair)
                                                  (CADR pair]
                            (if (AND (EQLENGTH ivSourceList 1)
                                     (NotSetValue (CAR ivSourceList)))
                                then (SETQ ivSourceList NIL))
                            (if (AND dropDontSavesFlg (NULL ivSourceList))
                                then (GO $$ITERATE))
                            (CONS ivName ivSourceList])
```

(ModifyInstance

```
[LAMBDA (classValList)                                  (* smL "19-May-86 18:31")
```

(* * Called from DEFINST and used to modify an existing instance after editing, or creating a new instance of the named class on reading in.)

```
  (LET ((className (CAR classValList))
        (names (CADR classValList))
        obj UID nameTable class)                        (* Get the class)
       (SETQ class (GetClassRec className))
       [COND
          ((NULL class)
           (printout T className " has no class defined for it" T "Defining one now:" T)
           (SETQ class (_ ($ Class)
                          New className]                (* Get the UID)
       [COND
          ((UIDP names)
           (SETQ UID names)
           (SETQ names NIL))
          (T (SETQ names (REVERSE names))
             (SETQ UID (pop names]                      (* Get the object)
       (SETQ obj (GetObjFromUID UID))
       (COND
          ((NULL (CDDR classValList))                   (* Just a reference to obj)
           (OR obj (NewObject class UID)))
          (T [COND
                (obj (BlankInstance class obj))
                (T (SETQ obj (NewObject class UID]
             (COND
                (names (NameObject obj names)))
             (FillInst (CDDR classValList)
                       obj)
             obj])
```

(NewClass

```
[LAMBDA (className metaClass)                           ; Edited 15-Aug-90 13:01 by jds
```

(* * Creates a new class of the given name, and returns the class record as the value.
Does not check for old defintion existing.)

```
  (LET (cls)
       (COND
          ((NULL className)
           (ERROR "A Class must be given a name at creation.
                  " NIL T))
          (T (NewEntity [SETQ cls (create class
                                          className _ className
                                          localSupers _ (LIST ($ Tofu))
                                          supers _ (LIST ($ Tofu))
                                          metaClass _ (COND
                                                         ((type? class metaClass)
                                                          metaClass)
                                                         (T (GetClassRec (OR metaClass 'Class]
                                    (COND
                                       ((LISTP className)
                                        (LIST className (Make-UID)))
                                       (T className)))
             (push (fetch (class subClasses) of ($ Tofu))
                   cls)
             (AND (LITATOM className)
                  (MARKASCHANGED className 'CLASSES 'DEFINED))
             cls])
```

(NewObject

```
[LAMBDA (class UID)                                     (* smL " 7-Apr-86 09:50")
```

(* * Create a new empty object for ModifyInstance for DEFINST or DEFINSTANCES.
Set up inames and idescrs from class.)

```
  (COND
     (UID (LET ((obj (GetObjFromUID UID)))
```

```
                (COND
                   (obj (BlankInstance class obj))
                   (T (SETQ obj (BlankInstance class))
                      (CreateEntity obj UID)
                      obj])
```

(UpdateClassIVs

```
  [LAMBDA (classRec)                                          ; Edited 15-Aug-90 13:01 by jds

          (* Called from UpdateSubClassIVs to update the Instance variable lists in this subclass and all its lower subs)

    (/replace (class ivDescrs) of classRec with (for name in (fetch (class localIVs) of classRec)
                                                   collect (FetchCIVDescr classRec name)))

          (* * Make names and descrs in class be just those defined locally, so that UpdateIVDescrs works)

    (/replace (class ivNames) of classRec with (fetch (class localIVs) of classRec))

          (* Now update the descrs from the supers, and then the subs go down to each of the subs.
          Don't mark as changed, since it is only the super that has really changed.)

    (/replace (class supers) of classRec with (ComputeSupersList (fetch (class localSupers) of classRec)))
                                                                  (* make sure you flush the IVIndex cache!)
    (FlushIVIndexCache)
    (UpdateIVDescrs classRec])
```

(UpdateIVDescrs

```
  [LAMBDA (classRec)                                          ; Edited 15-Aug-90 13:01 by jds

          (* * Starts with lvNames=ivNames and descrs set correspondingly.
          Called from UpdateIVDescrs. Fetches all the names of IVs defined for this class directly, or indirectly through supers, and
          creates the appropriate ivNames and ivDescrs)

    (PROG (varNames)

          (* Set varNames to be the list of names as inherited from the supers list of this object)

          [for class bind names first (SETQ varNames (APPEND (fetch (class localIVs) of classRec)))
             in (Supers classRec) do                          (* varNames is a copy of that in classRec)
                                  (for name in [SETQ names (COND
                                                             ((LISTP class)
                                                              (* Take only the local variables. This is a mixin)
                                                              (fetch (class localIVs) of (CAR class)))
                                                             (T (fetch (class ivNames) of class]
                                     do [COND
                                           ((FMEMB name varNames)

          (* remove name from later in list. Order should be as though we had created list starting from the top of the supers hierarchy)

                                                            (SETQ varNames (DREMOVE name varNames]
                                     finally (SETQ varNames (APPEND names varNames]

          (* Now collect the descr for each variable, bringing down the nearest description found by going up the supers list)

          [/replace (class ivDescrs) of classRec
             with (for name in varNames
                     collect (OR (AND (FMEMB name (fetch (class localIVs) of classRec))
                                      (FetchCIVDescr classRec name))
                                 (for class in (Supers classRec)
                                    do (RETURN (OR (AND (FMEMB name (fetch (class localIVs) of class))
                                                        (FetchCIVDescr class name))
                                                   (GO MORE)))
                                 MORE]
          (/replace (class ivNames) of classRec with varNames))      (* Now update subclasses)
    (for sub in (fetch (class subClasses) of classRec) do (UpdateClassIVs sub])

)

(MOVD? 'BootNameObject 'NameObject)

(ADDTOVAR NLAMA DEFINST DEFINSTANCES DEFCLASS DEFCLASSES @ _@)

(DECLARE%: EVAL@COMPILE

[I.S.OPR 'IN-SUPERS-OF NIL '(bind \Supers (\InitialObject _ BODY) declare%: (LOCALVARS \Supers \InitialObject)
                                first (SETQ I.V. (if (type? class \InitialObject)
                                                     then \InitialObject
                                                   else (Class \InitialObject)))
                                    (SETQ \Supers (fetch supers of I.V.))
                                repeatwhile (if \Supers
                                                then (SETQ I.V. (pop \Supers))
                                                     T
                                              else NIL]
)
```

(PUTPROPS **LOOPSSTRUC COPYRIGHT** ("Venue & Xerox Corporation" 1983 1984 1985 1986 1987 1988 1990 1991 1993))

FUNCTION INDEX

MACRO INDEX

VARIABLE INDEX

PROPERTY INDEX

RECORD INDEX

I.S.OPR INDEX