```
File created:   30-Jun-86 16:38:25  {PHYLUM}<LANNING>RPC>LUPINE.;1

  changes to:   (FNS \MakeUnmarshalRecord)

previous date:   1-Aug-85 12:16:27  {PHYLUM}<LISP>KOTO>LISPUSERS>LUPINE.;1

   Read Table:  OLD-INTERLISP-FILE

     Package:   INTERLISP

       Format:   XCCS
```

```
(RPAQQ LUPINECOMS
       [(DECLARE: DONTCOPY (RECORDS ArgSpec Fragment FunctionSpec FieldSpec LupineType RecordLayout TypeSpec))
        (* Basic stub construction)
        (FNS Lupine \ServerComs \MakeUnmarshal \MakeUnmarshal1 \MakeUnmarshalRecord \FixedFetch
             \MakeArgsUnmarshal \ClientComs \MakeMarshal \MakeMarshal1 \MakeMarshalRecord \FixedStore
             \MakeArgsMarshal)
        (* Checking of declarations)
        (FNS \CheckSpec \CheckType \CheckType1 \CheckRecordDecl)
        (* Type table construction)
        (FNS \DeclareTypes \Allocate \AllocateRecord)
        (* Utilities)
        (FNS \TypeName \BaseType \TypeLayout \TypeSize \LupineNotFixed \StaticsFirst \IsStatic)
        (VARS \LupineGetFns \LupinePutFns \LupinePrimativeTypes \LupineInitialTypeTable \LupineNumberTypes
              \LupineNotFixedTypes \LupineDummyTypes \LupineStatics \LupineTypesWithParm)
        (GLOBALVARS \LupineGetFns \LupinePutFns \LupineStatics \LupineTypesWithParm \LupinePrimativeTypes
               \LupineDummyTypes \LupineNotFixedTypes \LupineNumberTypes \LupineInitialTypeTable)
        (DECLARE: DONTCOPY (CONSTANTS (\FirstLupineUserCall 4)
                                  (\FirstLupineSignal 4])

(DECLARE: DONTCOPY

(DECLARE: EVAL@COMPILE

(RECORD ArgSpec (argName argType))

(RECORD Fragment (fixed . notFixed))

(RECORD FunctionSpec (fn . specs))

(RECORD FieldSpec (fieldName fieldType))

(RECORD LupineType (typeName . typeParm))

(RECORD RecordLayout (need . fields))

(RECORD TypeSpec (type typeType definedStubs typeSize . typeBits))
)
)
```

(* * Basic stub construction)

```
(DEFINEQ

(Lupine
  [LAMBDA (packageName functionSpecList signalSpecList typeList lupineTypeString noServer noClient)
                                                                    (* ht: "31-Jul-85 11:47")
    (if (NOT (LITATOM packageName))
        then (ERROR "package name must be an atom" packageName))
    (if (AND functionSpecList (NLISTP functionSpecList))
        then (ERROR "function spec must be a list" functionSpecList))
    (if (AND signalSpecList (NLISTP signalSpecList))
        then (ERROR "signal spec must be a list" signalSpecList))
    (if (AND typeList (NLISTP typeList))
        then (ERROR "type declarations must be a list" typeList))
    (if (NOT lupineTypeString)
        then (printout T "type string defaulted to " [lupineTypeString_ (MKSTRING (PACK (LIST packageName (GDATE
                                                                                                          )
                                                                                               (ETHERHOSTNUMBER]
                   T))
    (if (NOT (STRINGP lupineTypeString))
        then lupineTypeString_ (MKSTRING lupineTypeString))
    (RESETLST
        (RESETSAVE DFNFLG T)
        (LET ((typeTable (\DeclareTypes typeList))
              sName cName)
             (for s in functionSpecList do (\CheckSpec s typeTable))
             (for s in signalSpecList do (\CheckSpec s typeTable T))
             [if (NOT noServer)
                 then (SET sName_ (PACK (LIST (U-CASE packageName)
                                              'SERVERCOMS))
                           (\ServerComs packageName functionSpecList signalSpecList typeTable lupineTypeString))
                      (ADDFILE (PACK (LIST (U-CASE packageName)
                                        'SERVER]
           [if (NOT noClient)
               then (SET cName_ (PACK (LIST (U-CASE packageName)
                                           'CLIENTCOMS))
                         (\ClientComs packageName functionSpecList signalSpecList typeTable lupineTypeString))
                    (ADDFILE (PACK (LIST (U-CASE packageName)
                                        'CLIENT]
           (CONS sName cName)))])


(\ServerComs
  [LAMBDA ($packageName$ functionSpecList signalSpecList typeTable lupineTypeString)
                                                                  (* ht: " 1-Aug-85 12:13")
    (DECLARE (SPECVARS $packageName$))
    (LET ($fns$ dfn iName selTerms $rNames$ catchTerms labelTerms movds hideFn loads)
         (DECLARE (SPECVARS $fns$ $rNames$))

              (* * the NCONC is because some stuff gets pushed onto $fns$ underneath here)

         $fns$_
         (NCONC (bind (i _ \FirstLupineUserCall)
                      argNames nameMap rec rfl fn argSets end result for spec in functionSpecList
                    unless (spec:fn= '*) collect (fn_ (PACK (LIST spec:fn 'ServerStub)))
                                                  [argNames_ (for aSpec in spec:specs as i from 1
                                                                until (if (U-CASE aSpec:argName) = 'RETURNS
                                                                          then result_aSpec)
                                                                collect (CDAR (push nameMap (CONS aSpec:argName
                                                                                                  (PACK* 'l..arg i]
                                                  [argSets_ (for aSpec in (\StaticsFirst spec:specs typeTable)
                                                               collect (LIST 'SETQ (CDR (ASSOC aSpec:argName nameMap))
                                                                                   (\MakeUnmarshal aSpec:argType spec:fn
                                                                                          aSpec:argName typeTable]
                                                  (end_ (if result
                                                            then (\MakeArgsMarshal result:argType fn 'RESULT
                                                                        'l..result typeTable)))
                                                  [APPLY* (FUNCTION DEFINEQ)
                                                          (BQUOTE (, fn (l..cPup l..conv)
                                                                                   (* Lupine generated stub)
                                                                        (DECLARE (SPECVARS l..cPup l..conv))
                                                                        (PROG (l..result ,. argNames)
                                                                          ,.
                                                                          argSets
                                                                              (SETQ l..result
                                                                               (, (fetch fn of spec)
                                                                                  ,. argNames))
                                                                              (\StartReturn l..cPup)
                                                                          ,.
                                                                          end (RETURN l..cPup]
                                                  [selTerms_ (NCONC1 selTerms (BQUOTE (, i (, fn l..pup l..conv]
                                                  (push movds (CONS spec:fn (CONS (PACK* 'Hidden. spec:fn)
                                                                                  fn)))
                                                  (add i 1)
                                                  fn)
                $fns$)
         dfn_
         (PACK (LIST $packageName$ 'ServerDispatch))
         (if signalSpecList
             then (bind (j _ \FirstLupineSignal)
                        cName specs resultSpec margs umres for sSpec in signalSpecList unless (sSpec:fn= '*)
                     do (specs_sSpec:specs)
                        (margs_ (if (U-CASE specs:1:argName) = 'ARGS
                                    then (\MakeArgsMarshal (pop specs)
                                                :argType sSpec:fn 'SIGARGS 'arg typeTable)))
                        (if (U-CASE specs:1:argName) = 'RETURNS
                            then umres_ (\MakeArgsUnmarshal specs:1:argType sSpec:fn 'RESULT typeTable))
                        [push $fns$ (CAR (APPLY* (FUNCTION DEFINEQ)
                                                 (BQUOTE (, [SETQ cName (PACK (LIST 'Catch (fetch fn of sSpec]
                                                            (arg l..conv)
                                                                    (* Lupine generated signal catcher)
                                                            (DECLARE (USEDFREE l..cPup))
                                                            (\StartSignal l..cPup)
                                                            (\AddPupWord l..cPup , j)
                                                            ., margs (SETQ l..cPup (\Call l..cPup NIL l..conv))
                                                            , umres]
                        [catchTerms_ (NCONC catchTerms (APPEND (BQUOTE (, (fetch fn of sSpec)
                                                                          =>
                                                                          (sresume (, cName arg l..conv]
                        (add j 1)))

              (* * the (, (QUOTE enable)) is to keep the form from being prettyprinted)

         [APPLY* (FUNCTION DEFINEQ)
                 (BQUOTE (, dfn (l..pup request l..conv)          (* Lupine generated dispatcher)
                            (, 'enable ., catchTerms (SELECTQ request
                                                              .,
                                                              selTerms
                                                              (SHOULDNT))
```

```
                              ., labelTerms]
          (push $fns$ dfn)
          iName_
          (PACK (LIST $packageName$ 'ServerInterface))
          (if (NOT (BOUNDP iName))
              then (SET iName NIL))
          [push $fns$ (CAR (APPLY* (FUNCTION DEFINEQ)
                                   (BQUOTE (, (PACK (LIST 'Unexport $packageName$))
                                              NIL                    (* Lupine generated interface)
                                              [UnexportInterface (OR , iName (ERROR "not exported"
                                                                               (QUOTE , $packageName$]
                                              (SETQ , iName NIL]
          [push $fns$ (CAR (APPLY* (FUNCTION DEFINEQ)
                                   (BQUOTE (, (PACK (LIST 'Export $packageName$))
                                              (type instance version user password)
                                                                     (* Lupine generated interface)
                                              (if , iName
                                                  then (ERROR "Already exported" (QUOTE , $packageName$)))
                                              (SETQ , iName (ExportInterface user password (OR type ,
                                                                                   lupineTypeString)
                                                                     instance version (FUNCTION , dfn]
          [push $fns$ (CAR (APPLY* (FUNCTION DEFINEQ)
                                   (BQUOTE (, (SETQ hideFn (PACK (LIST 'Hide $packageName$ 'ServerMovds)))
                                              NIL                    (* Lupine generated utility)
                                              (for p in (QUOTE , movds) do (PUTD (CADR p)
                                                                                (GETD (CAR p)))
                                                                          (PUTD (CAR p))
                                                                          (CHANGENAME (CDDR p)
                                                                                (CAR p)
                                                                                (CADR p]
          [LET ((files (NCONC1 (bind wh res for r in $rNames$
                                  when (if wh_ (WHEREIS r 'RECORDS)
                                           else (printout T T "Note - the record " r " is not on any known file" T)
                                                )
                                  do (pushnew res wh:1) finally (RETURN res))
                              'SIGNAL)))
               loads_
               (LIST (BQUOTE (DECLARE: DONTCOPY EVAL@COMPILE EVAL@LOAD (FILES (LOADCOMP)
                                                                              ., files]
          (BQUOTE (                                                  (* Created by Lupine , (GDATE))
                  (FNS ,. $fns$)
                  (VARS (, iName))
                  (GLOBALVARS , iName)
                  ., loads (DECLARE: EVAL@LOAD DONTEVAL@COMPILE DOCOPY (FILES (SYSLOAD)
                                                                               RPC))
                  (P (COND
                        ((EQ 'Y (ASKUSER 15 'N "Hide server fns (must have been already loaded)? "))
                         (, hideFn])
```

(\MakeUnmarshal

```
  [LAMBDA (type fn name typeTable pupName)                           (* ht: " 1-Aug-85 08:57")
    (if type
        then (if (NOT pupName)
                 then pupName_ 'l..cPup)
             (LET (fragment)
                  fragment_
                  (\MakeUnmarshal1 type fn name typeTable 0 16 pupName)
                  (if NOT (fragment:notFixed)
                      then fragment:fixed
                    else (BQUOTE (LET ((l..datum , (fetch fixed of fragment)))
                                      .,
                                      (fetch notFixed of fragment)
                                      l..datum])
```

(\MakeUnmarshal1

```
  [LAMBDA (type fn name typeTable whereAreWe size pupName)           (* ht: " 1-Aug-85 09:09")
    (LET ((typeName (\TypeName type))
          (typeParm (if (LISTP type)
                        then type:typeParm))
          afn typeSpec)
         (SELECTQ typeName
              ((RECORD SEQRECORD)
                   (SHOULDNT))
              (LIST [create Fragment
                            fixed _ (BQUOTE (PROGN (SETQ , pupName (\CheckPupExhausted , pupName 2))
                                                   (for l..i from 1 to (\GetArgDblWord , pupName l..conv)
                                                      collect , (\MakeUnmarshal (CAR typeParm)
                                                                                fn name typeTable pupName])
              (REF                                                   (* no-op except for SEQRECORDs)
                   [LET ((trueType (\BaseType typeParm:1 typeTable))
                         typeSpec)
                        (if (AND (LISTP trueType)
                                 trueType:typeName= 'SEQRECORD)
                            then
```

```
           (* * have to back up one)

                                          (\MakeUnmarshalRecord (OR typeSpec_ (for tte in typeTable thereis
                                                                                                  tte:typeType=trueType
                                                                                )
                                                                   (SHOULDNT))
                                                 fn name typeTable whereAreWe (if size=16
                                                                                  then typeSpec:typeSize
                                                                                else (SHOULDNT))
                                                 pupName)
                                   else (create Fragment
                                               fixed _ (LET [(nilCheck (BQUOTE (\GetArgBool , pupName l..conv]
                                                            [if pupName= 'l..pup
                                                                then nilCheck_ (BQUOTE (PROG1 ,
                                                                                              nilCheck
                                                                                              (SETQ l..pup l..cPup))]
                                                            (BQUOTE (if , nilCheck
                                                                        then NIL
                                                                      else , (\MakeUnmarshal (CAR typeParm)
                                                                                    fn name typeTable pupName])
                     (BITS                                                     (* Just a bit special)
                           (create Fragment
                                  fixed _ (if (ILESSP size 16)
                                               then (\FixedFetch size type whereAreWe fn name typeTable)
                                             elseif whereAreWe~=0
                                               then (HELP "bad layout")
                                             elseif (ILEQ typeParm:1 16)
                                               then

           (* * note that even if we get here with a BITS record of <16 size, we just get a whole word)

                                                    (BQUOTE (\GetArgWord , pupName l..conv))
                                             elseif typeParm:1=32
                                               then                           (* closest we get to LONG CARDINAL)
                                                    (BQUOTE (\GetArgDblWord , pupName l..conv))
                                             else (SHOULDNT))))
                     (if (FMEMB typeName \LupinePrimativeTypes)
                         then (if (ILESSP size 16)
                                  then (create Fragment
                                              fixed _ (\FixedFetch size type whereAreWe fn name typeTable))
                                elseif whereAreWe~=0
                                  then (HELP "bad layout")
                                elseif afn_ (CDR (ASSOC typeName \LupineGetFns))
                                  then

           (* * note that even if we get here with a BITS record of <16 size, we just get a whole word courtesy of the GetFns table entry)

                                        [create Fragment
                                              fixed _ (LET [(sr (BQUOTE (, afn , pupName .,
                                                                             (if typeParm
                                                                                 then (LIST (KWOTE typeParm)))
                                                                             l..conv]
                                                            (if pupName= 'l..cPup
                                                                then sr
                                                              else (BQUOTE (PROG1 ,
                                                                                   sr
                                                                                   (SETQ l..pup l..cPup))]
                                else (SHOULDNT))
                       elseif typeSpec_ (ASSOC typeName typeTable)
                         then (if typeSpec:typeType:type= 'RECORD
                                  then (\MakeUnmarshalRecord typeSpec fn name typeTable whereAreWe
                                              (if (AND size=16 (IGREATERP typeSpec:typeSize 16))
                                                  then typeSpec:typeSize
                                                else size)
                                              pupName)
                                else (\MakeUnmarshal1 typeSpec:typeType fn name typeTable whereAreWe size pupName))
                       else (ERROR "Invalid spec" (LIST fn name type])
```

(\MakeUnmarshalRecord

```
  [LAMBDA (spec fn name typeTable startBit bitWidth pupName)     (* smL "30-Jun-86 16:17")
    (LET
     ((fnName (PACK* (QUOTE Unmarshal)
                     $packageName$
                     (fetch type of spec)
                     (QUOTE #)
                     startBit
                     (QUOTE #)
                     bitWidth))
      (bits (fetch typeBits of spec))
      (seq? (EQ (fetch typeName of (fetch typeType of spec))
                (QUOTE SEQRECORD)))
      fetches createExpr indirects umc notFixed someNot res seqSpec seqEltFetch seqSubSpec seqStatic? leftOver)
     [if (NOT (FMEMB fnName (fetch definedStubs of spec)))
         then
         (push $rNames$ (fetch type of spec))
         (if (ILESSP (fetch typeSize of spec)
                    bitWidth)
              then (SETQ bits (CONS (IPLUS (CAR bits)
                                           (DIFFERENCE bitWidth (fetch typeSize of spec)))
                                    (CDR bits)))
            elseif (AND (IGREATERP (fetch typeSize of spec)
                                   bitWidth)
                        (NOT (ZEROP startBit)))
              then (HELP "bad layout"))
          [SETQ fetches
           (bind (whereAreWe _ startBit)
                 (nFixed _ 0)
                 last for field in (fetch typeParm of (fetch typeType of spec)) as size in bits
              join (if (AND (LISTP (fetch fieldType of field))
                            (EQ (fetch typeName of (fetch fieldType of field))
                                (QUOTE SEQUENCE)))
                       then (if seq?
                                then (SETQ seqSpec field)
                                     [SETQ seqSubSpec (CAR (fetch typeParm of (fetch fieldType of field]
```

(* * * This should be a call to \MakeUnmarshal1, but because of a CEDAR Lupine bug, is not
(see below for more discussion) -
If CEDAR is ever fixed, the call should be as follows: (SETQ seqEltFetch
(\MakeUnmarshal1 seqSubSpec fn name typeTable whereAreWe size
(QUOTE l..pup))))

```
                                     [SETQ seqEltFetch (\MakeUnmarshal seqSubSpec fn name typeTable
                                                              (if (SETQ seqStatic? (\IsStatic seqSubSpec typeTable))
                                                                  then (QUOTE l..pup)
                                                                else (QUOTE l..cPup]
                              else (SHOULDNT))
                           NIL
                     elseif (\LupineNotFixed field typeTable)
                       then (add nFixed 1)
                            (SETQ someNot T)
                            NIL
                     else (SETQ umc (\MakeUnmarshal1 (fetch fieldType of field)
                                           fn name typeTable whereAreWe size (QUOTE l..pup)))
                          [if (fetch notFixed of umc)
                              then (SETQ notFixed (NCONC1 notFixed
                                                          (BQUOTE (LET ((l..datum (fetch (, (fetch type of spec)
                                                                                             ,
                                                                                             (fetch fieldName
                                                                                                of field))
                                                                                      of l..datum)))
                                                                       .,
                                                                       (fetch notFixed of umc]
                          (SETQ umc (fetch fixed of umc))
                          (SETQ whereAreWe (LOGAND 15 (IPLUS whereAreWe size)))
                          (if (NOT (ZEROP nFixed))
                              then (SETQ umc (BQUOTE (PROGN (SETQ l..pup (\SkipWordsIn l..pup , (LLSH nFixed 1)))
                                                            , umc)))
                                   (SETQ nFixed 0))
                          (SETQ last (LIST (fetch fieldName of field)
                                           (QUOTE _)
                                           umc)))
              finally (if (NOT (ZEROP nFixed))
                          then (SETQ leftOver (BQUOTE (SETQ l..pup (\SkipWordsIn l..pup , (LLSH nFixed 1]
          (SETQ createExpr (BQUOTE (create , (fetch type of spec)
                                           ., fetches)))
          [if leftOver
              then (SETQ createExpr (BQUOTE (PROG1 ,
                                                   createExpr
                                                   ,
                                                   leftOver)]
          (if someNot
              then [SETQ indirects (for field in (fetch typeParm of (fetch typeType of spec))
                                      when (\LupineNotFixed field typeTable)
                                      collect (SETQ umc (\MakeUnmarshal (fetch fieldType of field)
                                                               fn name typeTable (QUOTE l..cPup)))
                                            (BQUOTE (replace (, (fetch type of spec)
                                                                ,
                                                                (fetch fieldName of field))
                                                       of l..datum with , umc]
                   (SETQ notFixed (NCONC notFixed indirects)))
          (SETQ createExpr
           (if seq?
               then [LET (prelim term f nf)
```

(* * the code in this comment is the way this **should** work, with only the non-statics following after, but in fact as CEDAR
Lupine stands if a sequence's element type has any non-static parts, the WHOLE element gets repeated.
There is a further patch associated with this higher up in this function.
If this ever gets fixed, replace the if statement which follows this comment with this code:
(if (fetch notFixed of seqEltFetch) then (SETQ f (fetch fixed of seqEltFetch))
(SETQ nf (fetch notFixed of seqEltFetch)) elseif (\LupineNotFixed
(fetch fieldType of seqSpec) typeTable) then (SETQ prelim (BQUOTE
(first (\SkipWordsIn l..pup (LLSH size 1))))) (SETQ nf (fetch fixed of seqEltFetch)) else
(SETQ f (fetch fixed of seqEltFetch))))

```
                        (if seqStatic?
                            then (SETQ f seqEltFetch)
                          else [SETQ prelim (BQUOTE (first (\SkipWordsIn l..pup (ITIMES , (LRSH (\TypeSize
                                                                                                     seqSubSpec
                                                                                                     typeTable)
                                                                                              4)
                                                                                       l..size]
                              (SETQ nf seqEltFetch))
                        [if nf
                            then [SETQ term (if f
                                                 then (BQUOTE (LET ((l..datum (CAR l..p)))
                                                                   ., nf))
                                               else (BQUOTE (RPLACA l..p , nf]
                                 (SETQ notFixed (NCONC1 notFixed
                                                        (BQUOTE (for l..p
                                                                   on (fetch (, (fetch type of spec)
                                                                                ,
                                                                                (fetch fieldName of seqSpec))
                                                                         of l..datum)
                                                                   do , term]
                        (BQUOTE ((SETQ l..pup (\CheckPupExhausted l..pup 3))
                                 (if (\GetArgBool l..pup l..conv)
                                     then NIL
                                   else (LET ((l..size (\GetArgDblWord l..pup l..conv)))
                                             (bind (l..result _ , createExpr)
                                                   ., prelim for l..i from 1 to l..size collect , f
                                                finally (replace (, (fetch type of spec)
                                                                     ,
                                                                     (fetch fieldName of seqSpec))
                                                           of l..result with $$VAL)
                                                        (RETURN l..result]
            else (LIST createExpr)))
        [if (AND seq? notFixed)
            then (SETQ notFixed (LIST (BQUOTE (if l..datum
                                                  then ., notFixed]
        (APPLY* (FUNCTION DEFINEQ)
               (BQUOTE (, fnName (l..pup l..conv)                (* Lupine generated stub)
                          ., createExpr)))

         (* * must record the notFixed generated here so they get used if the function gets re-used)

        (PUT fnName (QUOTE LupineNotFixed)
             notFixed)
        (push $fns$ fnName)
        (push (fetch definedStubs of spec)
              fnName)
      else (SETQ notFixed (APPEND (GETP fnName (QUOTE LupineNotFixed]
    (SETQ res (BQUOTE (, fnName , pupName l..conv)))
    [if pupName= (QUOTE l..pup)
        then

         (* * horrible kludge as our function may have reset l..cPup but our caller won't see that)

             (SETQ res (LIST (QUOTE PROG1)
                             res
                             (QUOTE (SETQ l..pup l..cPup]
    (create Fragment
           fixed _ res
           notFixed _ notFixed])
```

```
(\FixedFetch
  [LAMBDA (size type whereAreWe fn name typeTable)             (* ht: " 1-Aug-85 08:54")
    (LET ((typeName (\TypeName type))
          (typeParm (if (LISTP type)
                        then type:typeParm))
          ff bitNum form)
         (SELECTQ typeName
             (RECORD (SHOULDNT))
             ((BOOLEAN BITS ENUMERATION)

         (* * compute the field descriptor)

                  bitNum_ (IPLUS (LLSH whereAreWe 4)
                                 size-1)

         (* * make the call to FETCHFIELD)

                        ff_
                        (BQUOTE (FETCHFIELD ' (NIL 0 (BITS ., bitNum))
                                           (\CurrentPupBase l..pup)))
                        form_
                        (SELECTQ typeName
                            (BITS ff)
                            (ENUMERATION [BQUOTE (CAR (NTH (QUOTE , typeParm)
                                                           (ADD1 , ff])
```

```
                                (BOOLEAN (BQUOTE (NOT (ZEROP , ff))))
                                (SHOULDNT))
                            (if (ZEROP whereAreWe)
                                then

             (* * need to check there is room in the l..pup)

                                     (BQUOTE (PROGN (SETQ l..pup (\CheckPupExhausted l..pup 1))
                                                    , form))
                              elseif (IPLUS whereAreWe size)
                                     =16
                                then

             (* * must advance the counter)

                                     (BQUOTE (PROG1 ,
                                                    form
                                                    (\IncrDataOffset l..pup 1)))
                              else form))
                  (SHOULDNT])
```

(\MakeArgsUnmarshal

```
  [LAMBDA (spec fn multTypeName typeTable)                       (* ht: " 1-Aug-85 08:51")
    (LET (resultSpec)
         (if (AND (LITATOM spec)
                  resultSpec_
                  (\BaseType spec typeTable)
                  (LISTP resultSpec)
                  resultSpec:typeName=multTypeName)
             then                                                (* create the record)
                  (push $rNames$ spec)
                  [NCONC (LIST 'create spec)
                         (for rSpec in (\StaticsFirst resultSpec:typeParm typeTable)
                            join (BQUOTE (, (fetch fieldName of rSpec)
                                            _ , (\MakeUnmarshal (fetch fieldType of rSpec)
                                                                fn
                                                                (fetch fieldName of rSpec)
                                                                typeTable]
           else (\MakeUnmarshal spec fn 'l..result typeTable])
```

(\ClientComs

```
  [LAMBDA ($packageName$ functionSpecList signalSpecList typeTable lupineTypeString)
                                                                 (* ht: " 1-Aug-85 09:39")
    (DECLARE (SPECVARS $packageName$))
    (LET ($fns$ dfn selTerms iName cName typeSels $rNames$ sDisp sigTerms movds movdFn result loads)
         (DECLARE (SPECVARS $fns$ $rNames$))
         iName_
         (PACK (LIST $packageName$ 'ClientInterface))
         sDisp_
         (PACK (LIST 'Dispatch $packageName$ 'Signals))

             (* * the NCONC is because some stuff gets pushed onto $fns$ underneath here)

         $fns$_
         (NCONC (bind (i _ \FirstLupineUserCall)
                      argNames rec rfl fn argPuts end stubFn for spec in functionSpecList unless spec:fn= '*
                   collect (fn_spec:fn)
                           (stubFn_ (PACK (LIST 'RPCClientStub. fn)))
                           (argPuts_ (for aSpec in (\StaticsFirst spec:specs typeTable)
                                        join (\MakeMarshal aSpec:argType aSpec:argName fn aSpec:argName typeTable)))
                           (argNames_ (NCONC (for aSpec in spec:specs until (if (U-CASE aSpec:argName) = 'RETURNS
                                                                                  then result_aSpec)
                                                collect aSpec:argName)
                                             (LIST 'l..interfaceArg 'l..conv)))
                           (end_ (if result
                                     then (\MakeArgsUnmarshal result:argType fn 'RESULT typeTable)))
                           [APPLY* (FUNCTION DEFINEQ)
                                   (BQUOTE (, stubFn , argNames     (* Lupine generated stub)
                                                 (PROG [l..cPup (l..interface (OR l..interfaceArg (CAR , iName]
                                                       (DECLARE (SPECVARS l..cPup))
                                                       (SETQ l..cPup (\StartCall (CAR l..interface)
                                                                                 (CDR l..interface)
                                                                                 l..conv))
                                                       (\AddPupWord l..cPup , i l..conv)
                                                   ,.
                                                   argPuts
                                                       (SETQ l..cPup (\Call l..cPup (FUNCTION , sDisp)
                                                                                 l..conv))
                                                       (RETURN (PROG1 ,
                                                                      end
                                                                      (\RELEASE.PUP l..cPup))]
                           (push movds (CONS stubFn fn))
                           (add i 1)
                           stubFn)
                $fns$)
```

```
          sigTerms_
          (bind (j _ \FirstLupineSignal)
               mres umargs specs for sSpec in signalSpecList unless sSpec:fn= '*
             collect (specs_sSpec:specs)
                   (umargs_ (if (U-CASE specs:1:argName) = 'ARGS
                            then (\MakeArgsUnmarshal (pop specs)
                                         :argType sSpec:fn 'SIGARGS typeTable)))
                   (if (U-CASE specs:1:argName) = 'RETURNS
                       then mres_ (\MakeArgsMarshal specs:1:argType sSpec:fn 'RESULT 'l..result typeTable))
                   (PROG1 (BQUOTE (, j (PROG (l..result)
                                             (SETQ l..result (Signal (QUOTE , (fetch fn of sSpec))
                                                                       , umargs))
                                             (\StartReturn l..cPup)
                                         .,
                                         mres)))
                            (add j 1)))
          (APPLY* (FUNCTION DEFINEQ)
                 (BQUOTE (, sDisp (l..cPup l..conv)              (* Lupine generated dispatcher)
                             (DECLARE (SPECVARS l..cPup l..conv))
                             (SELECTQ (\GetArgWord l..cPup l..conv)
                                  .,
                                  sigTerms
                                  (SHOULDNT))
                             l..cPup)))
          (push $fns$ sDisp)
          (if (NOT (BOUNDP iName))
              then (SET iName NIL))
          [push $fns$ (CAR (APPLY* (FUNCTION DEFINEQ)
                                   (BQUOTE (, (PACK (LIST 'Unimport $packageName$))
                                             (l..interface)       (* Lupine generated interface)
                                             (if l..interface
                                                 then (if (FMEMB l..interface , iName)
                                                          then (UnimportInterface l..interface)
                                                               (SETQ , iName (DREMOVE l..interface , iName))
                                                        else (ERROR "not imported" l..interface))
                                               else (for e in , iName do (UnimportInterface e))
                                                    (SETQ , iName NIL]
          [push $fns$ (CAR (APPLY* (FUNCTION DEFINEQ)
                                   (BQUOTE (, (PACK (LIST 'Import $packageName$))
                                             (type instance version)
                                                                  (* Lupine generated interface)
                                             (CAR (push , iName (ImportInterface (OR type , lupineTypeString)
                                                                         instance version]
          [push $fns$ (CAR (APPLY* (FUNCTION DEFINEQ)
                                   (BQUOTE (, (SETQ movdFn (PACK (LIST 'MovdsFor $packageName$)))
                                             NIL                  (* Lupine generated utility)
                                             (for p in (QUOTE , movds) do (PUTD (CDR p)
                                                                                 (GETD (CAR p]
          [if $rNames$
              then (LET [(files (bind wh res for r in $rNames$
                                   when (if wh_ (WHEREIS r 'RECORDS)
                                            else (printout T T "Note - the record " r " is not on any known file" T)
                                            )
                                   do (pushnew res wh:1) finally (RETURN res]
                      loads_
                      (LIST (BQUOTE (DECLARE: DONTCOPY EVAL@COMPILE EVAL@LOAD (FILES (LOADCOMP)
                                                                                ., files]
          (BQUOTE (                                               (* Created by Lupine , (GDATE))
                   (FNS ,. $fns$)
                   (VARS (, iName))
                   (GLOBALVARS , iName)
                   (P (, movdFn))
                   ., loads (DECLARE: EVAL@LOAD DONTEVAL@COMPILE DOCOPY (FILES (SYSLOAD)
                                                                               RPC])
```

(\MakeMarshal

```
  [LAMBDA (type val fn name typeTable pupName)                   (* ht: " 1-Aug-85 08:57")
    (if type
        then (if (NOT pupName)
                 then pupName_ 'l..cPup)
             (LET (fragment)
                  fragment_
                  (\MakeMarshal1 type val fn name typeTable 0 16 pupName)
                  (if NOT (fragment:notFixed)
                      then fragment:fixed
                    else (NCONC fragment:fixed (if val~= 'l..datum
                                                   then [LIST (BQUOTE (LET ((l..datum , val))
                                                                           .,
                                                                           (fetch notFixed of fragment]
                                                 else fragment:notFixed])
```

(\MakeMarshal1

```
  [LAMBDA (type val fn name typeTable whereAreWe size pupName)   (* ht: " 1-Aug-85 08:59")
    (LET ((typeName (\TypeName type))
          (typeParm (if (LISTP type)
                    then type:typeParm))
          afn typeSpec)
         (SELECTQ typeName
             ((RECORD SEQRECORD)
                  (SHOULDNT))
             (LIST [create Fragment
                           fixed _ (LIST (BQUOTE (PROGN (\CheckPupOverflow , pupName 4)
                                                          (\AddPupDblWord , pupName (LENGTH , val)
                                                                 l..conv)
                                                          (for l..v in , val
                                                             do ., (\MakeMarshal (CAR typeParm)
                                                                          'l..v fn name typeTable pupName])
             (REF                                                  (* no-op except for SEQRECORDs)
                  [LET ((trueType (\BaseType typeParm:1 typeTable))
                        typeSpec)
                       (if (AND (LISTP trueType)
                                trueType:typeName= 'SEQRECORD)
                           then

          (* * have to back up one)

                                (\MakeMarshalRecord (OR typeSpec_ (for tte in typeTable thereis
                                                                                          tte:typeType=trueType
                                                                                   )
                                                                 (SHOULDNT))
                                       val fn name typeTable whereAreWe (if size=16
                                                                            then typeSpec:typeSize
                                                                          else (SHOULDNT))
                                       pupName)
                         else (create Fragment
                                     fixed _ (LIST (BQUOTE (if , val
                                                                then (\AddPupBoolean , pupName NIL l..conv)
                                                                     .,
                                                                     (\MakeMarshal (CAR typeParm)
                                                                            val fn name typeTable pupName)
                                                              else (\AddPupBoolean , pupName T l..conv])
             (BITS (create Fragment
                           fixed _ (if (ILESSP size 16)
                                       then (\FixedStore size type val whereAreWe fn name typeTable)
                                     elseif whereAreWe~=0
                                       then (HELP "bad layout")
                                     elseif (ILEQ typeParm:1 16)
                                       then

          (* * note that even if we get here with a BITS record of <16 size, we just put a whole word)

                                            (LIST (BQUOTE (\AddPupWord , pupName , val l..conv)))
                                     elseif typeParm:1=32
                                       then                     (* closest we get to LONG CARDINAL)
                                            (LIST (BQUOTE (\AddPupDblWord , pupName , val l..conv)))
                                     else (SHOULDNT))))
             (if (FMEMB typeName \LupinePrimativeTypes)
                 then (if (ILESSP size 16)
                          then (create Fragment
                                      fixed _ (\FixedStore size type val whereAreWe fn name typeTable))
                        elseif whereAreWe~=0
                          then (HELP "bad layout")
                        elseif afn_ (CDR (ASSOC typeName \LupinePutFns))
                          then

          (* * note that if we get here with a BITS record of <16 size, we just get a whole word courtesy of the PutFns table entry)

                               [create Fragment
                                      fixed _ (LIST (BQUOTE (, afn , pupName ., (if typeParm
                                                                                    then (LIST (KWOTE typeParm)))
                                                             , val l..conv]
                        else (SHOULDNT))
               elseif typeSpec_ (ASSOC typeName typeTable)
                 then (if typeSpec:typeType:type= 'RECORD
                          then (\MakeMarshalRecord typeSpec val fn name typeTable whereAreWe
                                      (if (AND size=16 (IGREATERP typeSpec:typeSize 16))
                                          then typeSpec:typeSize
                                        else size)
                                      pupName)
                        else (\MakeMarshal1 typeSpec:typeType val fn name typeTable whereAreWe size pupName))
               else (ERROR "Invalid spec" (LIST fn name type])
```

(\MakeMarshalRecord

```
  [LAMBDA (spec val fn name typeTable startBit bitWidth pupName) (* ht: " 1-Aug-85 12:16")
    (LET
     ((fnName (PACK* 'Marshal $packageName$ spec:type '# startBit '# bitWidth))
      (bits spec:typeBits)
      (seq? spec:typeType:typeName= 'SEQRECORD)
      seqSpec seqSubSpec seqEltStore seqStatic? stores notFixed indirects mc someNot)
     (if (NOT (FMEMB fnName spec:definedStubs))
         then
```

```
          (push $rNames$ spec:type)
          (if (ILESSP spec:typeSize bitWidth)
              then bits_ (CONS (IPLUS bits:1 bitWidth-spec:typeSize)
                               bits::1)
            elseif (AND (IGREATERP spec:typeSize bitWidth)
                        (NOT (ZEROP startBit)))
              then (HELP "bad layout"))
          [stores_
           (bind (whereAreWe _ startBit)
                 (nFixed _ 0) for field in spec:typeType:typeParm as size in bits
              join (if (AND (LISTP field:fieldType)
                            field:fieldType:typeName= 'SEQUENCE)
                       then (if seq?
                                then (seqSpec_field)
                                     (seqSubSpec_field:fieldType:typeParm:1)

           (* * * This should be a call to \MakeMarshal1 but because of a CEDAR Lupine bug, is not
           (see below for more discussion) -
           If CEDAR is ever fixed, the call should be as follows: (SETQ seqEltStore
           (\MakeMarshal1 seqSubSpec (QUOTE l..datum) fn name typeTable whereAreWe size
           (QUOTE l..pup))))

                                     (seqEltStore_ (\MakeMarshal seqSubSpec 'l..datum fn name typeTable
                                                          (if seqStatic?_ (\IsStatic seqSubSpec typeTable)
                                                              then 'l..pup
                                                            else 'l..cPup)))
                              else (SHOULDNT))
                            NIL
                     elseif (\LupineNotFixed field typeTable)
                       then (add nFixed 1)
                            (someNot_T)
                            NIL
                     else (mc_ (\MakeMarshal1 field:fieldType (BQUOTE (fetch (, (fetch type of spec)
                                                                                 ,
                                                                                 (fetch fieldName of field))
                                                                          of l..datum))
                                         fn name typeTable whereAreWe size 'l..pup))
                          [if mc:notFixed
                              then notFixed_ (NCONC1 notFixed
                                                     (BQUOTE (LET ((l..datum (fetch (, (fetch type of spec)
                                                                                         ,
                                                                                         (fetch fieldName of field))
                                                                                  of l..datum)))
                                                                  .,
                                                                  (fetch notFixed of mc]
                          (mc_mc:fixed)
                          (whereAreWe_ (LOGAND 15 (IPLUS whereAreWe size)))
                          (if (NOT (ZEROP nFixed))
                              then (PROG1 (CONS (BQUOTE (\SkipBytesOut l..pup , (LLSH nFixed 2)))
                                                mc)
                                          nFixed_0)
                            else mc))
              finally (if (NOT (ZEROP nFixed))
                          then $$VAL_ (NCONC1 $$VAL (BQUOTE (\SkipBytesOut l..pup , (LLSH nFixed 2]
          (if someNot
              then (indirects_ (for field in spec:typeType:typeParm when (\LupineNotFixed field typeTable)
                                  join (\MakeMarshal field:fieldType (BQUOTE (fetch (, (fetch type of spec)
                                                                                        ,
                                                                                        (fetch fieldName of field))
                                                                                 of l..datum))
                                              fn name typeTable 'l..cPup)))
                   (notFixed_ (NCONC notFixed indirects)))
          [if seq?
              then (LET (code f nf)

           (* * the code in this comment is the way this **should** work, with only the non-statics following after, but in fact as CEDAR
           Lupine stands if a sequence's element type has any non-static parts, the WHOLE element gets repeated.
           There is a further patch associated with this higher up in this function.
           If this ever gets fixed, replace the if statement which follows this comment with this code:
           (if (fetch notFixed of seqEltStore) then (SETQ f (fetch fixed of seqEltStore))
           (SETQ nf (fetch notFixed of seqEltStore)) elseif (\LupineNotFixed
           (fetch fieldType of seqSpec) typeTable) then (SETQ code (LIST
           (QUOTE (\SkipBytesOut l..pup (LLSH (LENGTH l..sequence) 2)))))
           (SETQ nf (fetch fixed of seqEltStore)) else (SETQ f (fetch fixed of seqEltStore))))

                        (if seqStatic?
                            then f_seqEltStore
                          else [code_ (LIST (BQUOTE (\SkipBytesOut l..pup (ITIMES , (LRSH (\TypeSize seqSubSpec
                                                                                                    typeTable)
                                                                                          3)
                                                                                   (LENGTH l..sequence]
                               (nf_seqEltStore))
                        [if f
                            then code_ (LIST (BQUOTE (for l..datum in l..sequence do ., f]
                        [if nf
                            then notFixed_ (NCONC1 notFixed
                                                   (BQUOTE (for l..datum
                                                                 in (fetch (, (fetch type of spec)
                                                                              ,
                                                                              (fetch fieldName of seqSpec))
                                                                       of l..datum)
                                                                 do ., nf]
                    stores_
                    (BQUOTE ((\CheckPupOverflow l..pup 6)
                             (if l..datum
                                 then (LET ((l..sequence (fetch (, (fetch type of spec)
                                                                    ,
                                                                    (fetch fieldName of seqSpec))
                                                             of l..datum)))
                                           (\AddPupBoolean l..pup NIL l..conv)
                                           (\AddPupDblWord l..pup (LENGTH l..sequence)
                                                  l..conv)
                                           ., stores ., code)
                               else (\AddPupBoolean l..pup T l..conv]
          [if (AND seq? notFixed)
              then notFixed_ (LIST (BQUOTE (if l..datum
                                                 then ., notFixed]
          (APPLY* (FUNCTION DEFINEQ)
                 (BQUOTE (, fnName (l..pup l..datum l..conv)        (* Lupine generated stub)
                             ., stores)))

           (* * must record the notFixed generated here so they get used if the function gets re-used)

          (PUT fnName 'LupineNotFixed notFixed)
          (push $fns$ fnName)
          (push spec:definedStubs fnName)
        else notFixed_ (APPEND (GETP fnName 'LupineNotFixed)))
      (create Fragment
             fixed _ (LIST (BQUOTE (, fnName , pupName , val l..conv)))
             notFixed _ notFixed])
```

(\FixedStore

```
  [LAMBDA (size type val whereAreWe fn name typeTable)            (* ht: " 1-Aug-85 09:10")
    (LET ((typeName (\TypeName type))
          (typeParm (if (LISTP type)
                        then type:typeParm))
          rf bitNum form)
         (SELECTQ typeName
             (RECORD (SHOULDNT))
             ((BOOLEAN BITS ENUMERATION)

          (* * compute the field descriptor)

                  bitNum_ (IPLUS (LLSH whereAreWe 4)
                                 size-1)

          (* * make the call to FETCHFIELD)

                  form_
                  (SELECTQ typeName
                      (BITS val)
                      (ENUMERATION [BQUOTE (for l..i from 0 as l..t in (QUOTE , typeParm)
                                              do (if (EQ l..t , val)
                                                     then (RETURN l..i))
                                              finally (Signal 'BoundsCheck (CONS , val (QUOTE , typeParm])
                      (BOOLEAN (BQUOTE (if , val
                                           then 1
                                         else 0)))
                      (SHOULDNT))
                  rf_
                  (BQUOTE (REPLACEFIELD ' (NIL 0 (BITS ., bitNum))
                                 (\CurrentPupPosition l..pup)
                                 , form))
                  (if (ZEROP whereAreWe)
                      then

          (* * need to check there is room in the l..pup)

                           (BQUOTE ((\CheckPupOverflow l..pup 2)
                                    , rf))
                    elseif (IPLUS whereAreWe size)
                           =16
                      then

          (* * must advance the counter -
          must use LIST here because BQUOTE causes NCONC problems)

                           (LIST rf ' (\IncrPupLength l..pup 2))
                    else (LIST rf)))
             (SHOULDNT])
```

(\MakeArgsMarshal

```
  [LAMBDA (spec fn multTypeName varName typeTable)               (* ht: "31-Jul-85 11:44")
    (LET (resultSpec)
         (if (AND (LITATOM spec)
                  resultSpec_
                  (\BaseType spec typeTable)
                  (LISTP resultSpec)
                  resultSpec:typeName=multTypeName)
             then                                                (* unpack a record)
                  (push $rNames$ spec)
                  (for rBit in (\StaticsFirst resultSpec:typeParm typeTable)
                     join (\MakeMarshal rBit:fieldType (BQUOTE (fetch (, spec , (fetch fieldName of rBit))
                                                                of , varName))
                                 fn rBit:fieldName typeTable))
           else (\MakeMarshal spec varName fn varName typeTable])
)
```

(* * Checking of declarations)

```
(DEFINEQ

(\CheckSpec
  [LAMBDA (spec typeTable sigFlg)                                (* ht: "31-Jul-85 11:35")
    (if (NLISTP spec)
        then (ERROR "each spec must be a list" spec))
    (if spec:fn~= '*
        then (if (NOT (LITATOM spec:fn))
                 then (ERROR "the fn/signal of a spec must be an atom" spec:fn))
             (if (NLISTP spec:specs)
                 then (if spec:specs
                          then (ERROR "the arg specs of a spec must be a list" spec:specs)))
             (if (OR spec:specs=NIL (U-CASE spec:specs:1:argName)
                     = 'RETURNS)
                 then (printout T "Note: " spec:fn " has no args" T))
             (bind aSpec argsAlready [an _ (if (AND (NOT sigFlg)
                                                    (GETD (fetch fn of spec)))
                                               then (ARGLIST (fetch fn of spec]
                for specP on spec:specs
                do (if (U-CASE specP:1:argName) = 'RETURNS
                       then (if specP::1
                                then (ERROR "RETURNS must be the last spec" specP)
                              else (GO $$OUT))
                     else aSpec_specP:1)
                   (if sigFlg
                       then (if (AND (U-CASE specP:1:argName)
                                     = 'ARGS (NOT argsAlready))
                                then argsAlready_T
                              else (ERROR "first and only arg spec for a signal must be called ARGS"))
                            (\CheckType aSpec typeTable NIL T)
                     else (if an
                              then [if aSpec:argName=an:1
                                       then (pop an)
                                     else (ERROR "arg name not right" (CONS aSpec:argName (pop an]
                            else (if (NOT (LITATOM aSpec:argName))
                                     then (ERROR "arg name must be litatom" aSpec))
                                 (if (GETD spec:fn)
                                     then (printout T "Note: spec has more arguments than function" , spec:fn ,
                                                 aSpec T)))
                          (\CheckType aSpec typeTable))
                finally (if an
                            then (printout T "Note: spec has fewer arguments than function" , spec:fn T)))
             (LET ((last spec:-1))
                  (if (U-CASE last:argName) = 'RETURNS
                      then (\CheckType last typeTable NIL T)
                    else (printout T "Note: " spec:fn " has no result" T])

(\CheckType
  [LAMBDA (spec typeTable inDecl inSpecial inRef inSeq)          (* ht: "30-Jul-85 09:01")
    (\CheckType1 spec (\TypeName spec:argType)
           (if (LISTP spec:argType)
               then spec:argType:typeParm)
           typeTable inDecl inSpecial inRef inSeq])

(\CheckType1
  [LAMBDA (spec typeName typeParm typeTable inDecl inSpecial inRef inSeq)
                                                                 (* ht: "31-Jul-85 15:57")
    (LET
     (trueType)
     (if (FMEMB typeName \LupinePrimativeTypes)
         then
         (if (FMEMB typeName \LupineTypesWithParm)
             then (if (NOT typeParm)
                      then (ERROR "Must have type parm for type" spec))
                  (SELECTQ typeName
                      ((LIST REF SEQUENCE)
                          (if (AND typeName= 'SEQUENCE (NOT inSeq))
                              then (ERROR "SEQUENCE field can occur only in SEQRECORDs" spec))
                          (\CheckType spec:argType typeTable NIL NIL typeName= 'REF))
                      (BITS (if (NOT (AND (NUMBERP typeParm:1)
                                          (IGREATERP typeParm:1 0)
                                          (OR (ILEQ typeParm:1 16)
                                              typeParm:1=32)
                                          typeParm::1=NIL))
                                then (ERROR "BITS type must have exactly one  numeric parameter in [1..16] U
                                            [32]" spec)))
                      ((RECORD RESULT SIGARGS)
                          (if (NOT inDecl)
                              then (ERROR "In line RECORDs/RESULTs/SIGARGSs not allowed - must be pre-declared
                                          as a named type" spec))
                          (\CheckRecordDecl spec typeParm)
                          (for fs in typeParm do (\CheckType fs typeTable)))
                      (SEQRECORD (if (NOT inDecl)
                                     then (ERROR "In line SEQRECORDs not allowed - must be pre-declared as a
                                                 named type" spec))
                                 (\CheckRecordDecl spec typeParm)
                                 (if [NOT (for fieldSpecPointer on typeParm
                                               thereis (PROG1 (if (U-CASE (\TypeName fieldSpecPointer:1:fieldType)) =
                                                                  'SEQUENCE
                                                                  then (if fieldSpecPointer::1
                                                                           then (ERROR "SEQUENCE must be the last
                                                                                       field of a SEQRECORD" spec)
                                                                         else T))
                                                              (\CheckType fieldSpecPointer:1 typeTable NIL NIL NIL T]
                                     then (ERROR "SEQRECORD must end with a SEQUENCE field" spec))
                                 (if (NOT (FMEMB (\TypeName typeParm:-2:fieldType)
                                                 \LupineNumberTypes))
                                     then (printout T "Warning - next to last field in SEQRECORD not a numeric
                                                 type?" , spec)))
                      NIL)
          elseif typeParm
            then (ERROR "Shouldnt have type parm for type" spec))
      elseif (ASSOC typeName typeTable)
        then (if typeParm
                 then (ERROR "Shouldnt have type parm for user-defined type" spec))
             (trueType_ (\BaseType typeName typeTable))
             (if (LISTP trueType)
                 then (if (AND (FMEMB trueType:typeName \LupineDummyTypes)
                               (NOT inSpecial))
                          then (ERROR "Can't use RESULT/SIGARGS except from RETURNS/ARGS spec" spec)
                        elseif (AND trueType:typeName= 'SEQRECORD (NOT inRef))
                          then (ERROR "Must get to SEQRECORD via a REF, not directly" spec)))
      else (ERROR "Not a type" spec])
```

(\CheckRecordDecl

```
  [LAMBDA (spec fieldSpecs)                                     (* ht: "30-Jul-85 09:41")
    (LET ((recFields (RECORDFIELDNAMES spec:fieldName)))
         (if (NOT recFields)
             then (ERROR "No record declaration for record type" spec))
         (if [NOT (AND (LENGTH recFields)
                       =
                       (LENGTH fieldSpecs)
                       (for fieldSpec in fieldSpecs always (FMEMB fieldSpec:fieldName recFields]
             then (ERROR "Field names in type declaration don't match up  with those of record" (LIST recFields
                                                                                                         spec])
)
```

(* * Type table construction)

```
(DEFINEQ
```

(\DeclareTypes

```
  [LAMBDA (typeDecls)                                           (* ht: "31-Jul-85 14:50")
    (bind newEntry (typeTable _ (APPEND \LupineInitialTypeTable))
         allocation for ty in typeDecls unless ty:type= '*
       do (if (NOT (AND ty:type (LITATOM ty:type)))
                then (ERROR "type declaration must begin with an atomic type name" ty))
          (if (OR NOT (ty:typeType)
                  ty::2)
                then (ERROR "there must be one and only one type in a type declaration" ty))
          (if (U-CASE ty:type) = 'INCLUDE
                then (if (AND ty:typeType (LITATOM ty:typeType))
                         then typeTable_ (NCONC typeTable (\DeclareTypes (EVALV ty:typeType)))
                       else (ERROR "INCLUDE must be of the form (INCLUDE <litatom>)" ty))
            else (\CheckType ty typeTable T)
                 (newEntry_ (create TypeSpec
                                    type _ ty:type
                                    typeType _ ty:typeType))
                 (if (NOT (AND (LISTP ty:typeType)
                            (FMEMB (U-CASE ty:typeType:typeName)
                                   \LupineDummyTypes)))
                   then (allocation_ (\Allocate ty:typeType typeTable))
                        (newEntry:typeSize_allocation:need)
                        (newEntry:typeBits_allocation:fields))
               (push typeTable newEntry))
       finally (RETURN typeTable])


(\Allocate
  [LAMBDA (type typeTable subFlg)                                      (* ht: "29-Jul-85 13:38")
    (let ((typeName (\TypeName type))
          (typeParm (if (LISTP type)
                        then type:typeParm))
          res rRes)
         res_
         [SELECTQ typeName
              ((STRING ATOM LIST STREAM ARB FIXP REF SEQUENCE)
                   32)
              (SSMALLP 16)
              ((RECORD SEQRECORD)
                   rRes_ (\AllocateRecord typeName typeParm typeTable)
                         (if subFlg
                             then rRes:need
                           else rRes))
              (if (FMEMB typeName \LupinePrimativeTypes)
                  then (let [(need (SELECTQ typeName
                                         (BOOLEAN 1)
                                         (BITS typeParm:1)
                                         (ENUMERATION (IMAX 1 (bind (max _ (SUB1 (LENGTH typeParm)))
                                                                 until (ZEROP max) count max_ (LRSH max 1))))
                                         (SHOULDNT]
                            (if (IGREATERP need 16)
                                then (SHOULDNT "Too big")
                              else need))
                else                                                   (* user defined type)
                     (fetch typeSize of (\TypeLayout typeName typeTable]
         (if subFlg
             then (LIST res)
           else (OR (LISTP res)
                    (create RecordLayout
                           need _ res
                           fields _ (LIST res])


(\AllocateRecord
  [LAMBDA (typeName typeParm typeTable)                                (* ht: "24-Jul-85 19:33")
    (bind (bitsLeft _ 16)
          (wordsUsed _ 0)
          left sub need for t in typeParm join (sub_ (\Allocate t:fieldType typeTable T))
                                               (need_sub:1)
                                               (if (IGREATERP need bitsLeft)
                                                   then (if (ZEROP bitsLeft)
                                                            then

          (* * run out -
          fix it)

                                                                 (add wordsUsed 1)
                                                          elseif bitsLeft~=16
                                                            then

          (* * expand the leftmost bit of the last thing to fit)

                                                                 (add left:1 bitsLeft)
                                                                 (add wordsUsed 1))
                                                        (bitsLeft_16)
                                                        (if (IGREATERP need 15)
                                                            then

          (* * must be some number of words)

                                                                 (add wordsUsed (LRSH need 4))
                                                                 (need_0)
                                                          else bitsLeft_bitsLeft-need)
                                                 else bitsLeft_bitsLeft-need)
                                               (left_sub)
       finally (if (ZEROP bitsLeft)
                   then (add wordsUsed 1)
                 elseif (AND (NOT (ZEROP wordsUsed))
                             bitsLeft~=16)
                   then

          (* * only sub-word records are allowed to be not a multiple of 16 -
          pad)

                        (add left:1 bitsLeft)
                        (add wordsUsed 1))
                (RETURN (create RecordLayout
                                need _ (if (ZEROP wordsUsed)
                                           then 16-bitsLeft
                                         else (LLSH wordsUsed 4))
                                fields _ $$VAL])
)
```

(* * Utilities)

```
(DEFINEQ

(\TypeName
  [LAMBDA (type)                                                     (* ht: "26-Jul-85 09:14")
    (LET ((typen (if (LISTP type)
                     then type:typeName
                   else type)))
         (if (FMEMB (U-CASE typen)
                    \LupinePrimativeTypes)
             then (U-CASE typen)
           else typen])

(\BaseType
  [LAMBDA (type typeTable)                                           (* ht: "26-Jul-85 09:04")
    (LET ((typeName (\TypeName type)))
         (if (FMEMB typeName \LupinePrimativeTypes)
             then type
           else (\BaseType (fetch typeType of (\TypeLayout typeName typeTable))
                       typeTable])

(\TypeLayout
  [LAMBDA (typeName typeTable)                                       (* ht: "24-Jul-85 11:32")
    (OR (ASSOC typeName typeTable)
        (HELP "Type not defined" typeName])

(\TypeSize
  [LAMBDA (type typeTable)                                           (* ht: "31-Jul-85 09:21")
    (LET ((typeName (\TypeName type))
          entry)
         (if entry_ (ASSOC typeName typeTable)
             then entry:typeSize
           elseif (FMEMB typeName \LupineNotFixedTypes)
             then 32
           else 16])

(\LupineNotFixed
  [LAMBDA (field typeTable)                                          (* ht: "26-Jul-85 09:08")
    (FMEMB (\TypeName (\BaseType field:fieldType typeTable))
           \LupineNotFixedTypes])

(\StaticsFirst
  [LAMBDA (specs typeTable)                                          (* ht: "25-Jul-85 16:13")
    (bind nonStatics for s in specs when (if (U-CASE s:argName)
                                              ~= 'RETURNS
                                              then (if (\IsStatic s:argType typeTable)
                                                       else (nonStatics_ (NCONC1 nonStatics s))
                                                            NIL))
       collect s finally (RETURN (NCONC $$VAL nonStatics])

(\IsStatic
  [LAMBDA (type typeTable)                                           (* ht: "26-Jul-85 09:01")
    (LET* ((trueType (\BaseType type typeTable))
           (typeName (\TypeName trueType)))
          (OR (MEMB typeName \LupineStatics)
              (AND typeName= 'RECORD (for f in trueType:typeParm always (\IsStatic f:fieldType typeTable])
)

(RPAQQ \LupineGetFns ((SSMALLP . \GetArgSmallp)
                      (FIXP . \GetArgDblWord)
                      (BOOLEAN . \GetArgBool)
                      (STRING . \UnmarshalString)
                      (ATOM . \UnmarshalAtom)
                      (STREAM . \UnmarshalStream)
                      (ENUMERATION . \GetArgEnum)
                      (ARB . \UnmarshalArb)))

(RPAQQ \LupinePutFns ((SSMALLP . \AddPupSmallp)
                      (FIXP . \AddPupDblWord)
                     (BOOLEAN . \AddPupBoolean)
                     (STRING . \MarshalString)
                     (ATOM . \MarshalAtom)
                     (STREAM . \MarshalStream)
                     (ENUMERATION . \AddPupEnum)
                     (ARB . \MarshalArb)
                     (BITS . \AddPupWord)))

(RPAQQ \LupinePrimativeTypes (SSMALLP FIXP BOOLEAN STRING ATOM STREAM ENUMERATION ARB BITS LIST RECORD RESULT
                                SIGARGS REF SEQRECORD SEQUENCE))

(RPAQQ \LupineInitialTypeTable
       ((CARDINAL (BITS 16)
               NIL 16 16)
        (* the next is not true, but is as close as we get)
        (LONGCARDINAL (BITS 32)
               NIL 32 32)))

(RPAQQ \LupineNumberTypes (FIXP SSMALLP ENUMERATION BITS))

(RPAQQ \LupineNotFixedTypes (STRING ATOM STREAM ARB LIST REF))

(RPAQQ \LupineDummyTypes (RESULT SIGARGS))

(RPAQQ \LupineStatics (SSMALLP FIXP BOOLEAN ENUMERATION BITS))

(RPAQQ \LupineTypesWithParm (ENUMERATION BITS LIST REF SEQRECORD SEQUENCE RECORD RESULT SIGARGS STREAM))

(DECLARE: DOEVAL@COMPILE DONTCOPY

(GLOBALVARS \LupineGetFns \LupinePutFns \LupineStatics \LupineTypesWithParm \LupinePrimativeTypes
       \LupineDummyTypes \LupineNotFixedTypes \LupineNumberTypes \LupineInitialTypeTable)
)

(DECLARE: DONTCOPY

(DECLARE: EVAL@COMPILE

(RPAQQ \FirstLupineUserCall 4)

(RPAQQ \FirstLupineSignal 4)

(CONSTANTS (\FirstLupineUserCall 4)
       (\FirstLupineSignal 4))
)
)

(PUTPROPS LUPINE COPYRIGHT ("Xerox Corporation" 1984 1985 1986))
```

{MEDLEY}<obsolete>lispusers>LUPINE.;1 28-Jun-2024 18:34:03

-- Listed on 30-Jun-2024 13:23:20 --

FUNCTION INDEX

VARIABLE INDEX

RECORD INDEX

CONSTANT INDEX