

File created: 4-Jan-93 18:04:53 {DSK}<python>lde>lispcore>sources>DEFSTRUCT.;2

previous date: 11-Jun-92 14:44:30 {DSK}<python>lde>lispcore>sources>DEFSTRUCT.;1

Read Table: XCL

Package: LISP

Format: XCCS

; Copyright (c) 1986, 1987, 1990, 1988, 1989, 1990, 1992, 1993 by Venue & Xerox Corporation. All rights reserved.

(IL:RPAQQ IL:DEFSTRUCTCOMS

(

;;; Implementation of Structure facilities of Common Lisp. (Chapter 19 of CLtL).

;;; public interface

(IL:DEFINE-TYPES IL:STRUCTURES)
(IL:FUNCTIONS DEFSTRUCT)

;;; top-level

(IL:DECLARE\ : IL:DOCOPY IL:DONTEVAL@LOAD (IL:FILES IL:DEFSTRUCT-RUN-TIME))

;;; parsing code

(IL:VARIABLES %DEFAULT-DEFSTRUCT-TYPE %DEFAULT-SLOT-TYPE %DEFAULT-STRUCTURE-INCLUDE
%DEFSTRUCT-OPTIONS %NO-CONSTRUCTOR %NO-PREDICATE %NO-COPIER %DEFSTRUCT-CONSP-OPTIONS
%DEFSTRUCT-EXPORT-OPTIONS)
(IL:FUNCTIONS ASSIGN-SLOT-ACCESSOR REMOVE-DOCUMENTATION RECORD-DOCUMENTATION ENSURE-VALID-TYPE
PARSE-SLOT DEFSTRUCT-PARSE-OPTIONS ENSURE-CONSISTENT-PS PS-NUMBER-OF-SLOTS PS-TYPE-SPECIFIER)

;;; slot resolution code

(IL:FUNCTIONS ASSIGN-SLOT-OFFSET RESOLVE-SLOTS INSERT-INCLUDED-SLOT MERGE-SLOTS NAME-SLOT DUMMY-SLOT
OFFSET-SLOT)

;;; data layout code

(IL:FUNCTIONS ASSIGN-STRUCTURE-REPRESENTATION COERCE-TYPE %STRUCTURE-TYPE-TO-FIELDSPEC
ASSIGN-FIELD-DESCRIPTORS STRUCTURE-POINTER-SLOTS)

;;; type system hooks

(IL:FUNCTIONS PROCESS-TYPE PREDICATE-BODY TYPE-EXPAND-STRUCTURE TYPE-EXPAND-NAMED-STRUCTURE
PS-NAME-SLOT-POSITION DEFAULT-PREDICATE-NAME DEFSTRUCT-SHARED-PREDICATE-OPTIMIZER
CACHE-PREDICATE-INFO)
(IL:VARIABLES %FUNCTION-DEFINING-FORM-KEYWORDS)

;;; accessors and setfs

(IL:FUNCTIONS SETF-NAME)
(IL:FUNCTIONS ACCESSOR-BODY PROCESS-ACCESSORS ESTABLISH-ACCESSORS DEFINE-ACCESSORS
DEFSTRUCT-SHARED-ACCESSOR-OPTIMIZER DEFSTRUCT-SHARED-SETF-EXPANDER CACHE-SLOT-INFO)
(IL:FUNCTIONS %MAKE-ACCESSOR-CLOSURE %MAKE-LIST-ACCESSOR %MAKE-ARRAY-ACCESSOR %MAKE-POINTER-ACCESSOR
%MAKE-BIT-ACCESSOR %MAKE-FLAG-ACCESSOR %MAKE-WORD-ACCESSOR %MAKE-FIXP-ACCESSOR
%MAKE-SMALL-FIXP-ACCESSOR %MAKE-FLOAT-ACCESSOR)

;;; constructor definition code

(IL:FUNCTIONS DEFINE-CONSTRUCTORS DEFINE-BOA-CONSTRUCTOR ARGUMENT-NAMES
BOA-ARG-LIST-WITH-INITIAL-VALUES BOA-SLOT-SETFS FIND-SLOT RAW-CONSTRUCTOR
BUILD-CONSTRUCTOR-ARGLIST BUILD-CONSTRUCTOR-SLOT-SETFS BOA-CONSTRUCTOR-P
DEFAULT-CONSTRUCTOR-NAME)

;;; copiers

(IL:FUNCTIONS DEFINE-COPIERS BUILD-COPIER-SLOT-SETFS BUILD-COPIER-TYPE-CHECK)

;;; print functions

(IL:VARIABLES %DEFAULT-PRINT-FUNCTION)

;;; internal stuff.

(IL:SETFS IL:FFETCHFIELD)

;;; utilities

(IL:FUNCTIONS DEFSTRUCT-ASSERT-SUBTYPEP)

;;; inspecting structures

(IL:FUNCTIONS STRUCTURE-OBJECT-P INSPECT-STRUCTURE-OBJECT STRUCTURE-OBJECT-INSPECT-FETCHFN
STRUCTURE-OBJECT-INSPECT-PROPPRINTFN STRUCTURE-OBJECT-INSPECT-STOREFN
STRUCTURE-OBJECT-PROPCOMMANDFN)

;; Defined last so functions required to load a defstruct are loaded first

(IL:STRUCTURES PS PARSED-SLOT)

;; Mapping between names of generated functions and their associated structures

(IL:FUNCTIONS STRUCTURE-FUNCTION-P STRUCTURE-FUNCTIONS)

;;; Editing structures

(IL:FUNCTIONS STRUCTURES.HASDEF STRUCTURES.EDITDEF)
(IL:P (IL:FILEPKGTYPE 'IL:STRUCTURES 'IL:HASDEF 'STRUCTURES.HASDEF 'IL:EDITDEF 'STRUCTURES.EDITDEF))
(IL:ADDVARS (IL:SHADOW-TYPES (IL:STRUCTURES IL:FNS)))
(IL:DECLARE\ IL:DOCOPY IL:DONTEVAL@LOAD (IL:ADDVARS (IL:INSPECTMACROS ((IL:FUNCTION
STRUCTURE-OBJECT-P
INSPECT-STRUCTURE-OBJECT
))))

;;; file properties

(IL:PROP IL:FILETYPE IL:DEFSTRUCT)
(IL:PROP IL:MAKEFILE-ENVIRONMENT IL:DEFSTRUCT))

;;; Implementation of Structure facilities of Common Lisp. (Chapter 19 of CLtL).

;;; public interface

(XCL:DEF-DEFINE-TYPE **IL:STRUCTURES** "Common Lisp structures")

(XCL:DEFDEFINER (**DEFSTRUCT** (:NAME (LAMBDA (WHOLE)
(LET ((NAME-AND-OPTIONS (SECOND WHOLE))
(IF (CONSP NAME-AND-OPTIONS)
(CAR NAME-AND-OPTIONS)
NAME-AND-OPTIONS))))
(:PROTOTYPE (LAMBDA (NAME)
(AND (SYMBOLP NAME)
(DEFSTRUCT (,NAME (:option "value")
"documentation string"
"slot-name" "initial-value"))))))
IL:STRUCTURES (NAME &REST SLOT-DESCRIPTIONS)
(LET* ((PS (**DEFSTRUCT-PARSE-OPTIONS** NAME))
(SLOT-DESCRIPTIONS (**REMOVE-DOCUMENTATION** PS SLOT-DESCRIPTIONS)))
(**RESOLVE-SLOTS** SLOT-DESCRIPTIONS PS)
(PROGN (EVAL-WHEN (EVAL COMPILE LOAD)
(SETF (PARSED-STRUCTURE ',(PS-NAME PS)
T)
(,PS))
,@(**ASSIGN-STRUCTURE-REPRESENTATION** PS)
,@(**PROCESS-TYPE** PS)
,@(**PROCESS-ACCESSORS** PS)
(EVAL-WHEN (EVAL COMPILE LOAD)
(ESTABLISH-SETFS-AND-OPTIMIZERS ',(PS-NAME PS)))
,@(**DEFINE-CONSTRUCTORS** PS)
,@(**DEFINE-COPIERS** PS)
,@(**RECORD-DOCUMENTATION** PS))))

;;; top-level

(IL:DECLARE\ IL:DOCOPY IL:DONTEVAL@LOAD
(IL:FILESLOAD IL:DEFSTRUCT-RUN-TIME)
)

;;; parsing code

(DEFVAR **%DEFAULT-DEFSTRUCT-TYPE** 'DATATYPE "The type of structures when no :type option is specified")

(DEFVAR **%DEFAULT-SLOT-TYPE** 'T "the type of any slot which does not specify a :type option")

(DEFCONSTANT **%DEFAULT-STRUCTURE-INCLUDE** 'STRUCTURE-OBJECT "datatype included by every structure")

(DEFPARAMETER **%DEFSTRUCT-OPTIONS**

' (:CONC-NAME :CONSTRUCTOR :COPIER :PREDICATE :INCLUDE :PRINT-FUNCTION :TYPE :INITIAL-OFFSET :NAMED :INLINE :FAST-ACCESSORS :TEMPLATE :EXPORT))

(DEFCONSTANT **%NO-CONSTRUCTOR** ':NONE "the value which says that no constructor was specified.")

(DEFCONSTANT **%NO-PREDICATE** ':NONE "the value which says that no constructor was specified")

(DEFCONSTANT **%NO-COPIER** ':NONE)

(DEFPARAMETER **%DEFSTRUCT-CONSP-OPTIONS** (REMOVE ':NAMED %DEFSTRUCT-OPTIONS))

(DEFPARAMETER **%DEFSTRUCT-EXPORT-OPTIONS** ' (:ACCESSOR :CONSTRUCTOR :PREDICATE :COPIER))

(DEFUN **ASSIGN-SLOT-ACCESSOR** (SLOT CONC-NAME)

;; assigns the accessor name to a slot

(IF (PSLOT-ACCESSOR SLOT)
 (SETF (PSLOT-ACCESSOR SLOT)
 (VALUES (INTERN (CONCATENATE 'STRING (STRING CONC-NAME)
 (STRING (PSLOT-NAME SLOT)))))))

(DEFUN **REMOVE-DOCUMENTATION** (PS SLOT-DESCRIPTIONS)

;; Records it if there is any documentation string.

(LET ((DOC? (CAR SLOT-DESCRIPTIONS))
 (COND ((STRINGP DOC?)
 ;; save it and return the rest of the slots.
 (SETF (PS-DOCUMENTATION-STRING PS)
 DOC?)
 (REST SLOT-DESCRIPTIONS))
 (T ;; no doc string, return the whole thing.
 SLOT-DESCRIPTIONS))))

(DEFUN **RECORD-DOCUMENTATION** (PS)

;; Returns a form which saves the documentation string for a structure.

(LET ((PARSED-DOCSTRING (PS-DOCUMENTATION-STRING PS))
 (IF PARSED-DOCSTRING
 '((SETF (DOCUMENTATION ', (PS-NAME PS)
 'STRUCTURE)
 , PARSED-DOCSTRING))))

(DEFUN **ENSURE-VALID-TYPE** (TYPE-FORM)

;; Bogus right now

TYPE-FORM)

(DEFUN **PARSE-SLOT** (DESCRIPTION &OPTIONAL (GENERATE-ACCESSOR T))

;; Takes a slot description from the defstruct body or included slots and returns a parsed version

(LET* ((DESCRIPTION (IF (CONSP DESCRIPTION)
 DESCRIPTION
 (LIST DESCRIPTION)))
 (SLOT (MAKE-PARSED-SLOT)))
 (XCL:DESTRUCTURING-BIND (NAME &OPTIONAL INITIAL-VALUE &REST SLOT-OPTIONS)
 DESCRIPTION
 (IF (SYMBOLP NAME)
 (SETF (PSLOT-NAME SLOT)
 NAME)
 (ERROR "Slot name not symbol: ~S" NAME))
 (SETF (PSLOT-INITIAL-VALUE SLOT)
 INITIAL-VALUE))

;; some variant of PCL's keyword-bind would be easier here, but it's incapable of producing reasonable error msgs for the user.
;; Maybe later.

(DO ((OPTION-PAIR SLOT-OPTIONS (CDDR OPTION-PAIR)))

```

((NULL OPTION-PAIR))
(CASE (CAR OPTION-PAIR)
  (:TYPE (SETF (PSLOT-TYPE SLOT)
    (ENSURE-VALID-TYPE (CADR OPTION-PAIR))))
  (:READ-ONLY (SETF (PSLOT-READ-ONLY SLOT)
    (AND (CADR OPTION-PAIR)
      T)))
  (OTHERWISE (IF (KEYWORDP INITIAL-VALUE)
    (ERROR "Initial value must be specified to use slot options. ~S"
      DESCRIPTION)
    (ERROR "Illegal slot option ~S in slot ~S" (CAR OPTION-PAIR)
      NAME))))))
(IF GENERATE-ACCESSOR
  (SETF (PSLOT-ACCESSOR SLOT)
    T))
SLOT))

```

(DEFUN **DEFSTRUCT-PARSE-OPTIONS** (NAME&OPTIONS)

;; Returns a structure representing the options in a defstruct call.

```

(LET* ((OPTIONS (IF (LISTP NAME&OPTIONS)
  NAME&OPTIONS
  (LIST NAME&OPTIONS)))
  (NAME (POP OPTIONS))
  (PS (MAKE-PS :NAME NAME :CONC-NAME (CONCATENATE 'STRING (STRING NAME)
    "-"))))
  (DOLIST (OPTION OPTIONS)
    (COND
      ((LISTP OPTION)
        (XCL:DESTRUCTURING-BIND (OPTION-KEYWORD &OPTIONAL (OPTION-VALUE NIL ARGUMENT-PROVIDED)
          &REST FURTHER-ARGUMENTS)
          OPTION
          (CASE OPTION-KEYWORD
            (:CONC-NAME
              ;; if the option is specified, but the option value is nil, then use the empty string as conc-name
              (SETF (PS-CONC-NAME PS)
                (OR OPTION-VALUE "")))
            (:CONSTRUCTOR
              ;; multiple constructors are allowed. If NIL is provided, then define no constructor.
              (COND
                ((NOT OPTION-VALUE)
                  (IF ARGUMENT-PROVIDED
                    ;; NIL was specified. Record that no constructor is to be built.
                    (SETF (PS-CONSTRUCTORS PS)
                      NIL)
                    ;; otherwise, it as though the option weren't specified (p. 312 clt) so leave the default value there.
                    ))
                ))
              (EQ (PS-CONSTRUCTORS PS)
                %NO-CONSTRUCTOR)
              ;; this is the first constructor specified. Make the field be a list now.
              (SETF (PS-CONSTRUCTORS PS)
                (LIST (IF FURTHER-ARGUMENTS
                  (CDR OPTION)
                  OPTION-VALUE))))
              (T
                ;; just push another one on the list of constructors.
                (PUSH (IF FURTHER-ARGUMENTS
                  (CDR OPTION)
                  OPTION-VALUE)
                  (PS-CONSTRUCTORS PS))))
            (:COPIER
              ;; if the argument is specified (even if it is nil), use it. Otherwise use the default COPY- form already in the ps.
              (IF ARGUMENT-PROVIDED
                (SETF (PS-COPIER PS)
                  OPTION-VALUE))
              (:PREDICATE (IF ARGUMENT-PROVIDED
                (SETF (PS-PREDICATE PS)
                  OPTION-VALUE))
              (:INCLUDE
                (SETF (PS-INCLUDE PS)
                  OPTION-VALUE)
                ;; if there are any included slots record them
                (SETF (PS-INCLUDED-SLOTS PS)
                  (CDDR OPTION)))
              (:PRINT-FUNCTION (COND
                ((AND ARGUMENT-PROVIDED (NULL OPTION-VALUE))
                  ;; extension to CLtL, if NIL is specified as the defprint, then the internal print function is
                  ;; specified.

```

```

      (SETF (PS-PRINT-FUNCTION PS)
            'IL:\PRINT-USING-ADDRESS))
      (ARGUMENT-PROVIDED (SETF (PS-PRINT-FUNCTION PS)
                                OPTION-VALUE))))
  (:TYPE (SETF (PS-TYPE PS)
              (COND
                ((EQ OPTION-VALUE 'LIST)
                 'LIST)
                ((EQ OPTION-VALUE 'VECTOR)
                 ; default the vector type to t
                 (SETF (PS-VECTOR-TYPE PS)
                       T)
                 'VECTOR)
                ((AND (CONSP OPTION-VALUE)
                      (EQ (CAR OPTION-VALUE)
                          'VECTOR))
                 (SETF (PS-VECTOR-TYPE PS)
                       (IL:%GET-CANONICAL-CML-TYPE (CADR OPTION-VALUE)))
                 'VECTOR)
                (T (ERROR "the specified :type is not list or subtype of vector:
                          ~S" OPTION-VALUE))))))
  (:INITIAL-OFFSET
   (IF (NOT (TYPEP OPTION-VALUE '(INTEGER 0 *)))
       (ERROR ":initial-offset isn't a non-negative integer: ~S" OPTION-VALUE))
   (SETF (PS-INITIAL-OFFSET PS)
         OPTION-VALUE))
  (:INLINE
   ;; Is one or both of :accessor, and :predicate or t, which is equivalent to both
   ;; Default is '(:accessor :predicate)
   ;; option (:inline :only) implies no fucallable accessors or predicate is generated
   (IF ARGUMENT-PROVIDED
       (SETF (PS-INLINE PS)
             OPTION-VALUE)))
  (:FAST-ACCESSORS
   ;; Is either t or nil, t implying no type checks for all accessors
   (IF ARGUMENT-PROVIDED
       (SETF (PS-FAST-ACCESSORS PS)
             OPTION-VALUE)))
  (:TEMPLATE
   ;; Is either t or nil -- t implying type datatype, no copier, predicate, print-function or constructors, and fast
   ;; accessors, and no new datatype declared.
   (IF ARGUMENT-PROVIDED
       (SETF (PS-TEMPLATE PS)
             OPTION-VALUE)))
  (:EXPORT
   ;; Edited by TT(13-June-90) Export Option is added for DEFSTRUCT(Medley 1.2). The Specified functions(ex
   ;; :constructor, :copier...) will be exported.
   (IF FURTHER-ARGUMENTS
       (ERROR "The specified export functions is not list or atom : ~S"
              (CONS :EXPORT (CONS OPTION-VALUE FURTHER-ARGUMENTS))))
   (IF ARGUMENT-PROVIDED
       (SETF (PS-EXPORT PS)
             OPTION-VALUE)
       (SETF (PS-EXPORT PS)
             T))))
   (OTHERWISE (ERROR "Bad option to defstruct: ~S." OPTION))))))
  (T (CASE OPTION
      (:NAMED (SETF (PS-NAMED PS)
                   T))
      (OTHERWISE (IF (MEMBER OPTION %DEFSTRUCT-CONSP-OPTIONS :TEST #'EQ)
                    (ERROR "defstruct option ~s must be in parentheses with its value" OPTION)
                    (ERROR "Bad option to defstruct: ~S." OPTION))))))
  (ENSURE-CONSISTENT-PS PS)
  (PS))

```

(DEFUN ENSURE-CONSISTENT-PS (PS)

;; Accomplishes the consistency checks that can't occur until all the options have been parsed.

```

  (IF (PS-INCLUDE PS)
      (LET* ((INCLUDE (PS-INCLUDE PS))
            (INCLUDED-PSTRUCTURE (PARSED-STRUCTURE INCLUDE)))
          ;; ensure that the user is not suicidal. If a structure includes itself, a *very* tight ucode loop will occur in the instancecp opcode.
          (IF (EQ INCLUDE (PS-NAME PS))
              (ERROR "You probably don't want ~S to include ~S." INCLUDE INCLUDE))
          ;; ensure that the included structure is defined.
          (IF (OR (NULL INCLUDED-PSTRUCTURE)
                  (PS-TEMPLATE INCLUDED-PSTRUCTURE))
              (ERROR "Included structure ~s is unknown or not instantiated." INCLUDE))
          ;; make sure the type of the included structure is the same

```

```

(IF (OR (NOT (EQ (PS-TYPE INCLUDED-PSTRUCTURE)
                (PS-TYPE PS)))
        (NOT (EQ (PS-VECTOR-TYPE INCLUDED-PSTRUCTURE)
                (PS-VECTOR-TYPE PS))))
    (ERROR "~s must be same type as included structure ~s" (PS-NAME PS)
            INCLUDE)))
(LET ((INLINE (PS-INLINE PS))
      (POSSIBLE-KEYWORDS '(:ACCESSOR :PREDICATE)))
    (CASE INLINE
      ((T)
       ;; this is the default case, so make the default be that only the accessors, predicates are inline.
       (SETF (PS-INLINE PS)
             POSSIBLE-KEYWORDS)
       ((NIL :ONLY) )
       (OTHERWISE (MAPCAR #'(LAMBDA (KEYWORD)
                            (IF (NOT (MEMBER KEYWORD POSSIBLE-KEYWORDS :TEST #'EQ))
                                (ERROR "~s must be one of ~s." KEYWORD POSSIBLE-KEYWORDS)))
                          (IF (CONSP INLINE)
                              INLINE
                              (SETF (PS-INLINE PS)
                                    (LIST INLINE)))))))
      (COND
        ((PS-TEMPLATE PS)
         (IF (NOT (EQ (PS-TYPE PS)
                     %DEFAULT-DEFSTRUCT-TYPE))
             (ERROR "Templated defstructs may not be of type: ~s" (PS-TYPE PS)))
         (IF (OR (NOT (EQ (PS-CONSTRUCTORS PS)
                         %NO-CONSTRUCTOR))
                (NOT (EQ (PS-PREDICATE PS)
                         %NO-PREDICATE))
                (NOT (EQ (PS-COPIER PS)
                         %NO-COPIER))
                (PS-PRINT-FUNCTION PS))
             (ERROR "Templated defstructs may not have constructors predicates copiers or print functions")))
        (T (IF (PS-PRINT-FUNCTION PS)
                (IF (NOT (EQ (PS-TYPE PS)
                            %DEFAULT-DEFSTRUCT-TYPE))
                    (ERROR "A print-function can't be specified for structures of type ~s" (PS-TYPE PS)))
                (LET ((INCLUDE (PS-INCLUDE PS)))
                    (IF INCLUDE
                        ;; CLtL is silent, but we inherit print-functions
                        (SETF (PS-PRINT-FUNCTION PS)
                              (PS-PRINT-FUNCTION (PARSED-STRUCTURE INCLUDE)))
                        ;; otherwise, use the default #s style printer
                        (SETF (PS-PRINT-FUNCTION PS)
                              %DEFAULT-PRINT-FUNCTION))))
                (IF (AND (EQ (PS-TYPE PS)
                            'VECTOR)
                        (EQ (PS-NAMED PS)
                            T))
                    ;; check that the vector type can actually hold the symbol required for the name.
                    (DEFSTRUCT-ASSERT-SUBTYPEP 'SYMBOL (PS-VECTOR-TYPE PS)
                                                ("vector of ~S cannot contain the symbol required for the :named options" (PS-VECTOR-TYPE
                                                                 PS))))
                    (IF (EQ (PS-PREDICATE PS)
                            %NO-PREDICATE)
                        ;; there is no predicate. (Note that this is not a null check. If this field is NIL the user explicitly gave NIL as the predicate.)
                        (IF (OR (EQ (PS-TYPE PS)
                                    'DATATYPE)
                                (PS-NAMED PS))
                            ;; If this structure is type datatype or named, use the default name
                            (SETF (PS-PREDICATE PS)
                                  (DEFAULT-PREDICATE-NAME (PS-NAME PS)))
                            ;; now set it to NIL to signal no predicate to the predicate builder.
                            (SETF (PS-PREDICATE PS)
                                  NIL))
                        (IF (EQ (PS-COPIER PS)
                                %NO-COPIER)
                            ;; Note that this is not a null check. If this field is NIL the user explicitly gave NIL as the copier
                            (SETF (PS-COPIER PS)
                                  (INTERN (CONCATENATE 'STRING "COPY-" (STRING (PS-NAME PS))))))
                        (LET ((EXPORTNAMES (PS-EXPORT PS)))
                            ;; If export-slot is nil, functions will not be exported. otherwise, export the specified functions.[Edited by TT (13-June-90)
                            (AND EXPORTNAMES (OR (EQ EXPORTNAMES T)
                                                  (AND (NOT (LISTP EXPORTNAMES))
                                                       (NOT (SETF (PS-EXPORT PS)
                                                                (SETQ EXPORTNAMES (LIST EXPORTNAMES))))))

```

```

(DOLIST (EXPORTNAME EXPORTNAMES T)
  (OR (MEMBER EXPORTNAME %DEFSTRUCT-EXPORT-OPTIONS)
    (ERROR "~S is not valid option keyword for :EXPORT" EXPORTNAME))))))
(COND
  ((EQ (PS-CONSTRUCTORS PS)
    %NO-CONSTRUCTOR)
    ;; There were no constructors specified. Default the value.
    (SETF (PS-CONSTRUCTORS PS)
      `((, (DEFAULT-CONSTRUCTOR-NAME (PS-NAME PS)))))))

```

```

(DEFUN PS-NUMBER-OF-SLOTS (PS)
  "the number of slots in an instance of this structure"
  (LENGTH (PS-ALL-SLOTS PS)))

```

```

(DEFUN PS-TYPE-SPECIFIER (PS)
  "returns list, vector, or (vector foo)"
  (ECASE (PS-TYPE PS)
    (LIST 'LIST)
    (VECTOR (LET ((ELEMENT-TYPE (PS-VECTOR-TYPE PS)))
      (IF (IL:NEQ ELEMENT-TYPE T)
        `(VECTOR ,ELEMENT-TYPE)
        'VECTOR))))))

```

;;; slot resolution code

```

(DEFUN ASSIGN-SLOT-OFFSET (PS)
  ;; Assigns the offsets for each slot for type vector and list.
  (LET* ((NAME (PS-NAME PS))
    (SLOTS (PS-ALL-SLOTS PS)))
    (ECASE (PS-TYPE PS)
      ((VECTOR LIST)
        ;; the field descriptor is just the offset.
        (DO ((I 0 (1+ I))
          (SLOT SLOTS (CDR SLOT))
          (NULL SLOT))
          (SETF (PSLOT-FIELD-DESCRIPTOR (CAR SLOT))
            I))))))

```

```

(DEFUN RESOLVE-SLOTS (LOCAL-SLOT-DESCRIPTIONS PS)
  ;; Combines the slot descriptions from the defstruct call with the included slot-descriptions from supers and the :includes option, and installs the
  ;; description in the parsed-structure
  (LET ((LOCAL-SLOTS (MAPCAR #'PARSE-SLOT LOCAL-SLOT-DESCRIPTIONS))
    (INCLUDED-SLOTS (MAPCAR #'PARSE-SLOT (PS-INCLUDED-SLOTS PS)))
    (INCLUDES (PS-INCLUDE PS)))
    (WHEN (PS-NAMED PS)
      ;; Adds the slot representing the name pseudo-slot.
      (IF (NOT (PS-NAMED PS))
        (ERROR ":named not supplied for this defstruct"))
      (PUSH (NAME-SLOT PS)
        LOCAL-SLOTS))
    (WHEN (NOT (EQ 0 (PS-INITIAL-OFFSET PS)))
      ;; Adds parsed-slots to the local-slots to represent the initial offset.
      (SETQ LOCAL-SLOTS (NCONC (XCL:WITH-COLLECTION (DOTIMES (I (PS-INITIAL-OFFSET PS))
        (XCL:COLLECT (OFFSET-SLOT))))
        LOCAL-SLOTS)))
    (IF INCLUDES
      (LET ((SUPER-SLOTS
        ;; must copy the slots, since the accessor-name will be destructively modified to use the new conc-name.
        (MAPCAR #'COPY-PARSED-SLOT (PS-ALL-SLOTS (PARSED-STRUCTURE INCLUDES))))
          ;; update the super-slots according to the included-slots, then make all-slots be (append merged-slots local-slots)
          (SETF (PS-ALL-SLOTS PS)
            (NCONC (MERGE-SLOTS INCLUDED-SLOTS SUPER-SLOTS PS)
              LOCAL-SLOTS)))
        (PROGN (IF INCLUDED-SLOTS
          (ERROR "Can't include slots when ~s includes no structure." (PS-NAME PS)))
          ;; no included slots, so the local-slots are it.
          (SETF (PS-ALL-SLOTS PS)
            LOCAL-SLOTS)))
      (WHEN (AND (NULL (PS-ALL-SLOTS PS))
        (EQ (PS-TYPE PS)
          %DEFAULT-DEFSTRUCT-TYPE))
        (PUSH (DUMMY-SLOT)
          LOCAL-SLOTS))

```

```

      (SETF (PS-ALL-SLOTS PS)
            LOCAL-SLOTS)
;; No longer require local slots to be recorded
      (SETF (PS-LOCAL-SLOTS PS)
            LOCAL-SLOTS)
;; now that all slots (included, super, local and filler) have been included, we can create accessor names.
      (LET ((CONC-NAME (PS-CONC-NAME PS))
            (DOLIST (SLOT (PS-ALL-SLOTS PS))
                  (ASSIGN-SLOT-ACCESSOR SLOT CONC-NAME)))
;; we can also record slot-names for the default-structure-printer and inspector.
      (SETF (PS-ALL-SLOT-NAMES PS)
            (MAPCAR #'PSLOT-NAME (PS-ALL-SLOTS PS)))
;; make sure that no slot names have been repeated (either from being explicitly listed twice in the defstruct, or using a slot name that is
;; present in the super without using :include for the slot)
      (DO ((SLOT-NAMES (PS-ALL-SLOT-NAMES PS)
                      (CDR SLOT-NAMES))
          ((NULL SLOT-NAMES))
          (IF (MEMBER (CAR SLOT-NAMES)
                      (CDR SLOT-NAMES)
                      :TEST #'EQ)
              (ERROR "The slot ~s is repeated in ~s." (CAR SLOT-NAMES)
                     (PS-ALL-SLOT-NAMES PS))))))

```

```

(DEFUN INSERT-INCLUDED-SLOT (NEW-SLOT SUPER-SLOTS PS)
;; Replaces the slot in super-slots that corresponds to new-slot with new-slot
  (FLET ((SAME-SLOT (SLOT1 SLOT2)
                (EQ (PSLOT-NAME SLOT1)
                    (PSLOT-NAME SLOT2))))
        (LET* ((TAIL (MEMBER NEW-SLOT SUPER-SLOTS :TEST #'SAME-SLOT))
               (OLD-SLOT (CAR TAIL)))
          (IF (NOT TAIL)
              (ERROR "included slot ~S not present in included structure ~S" (PSLOT-NAME NEW-SLOT)
                     (PS-INCLUDE PS)))
              ;; verify the inclusion rules.
              (IF (AND (PSLOT-READ-ONLY OLD-SLOT)
                       (NOT (PSLOT-READ-ONLY NEW-SLOT)))
                  (ERROR "included slot ~s must be read-only. It is in included structure ~S" (PSLOT-NAME NEW-SLOT)
                         (PS-INCLUDE PS)))
                  (DEFSTRUCT-ASSERT-SUBTYPEP (PSLOT-TYPE NEW-SLOT)
                                               (PSLOT-TYPE OLD-SLOT)
                                               ("Included slot ~S's type ~s is not a subtype of original slot type ~s" (PSLOT-NAME NEW-SLOT)
                                                                    (PSLOT-TYPE NEW-SLOT)
                                                                    (PSLOT-TYPE OLD-SLOT)))
                  ;; finally, we can replace the slot
                  (RPLACA TAIL NEW-SLOT))))))

```

```

(DEFUN MERGE-SLOTS (INCLUDED-SLOTS SUPER-SLOTS PS)
;; Takes the included-slots, and the local slots, then merges them with the slots from the super that aren't shadowed.
;; go through the slots from the super and replace the super's def with the overriding included-slot
  (DOLIST (NEW-SLOT INCLUDED-SLOTS)
        (INSERT-INCLUDED-SLOT NEW-SLOT SUPER-SLOTS PS))
SUPER-SLOTS)

```

```

(DEFUN NAME-SLOT (PS)
;; Returns a parsed-slot representing the 'name' field of a structure
  (PARSE-SLOT '(SI:--STRUCTURE-NAME-SLOT-- ', (PS-NAME PS)
              :READ-ONLY T)
              NIL))

```

```

(DEFUN DUMMY-SLOT ()
  (PARSE-SLOT '(SI:--STRUCTURE-DUMMY-SLOT-- NIL :READ-ONLY T :TYPE IL:XPOINTER)
              NIL))

```

```

(DEFUN OFFSET-SLOT ()
  (PARSE-SLOT '(, (GENSYM)
              :READ-ONLY T)
              NIL))
;; to make sure that names are unique, so that when the inspector works on :type list, there will be a unique name.
NIL))

```


;;; data layout code

```
(DEFUN ASSIGN-STRUCTURE-REPRESENTATION (PS)
  ;; Determines the descriptors and returns a form to create the datatype at loadtime.
  ;; Side effects ps.
  (LET ((LOCAL-SLOTS (PS-LOCAL-SLOTS PS)))
    ;; Local slots no longer need be recorded
    (SETF (PS-LOCAL-SLOTS PS)
          NIL)
    (CASE (PS-TYPE PS)
      ((VECTOR LIST)
       ;; just assign the the field descriptors (offsets). No run-time declaration is needed since the representation is known (list and vector)
       (ASSIGN-SLOT-OFFSET PS)
       NIL)
      (DATATYPE (LET* ((LOCAL-FIELD-SPECS (MAPCAR #'(LAMBDA (SLOT)
                                                    (%STRUCTURE-TYPE-TO-FIELDSPEC (PSLOT-TYPE
                                                                                   SLOT)))
                                                            LOCAL-SLOTS))
                     (SUPER-FIELD-SPECS (IF (PS-INCLUDE PS)
                                             (PS-FIELD-SPECIFIERS (PARSED-STRUCTURE (PS-INCLUDE PS))))
                     (ALL-FIELD-SPECS (APPEND SUPER-FIELD-SPECS LOCAL-FIELD-SPECS))
                     (STRUCTURE-NAME (PS-NAME PS)))
        (SETF (PS-FIELD-SPECIFIERS PS)
              ALL-FIELD-SPECS)
        (XCL:DESTRUCTURING-BIND (LENGTH &REST FIELD-DESCRIPTORS)
          (IL:TRANSLATE.DATATYPE (IF (NOT (PS-TEMPLATE PS))
                                     STRUCTURE-NAME)
                                ALL-FIELD-SPECS))
          ;; Note that this side-effects ps
          (ASSIGN-FIELD-DESCRIPTORS PS FIELD-DESCRIPTORS)
          ;; save the descriptors? No, even though the ones in the dtd are for the current world, not the
          ;; crosscompiling world. They are recomputed each redeclaration by TRANSLATE.DATATYPE
          (IF (NOT (PS-TEMPLATE PS))
              `((SI::%STRUCTURE-DECLARE-DATATYPE ' ,STRUCTURE-NAME ' ,ALL-FIELD-SPECS
              ' ,FIELD-DESCRIPTORS
              ' ,LENGTH
              ' , (OR (PS-INCLUDE PS)
                      %DEFAULT-STRUCTURE-INCLUDE))))))))))
```

```
(DEFUN COERCE-TYPE (ELEMENT-TYPE)
  ;; As in IL:%canonical-cml-type -- Returns the types (t, string-char, single-float, IL:xpointer, (unsigned-byte n) and (signed-byte n)
  (IF (CONSP ELEMENT-TYPE)
      (CASE (CAR ELEMENT-TYPE)
        (UNSIGNED-BYTE
         ;; Let the bits hang out
         (IF (> (CADR ELEMENT-TYPE)
                16)
             T
             ELEMENT-TYPE))
        (SIGNED-BYTE (IL:%GET-ENCLOSING-SIGNED-BYTE ELEMENT-TYPE))
        (MOD
         ;; From cmlarray -- reduces (mod n) to (unsigned-byte m)
         (IL:%REDUCE-MOD ELEMENT-TYPE))
        (INTEGER
         ;; From cmlarray -- reduces (integer x y) to (signed-byte m)
         (IL:%REDUCE-INTEGGER ELEMENT-TYPE))
        (MEMBER (IF (AND (EQ 2 (LENGTH (CDR ELEMENT-TYPE)))
                        (EVERY #'(LAMBDA (ELT)
                                (OR (EQ ELT T)
                                    (EQ ELT NIL))))
                    (CDR ELEMENT-TYPE)))
                ELEMENT-TYPE
                T))
        (T
         ;; Attempt type expansion
         (LET ((EXPANDER (TYPE-EXPANDER (CAR ELEMENT-TYPE)))
              (IF EXPANDER
                  (COERCE-TYPE (TYPE-EXPAND ELEMENT-TYPE EXPANDER))
                  T))))
      (CASE ELEMENT-TYPE
        ((T IL:FULLPOINTER IL:XPOINTER IL:FULLXPOINTER SINGLE-FLOAT STRING-CHAR) ELEMENT-TYPE)
        (IL:POINTER T)
        ((FLOAT SHORT-FLOAT LONG-FLOAT DOUBLE-FLOAT) 'SINGLE-FLOAT)
        (FIXNUM
```

;; Could be (signed-byte 32) -- but pointer representation is more efficient

```
T)
(Character 'STRING-CHAR)
(BIT '(UNSIGNED-BYTE 1))
(T (LET ((EXPANDER (TYPE-EXPANDER ELEMENT-TYPE)))
      (IF EXPANDER
          (COERCE-TYPE (TYPE-EXPAND ELEMENT-TYPE EXPANDER))
          T))))))
```

(DEFUN %STRUCTURE-TYPE-TO-FIELDSPEC (ELEMENT-TYPE)

;; Returns the most specific InterLisp type descriptor which will hold a given type.

;; Note: This function accepts only a limited subset of the Common Lisp type specifiers: T FLOAT SINGLE-FLOAT FIXNUM BIT (MOD n)
 ;; (UNSIGNED-BYTE n) INTEGER (INTEGER low high) IL:XPOINTER DOUBLE-IL:POINTER

```
(LET ((COERCED-TYPE (COERCE-TYPE ELEMENT-TYPE)))
  (IF (NOT (CONSP COERCED-TYPE))
      (CASE COERCED-TYPE
        ((T STRING-CHAR) 'IL:POINTER)
        ((IL:FULLPOINTER IL:XPOINTER IL:FULLXPOINTER) COERCED-TYPE)
        ((SINGLE-FLOAT) 'IL:FLOATP)
        (OTHERWISE 'IL:POINTER))
      (CASE (CAR COERCED-TYPE)
        (UNSIGNED-BYTE `(IL:BITS , (CADR COERCED-TYPE)))
        (SIGNED-BYTE (CASE (CADR COERCED-TYPE)
                           (16 'IL:SIGNEDWORD)
                           (32 'IL:FIXP)
                           (OTHERWISE 'IL:POINTER))))
        (MEMBER 'IL:FLAG)
        (OTHERWISE 'IL:POINTER))))))
```

(DEFUN ASSIGN-FIELD-DESCRIPTORS (PS FIELD-DESCRIPTORS)

;; Assigns the field descriptors for accessing each slot of the structure

```
(IF (NOT (EQ (PS-TYPE PS)
             'DATATYPE))
    (ERROR "Not a structure of type datatype"))
(DO ((F FIELD-DESCRIPTORS (CDR F))
     (SLOT (PS-ALL-SLOTS PS)
           (CDR SLOT)))
    ((NULL F))
  (SETF (PSLOT-FIELD-DESCRIPTOR (CAR SLOT))
        (CAR F)))
```

;; DON'T record where the pointer fields are for the circle printer. it will do this when it needs them.

;; (setf (ps-pointer-descriptors ps) (mapcan #'(lambda (descriptor) (case (caddr descriptor) ((il:pointer il:fullpointer il:xpointer il:fullxpointer) (list descriptor)))) field-descriptors))

)

(DEFUN STRUCTURE-POINTER-SLOTS (STRUCTURE-NAME)

;; record where the pointer fields are for the circle printer.

```
(LET ((PS (PARSED-STRUCTURE STRUCTURE-NAME))
      (OR (PS-POINTER-DESCRIPTORS PS)
          (SETF (PS-POINTER-DESCRIPTORS PS)
                (MAPCAN #'(LAMBDA (DESCRIPTOR)
                          (CASE (CADDR DESCRIPTOR)
                            ((IL:POINTER IL:FULLPOINTER IL:XPOINTER IL:FULLXPOINTER) (LIST DESCRIPTOR
                                                                                               ))))
                        (MAPCAR #'PSLOT-FIELD-DESCRIPTOR (PS-ALL-SLOTS PS)))))))
```

;; type system hooks

(DEFUN PROCESS-TYPE (PS)

;; adds the structure to the common lisp type system and defines the predicate, if any.

```
(IF (NOT (PS-TEMPLATE PS))
    (LET*
      ((NAME (PS-NAME PS))
       (TYPE (PS-TYPE PS))
       (PREDICATE (PS-PREDICATE PS))
       (PREDICATE-BODY (AND PREDICATE (PREDICATE-BODY PS 'OBJECT)))
       (EXPORTNAME (PS-EXPORT PS)))
      (IF (AND PREDICATE (OR (EQ EXPORTNAME T)
                             (MEMBER :PREDICATE EXPORTNAME)))
          (EXPORT PREDICATE)
          `(,@ (COND
                ((EQ TYPE 'DATATYPE)
                 `(EVAL-WHEN (EVAL LOAD COMPILE)
```

; Edited by TT(13-June-90) Export Option Follow up

```

      (SETF (TYPE-EXPANDER ',NAME)
            'TYPE-EXPAND-STRUCTURE)))
    ((PS-NAMED PS)
     `((EVAL-WHEN (EVAL LOAD COMPILE)
      (SETF (TYPE-EXPANDER ',NAME)
            'TYPE-EXPAND-NAMED-STRUCTURE))))))
,@(WHEN PREDICATE
  (LET* ((INLINE (PS-INLINE PS))
         (INLINE-P (AND (EQ TYPE 'DATATYPE)
                        (OR (EQ INLINE :ONLY)
                            (AND (CONSP INLINE)
                                (MEMBER :PREDICATE INLINE :TEST #'EQ))))))
        (INLINE-ONLY-P (EQ INLINE :ONLY)))
    (IF (NULL INLINE-P)
        ;; Flush optimizer (a bit extreme, but also gets rid of old definline optimizers from the old defstruct
        (SETF (COMPILER:OPTIMIZER-LIST PREDICATE)
              NIL))
        `(@ (IF (NOT INLINE-ONLY-P)
                `(DEFUN ,PREDICATE (OBJECT)
                  ,PREDICATE-BODY)))
          ,@ (IF INLINE-P
                `(EVAL-WHEN (EVAL LOAD COMPILE)
                  (ESTABLISH-PREDICATE ',(PS-NAME PS))))))))))

```

```

(DEFUN PREDICATE-BODY (PS ARG)
  (LET ((PREDICATE (PS-PREDICATE PS))
        (TYPE (PS-TYPE PS)))
    (CASE TYPE
      (DATATYPE
       ;; for datatypes, always create a predicate. Use typep
       `(TYPEP ,ARG ',(PS-NAME PS)))
      (OTHERWISE
       ;; vectors and lists can only have a predicate if they are named
       (IF (NOT (PS-NAMED PS))
           (ERROR "The predicate ~s may not be specified for ~s because it is not :name'd" PREDICATE
                 (PS-NAME PS)))
           `(AND (TYPEP ,ARG ',(IF (EQ TYPE 'LIST)
                                   'CONS
                                   'VECTOR))
                 (EQ ,(IF (EQ TYPE 'LIST)
                          `(NTH , (PS-NAME-SLOT-POSITION PS)
                                ,ARG)
                          `(AREF ,ARG , (PS-NAME-SLOT-POSITION PS)))
                   ',(PS-NAME PS))))))

```

```

(DEFUN TYPE-EXPAND-STRUCTURE (TYPE-FORM)
  `(:DATATYPE ,(CAR TYPE-FORM)))

```

```

(DEFUN TYPE-EXPAND-NAMED-STRUCTURE (TYPE-FORM)
  `(SATISFIES ,(PS-PREDICATE (PARSED-STRUCTURE (CAR TYPE-FORM))))

```

```

(DEFUN PS-NAME-SLOT-POSITION (PS)
  "returns the offset of the name slot for ps."
  (LET* ((INCLUDE (PS-INCLUDE PS))
         (SUPER-SLOTS (AND INCLUDE (PS-ALL-SLOTS (PARSED-STRUCTURE INCLUDE))))
         (+ (PS-INITIAL-OFFSET PS)
            (LENGTH SUPER-SLOTS))))

```

```

(DEFUN DEFAULT-PREDICATE-NAME (STRUCTURE-NAME)
  (VALUES (INTERN (CONCATENATE 'STRING (STRING STRUCTURE-NAME)
                              "-P"))))

```

```

(DEFUN DEFSTRUCT-SHARED-PREDICATE-OPTIMIZER (FORM &OPTIONAL ENVIRONMENT CONTEXT)
  (XCL:DESTRUCTURING-BIND (PREDICATE OBJECT)
    FORM
    (LET ((NAME (GETHASH PREDICATE *DEFSTRUCT-INFO-CACHE*)))
      (IF (NULL NAME)
          (SETQ NAME (CACHE-PREDICATE-INFO PREDICATE)))
      (IF NAME
          `(TYPEP ,OBJECT ',NAME)
          (COMPILER:PASS))))

```

```

(DEFUN CACHE-PREDICATE-INFO (PREDICATE)
  ;; Establishes a shared a shared optimizer for a defstruct predicate
  (LET ((PS (GET-PS-FROM-PREDICATE PREDICATE T)))
    (WHEN PS

```

```
(SETF (GETHASH PREDICATE *DEFSTRUCT-INFO-CACHE*)
      (PS-NAME PS))))
```

```
(DEFCONSTANT %FUNCTION-DEFINING-FORM-KEYWORDS '(:ACCESSOR :COPIER :PREDICATE :BOA-CONSTRUCTOR
                                                :CONSTRUCTOR)
            "all the legal contexts for function-defining-form in
            defstruct")
```

:: accessors and setfs

```
(DEFUN SETF-NAME (ACCESSOR-NAME)
  "produces the name of the setf function for this accessor"
  (XCL:PACK (LIST '%SETF- ACCESSOR-NAME)))
```

```
(DEFUN ACCESSOR-BODY (SLOT ARGUMENT STRUCTURE-TYPE &OPTIONAL (NO-TYPE-CHECK NIL))
  ;; Returns a form which fetches slot from argument
  (ECASE STRUCTURE-TYPE
    (DATATYPE `(, (IF NO-TYPE-CHECK
                     'IL:FFETCHFIELD
                     'IL:FETCHFIELD)
              ', (PSLOT-FIELD-DESCRIPTOR SLOT)
              , ARGUMENT))
    (LIST `(NTH ', (PSLOT-FIELD-DESCRIPTOR SLOT)
              , ARGUMENT))
    (VECTOR `(AREF , ARGUMENT , (PSLOT-FIELD-DESCRIPTOR SLOT))))))
```

```
(DEFUN PROCESS-ACCESSORS (PS)
  (IF (NOT (EQ (PS-INLINE PS)
              :ONLY))
      (IF COMPILER::*NEW-COMPILER-IS-EXPANDING*
          '((ESTABLISH-ACCESSORS ', (PS-NAME PS))
            '(EVAL-WHEN (EVAL)
                       (ESTABLISH-ACCESSORS ', (PS-NAME PS))
            (EVAL-WHEN (LOAD)
                       ,@(DEFINE-ACCESSORS PS))))))
```

```
(DEFUN ESTABLISH-ACCESSORS (PS-NAME)
  ;; Makes a closure for every accessor
  (LET* ((PS (PARSED-STRUCTURE PS-NAME))
         (STRUCTURE-TYPE (PS-TYPE PS)))
    (MAPCAN #'(LAMBDA (SLOT)
              (LET ((ACCESSOR (PSLOT-ACCESSOR SLOT))
                    (EXPORTNAME (PS-EXPORT PS)))
                (WHEN ACCESSOR
                  (IF (OR (EQ EXPORTNAME T)
                          (MEMBER :ACCESSOR EXPORTNAME))
                      (EXPORT ACCESSOR) ; Edited by TT(13-June-90) Export Option Follow up
                      (SETF (SYMBOL-FUNCTION ACCESSOR)
                            (%MAKE-ACCESSOR-CLOSURE SLOT STRUCTURE-TYPE))))))
            (PS-ALL-SLOTS PS))))
```

```
(DEFUN DEFINE-ACCESSORS (PS)
  ;; Returns the forms that when evaluated, define the accessors
  ;; Only used by the byte compiler
  (LET ((NAME (PS-NAME PS))
        (STRUCTURE-TYPE (PS-TYPE PS)))
    ;; the arg-name must be the structure name, since it is already in the raw-accessors.
    (MAPCAN #'(LAMBDA (SLOT)
              (LET ((ACCESSOR (PSLOT-ACCESSOR SLOT))
                    (EXPORTNAME (PS-EXPORT PS)))
                (WHEN ACCESSOR
                  (IF (OR (EQ EXPORTNAME T)
                          (MEMBER :ACCESSOR EXPORTNAME))
                      (EXPORT ACCESSOR) ; Edited by TT(13-June-90) Export Option follow-up.
                      '((DEFUN ,ACCESSOR (,NAME)
                        , (ACCESSOR-BODY SLOT NAME STRUCTURE-TYPE))))))
            (PS-ALL-SLOTS PS))))
```

```
(DEFUN DEFSTRUCT-SHARED-ACCESSOR-OPTIMIZER (FORM &OPTIONAL ENVIRONMENT CONTEXT)
  (XCL:DESTRUCTURING-BIND (ACCESSOR OBJECT)
    FORM
    (LET ((SLOT-INFO (GETHASH ACCESSOR *DEFSTRUCT-INFO-CACHE*))
          (IF (NULL SLOT-INFO)
              (SETQ SLOT-INFO (CACHE-SLOT-INFO ACCESSOR)))
          (IF SLOT-INFO
```

```
(XCL:DESTRUCTURING-BIND (TYPE SLOT FAST-ACCESSORS-P)
  SLOT-INFO
  (ACCESSOR-BODY SLOT OBJECT TYPE FAST-ACCESSORS-P))
'COMPILER:PASS)))
```

```
(DEFINE-SHARED-SETF-MACRO DEFSTRUCT-SHARED-SETF-EXPANDER ACCESSOR (DATUM) (NEW-VALUE)
```

;; Shared setf expander for all defstruct slot accessors

```
(LET ((SLOT-INFO (GETHASH ACCESSOR *DEFSTRUCT-INFO-CACHE*)))
  (WHEN (NULL SLOT-INFO)
    (SETQ SLOT-INFO (CACHE-SLOT-INFO ACCESSOR)))
  (XCL:DESTRUCTURING-BIND (TYPE SLOT FAST-ACCESSORS-P)
    SLOT-INFO
    (LET ((DESCRIPTOR (PSLOT-FIELD-DESCRIPTOR SLOT)))
      (ECASE TYPE
        (DATATYPE `(, (IF FAST-ACCESSORS-P
          'IL:FREPLACEFIELD
          'IL:REPLACEFIELD)
          ,DESCRIPTOR
          ,DATUM
          ,NEW-VALUE))
        (LIST `(SETF (NTH ,DESCRIPTOR ,DATUM)
          ,NEW-VALUE))
        (VECTOR (MACROLET ((SIMPLE-P (X)
          `(OR (SYMBOLP ,X)
            (CONSTANTP ,X))))
          (IF (AND (SIMPLE-P DATUM)
            (SIMPLE-P NEW-VALUE))
            `(XCL:ASET ,NEW-VALUE ,DATUM ,DESCRIPTOR)
            (LET ((D (GENSYM))
              (V (GENSYM)))
              `(LET ((,D ,DATUM)
                (,V ,NEW-VALUE))
                  (XCL:ASET ,V ,D ,DESCRIPTOR))))))))))
```

```
(DEFUN CACHE-SLOT-INFO (ACCESSOR)
```

;;; saves the internal accessors in a hash table so that setf methods can be generated at interpret/compile time.

```
(LET* ((PS (GET-PS-FROM-ACCESSOR ACCESSOR))
  (FAST-ACCESSORS (PS-FAST-ACCESSORS PS)))
  (SETF (GETHASH ACCESSOR *DEFSTRUCT-INFO-CACHE*) ; Make a copy of the slot to keep refcounts down
    (LIST (PS-TYPE PS)
      (COPY-TREE (GET-SLOT-DESCRIPTOR-FROM-PS ACCESSOR PS))
      (AND FAST-ACCESSORS T))))
```

```
(DEFUN %MAKE-ACCESSOR-CLOSURE (SLOT STRUCTURE-TYPE)
```

```
(LET ((DESCRIPTOR (PSLOT-FIELD-DESCRIPTOR SLOT))
  (ECASE STRUCTURE-TYPE
    (DATATYPE (XCL:DESTRUCTURING-BIND (TYPENAME OFFSET FIELD-DESCRIPTOR)
      DESCRIPTOR
      (CASE FIELD-DESCRIPTOR
        ((IL:POINTER IL:FULLPOINTER IL:XPOINTER IL:FULLXPOINTER) (
          %MAKE-POINTER-ACCESSOR
          TYPENAME OFFSET))
        (IL:FLOATP (%MAKE-FLOAT-ACCESSOR TYPENAME OFFSET))
        (IL:FIXP (%MAKE-FIXP-ACCESSOR TYPENAME OFFSET))
        (OTHERWISE
          ;; Must be a bit field
          (LET* ((FIELD-TYPE (CAR FIELD-DESCRIPTOR))
            (FIELD-ARG (CDR FIELD-DESCRIPTOR))
            (SIZE (1+ (LOGAND FIELD-ARG 15)))
            (POSITION (- 16 (+ SIZE (ASH FIELD-ARG -4)))))
            (ECASE FIELD-TYPE
              (IL:BITS (IF (EQ SIZE 16)
                (%MAKE-WORD-ACCESSOR TYPENAME OFFSET)
                (%MAKE-BIT-ACCESSOR TYPENAME OFFSET POSITION SIZE))
              (IL:FLAGBITS (IF (EQ SIZE 1)
                (%MAKE-FLAG-ACCESSOR TYPENAME OFFSET POSITION)
                (ERROR "Illegal field descriptor: ~s" DESCRIPTOR))
              (IL:SIGNEDBITS (IF (EQ SIZE 16)
                (%MAKE-SMALL-FIXP-ACCESSOR TYPENAME OFFSET)
                ;; Would be better to say here "Inconvenient field descriptor"
                (ERROR "Illegal field descriptor: ~s"
                  DESCRIPTOR))))))))))
    (LIST (%MAKE-LIST-ACCESSOR DESCRIPTOR))
    (VECTOR (%MAKE-ARRAY-ACCESSOR DESCRIPTOR))))))
```

```
(DEFUN %MAKE-LIST-ACCESSOR (OFFSET)
```

```
#' (LAMBDA (LIST)
      (NTH OFFSET LIST)))
```

```
(DEFUN %MAKE-ARRAY-ACCESSOR (OFFSET)
      #' (LAMBDA (VECTOR)
            (AREF VECTOR OFFSET)))
```

```
(DEFUN %MAKE-POINTER-ACCESSOR (TYPE OFFSET)
      (IF TYPE
        #' (LAMBDA (OBJECT)
              (IF (NOT (IL:\INSTANCE-P OBJECT TYPE))
                  (ERROR "Arg not ~s: ~s" TYPE OBJECT)
                  (IL:\GETBASEPTR OBJECT OFFSET)))
          #' (LAMBDA (OBJECT)
                (IL:\GETBASEPTR OBJECT OFFSET))))
```

```
(DEFUN %MAKE-BIT-ACCESSOR (TYPE WORD-OFFSET OFFSET SIZE)
      (IF TYPE
        #' (LAMBDA (OBJECT)
              (IF (NOT (IL:\INSTANCE-P OBJECT TYPE))
                  (ERROR "Arg not ~s: ~s" TYPE OBJECT)
                  (LDB (BYTE SIZE OFFSET)
                       (IL:\GETBASE OBJECT WORD-OFFSET))))
          #' (LAMBDA (OBJECT)
                (LDB (BYTE SIZE OFFSET)
                     (IL:\GETBASE OBJECT WORD-OFFSET))))
```

```
(DEFUN %MAKE-FLAG-ACCESSOR (TYPE WORD-OFFSET OFFSET)
      (IF TYPE
        #' (LAMBDA (OBJECT)
              (IF (NOT (IL:\INSTANCE-P OBJECT TYPE))
                  (ERROR "Arg not ~s: ~s" TYPE OBJECT)
                  (NOT (EQ 0 (LDB (BYTE 1 OFFSET)
                                   (IL:\GETBASE OBJECT WORD-OFFSET))))))
          #' (LAMBDA (OBJECT)
                (NOT (EQ 0 (LDB (BYTE 1 OFFSET)
                               (IL:\GETBASE OBJECT WORD-OFFSET))))))
```

```
(DEFUN %MAKE-WORD-ACCESSOR (TYPE OFFSET)
      (IF TYPE
        #' (LAMBDA (OBJECT)
              (IF (NOT (IL:\INSTANCE-P OBJECT TYPE))
                  (ERROR "Arg not ~s: ~s" TYPE OBJECT)
                  (IL:\GETBASE OBJECT OFFSET)))
          #' (LAMBDA (OBJECT)
                (IL:\GETBASE OBJECT OFFSET)))
```

```
(DEFUN %MAKE-FIXP-ACCESSOR (TYPE OFFSET)
      (IF TYPE
        #' (LAMBDA (OBJECT)
              (IF (NOT (IL:\INSTANCE-P OBJECT TYPE))
                  (ERROR "Arg not ~s: ~s" TYPE OBJECT)
                  (IL:\GETBASEFIXP OBJECT OFFSET)))
          #' (LAMBDA (OBJECT)
                (IL:\GETBASEFIXP OBJECT OFFSET)))
```

```
(DEFUN %MAKE-SMALL-FIXP-ACCESSOR (TYPE OFFSET)
      (IF TYPE
        #' (LAMBDA (OBJECT)
              (IF (NOT (IL:\INSTANCE-P OBJECT TYPE))
                  (ERROR "Arg not ~s: ~s" TYPE OBJECT)
                  (IL:\GETBASESMALL-FIXP OBJECT OFFSET)))
          #' (LAMBDA (OBJECT)
                (IL:\GETBASESMALL-FIXP OBJECT OFFSET)))
```

```
(DEFUN %MAKE-FLOAT-ACCESSOR (TYPE OFFSET)
      (IF TYPE
        #' (LAMBDA (OBJECT)
              (IF (NOT (IL:\INSTANCE-P OBJECT TYPE))
                  (ERROR "Arg not ~s: ~s" TYPE OBJECT)
                  (IL:\GETBASEFLOATP OBJECT OFFSET)))
          #' (LAMBDA (OBJECT)
                (IL:\GETBASEFLOATP OBJECT OFFSET)))
```

;;; constructor definition code

```
(DEFUN DEFINE-CONSTRUCTORS (PS)
```

;; Returns the forms that when evaluated, define the constructors

```
(IF (NOT (PS-TEMPLATE PS))
  (LET* ((CONSTRUCTORS (PS-CONSTRUCTORS PS))
         (SLOTS (PS-ALL-SLOTS PS))
         (RESULT-ARG (PS-NAME PS))
         (ALL-BOAS? (EVERY #'BOA-CONSTRUCTOR-P CONSTRUCTORS))
         (EXPORTNAME (PS-EXPORT PS))
         (IF (OR (EQ EXPORTNAME T)
                 (MEMBER :CONSTRUCTOR EXPORTNAME))
             (EXPORT CONSTRUCTORS)
             (COND
              (ALL-BOAS?
               ;; don't bother building the arglist etc.
               (MAPCAR #'(LAMBDA (CONSTRUCTOR)
                        (DEFINE-BOA-CONSTRUCTOR CONSTRUCTOR PS))
                       CONSTRUCTORS))
              (T (LET* ((ARGUMENT-LIST (BUILD-CONSTRUCTOR-ARGLIST SLOTS))
                       (SLOT-SETFS (BUILD-CONSTRUCTOR-SLOT-SETFS SLOTS ARGUMENT-LIST PS))
                       (XCL:WITH-COLLECTION
                        (DOLIST (CONSTRUCTOR CONSTRUCTORS)
                          (XCL:COLLECT (COND
                                         ((BOA-CONSTRUCTOR-P CONSTRUCTOR)
                                          (DEFINE-BOA-CONSTRUCTOR CONSTRUCTOR PS))
                                         (T ;; keep the name of a standard constructor, if any, so that the #s form can work.
                                          (SETF (PS-STANDARD-CONSTRUCTOR PS)
                                                CONSTRUCTOR)
                                          ;; since we just built the object we're setting fields of, we don't need to type check it.
                                          `(DEFUN ,CONSTRUCTOR (&KEY ,@ARGUMENT-LIST)
                                             (LET ((,RESULT-ARG , (RAW-CONSTRUCTOR PS)))
                                                 ,@SLOT-SETFS
                                                 ,RESULT-ARG))))))))))))))
              (EXPORT CONSTRUCTORS))))))
  (EXPORT CONSTRUCTORS))
```

```
(DEFUN DEFINE-BOA-CONSTRUCTOR (NAME&ARGLIST PS)
  (LET* ((CONSTRUCTOR-NAME (CAR NAME&ARGLIST))
         (ARGLIST (CADR NAME&ARGLIST))
         (NEW-ARGUMENT-LIST (BOA-ARG-LIST-WITH-INITIAL-VALUES ARGLIST PS))
         (RESULT-ARG (PS-NAME PS))
         (SLOT-SETFS (BOA-SLOT-SETFS RESULT-ARG (ARGUMENT-NAMES NEW-ARGUMENT-LIST PS)))
         `(DEFUN ,CONSTRUCTOR-NAME ,NEW-ARGUMENT-LIST
            (LET ((,RESULT-ARG , (RAW-CONSTRUCTOR PS))
                  ,@SLOT-SETFS
                  ,RESULT-ARG))))
```

```
(DEFUN ARGUMENT-NAMES (ARG-LIST)
  (MAPCAN #'(LAMBDA (ARG)
            (COND
             ((CONSP ARG)
              (LIST ARG))
             ((MEMBER ARG LAMBDA-LIST-KEYWORDS)
              NIL)
             (T (LIST (LIST ARG :REQUIRED-ARG))))))
  ARG-LIST))
```

```
(DEFUN BOA-ARG-LIST-WITH-INITIAL-VALUES (ARG-LIST PS)
  (LET ((NEW-ARG-LIST (COPY-TREE ARG-LIST))
        (SLOTS (PS-ALL-SLOTS PS)))
    ;; for all the args from &optional up to &rest or &aux get the default value.
    (IL:FOR ARG-TAIL IL:ON (CDR (MEMBER '&OPTIONAL NEW-ARG-LIST))
      IL:DO (COND
             ((MEMBER (CAR ARG-TAIL)
                      LAMBDA-LIST-KEYWORDS)
              ;; we have found an ampersand arg, we're done the optionals.
              (RETURN))
             (T (LET ((OPTIONAL (CAR ARG-TAIL))
                      (SETF (CAR ARG-TAIL)
                            (COND
                             ((MEMBER OPTIONAL '(&REST &AUX))
                              ;; we have hit the end of the optionals, just return.
                              (RETURN))
                             (MEMBER OPTIONAL LAMBDA-LIST-KEYWORDS)
                              ;; illegal keyword here
                              (ERROR "~S cannot appear in a BOA constructor as it does in ~S." OPTIONAL
                                     ARG-LIST))
                             ((SYMBOLP OPTIONAL)
                              (LET ((INITIAL-VALUE-FORM (PSLOT-INITIAL-VALUE (FIND-SLOT OPTIONAL SLOTS))))
```

```

      (IF INITIAL-VALUE-FORM
        ` (, OPTIONAL , INITIAL-VALUE-FORM)
        ` (, OPTIONAL NIL , (IL:GENSYM))))
    ((AND (CONSP OPTIONAL)
      (CDR OPTIONAL))
      ;; already a default just leave it alone
      OPTIONAL)
    ((CONSP OPTIONAL)
      (LET ((INITIAL-VALUE-FORM (PSLOT-INITIAL-VALUE (FIND-SLOT (CAR OPTIONAL)
        SLOTS))))
        (IF INITIAL-VALUE-FORM
          ` (, (CAR OPTIONAL)
            , INITIAL-VALUE-FORM)
          ` (, (CAR OPTIONAL)
            NIL
            , (IL:GENSYM))))))))
  NEW-ARG-LIST))

```

```

(DEFUN BOA-SLOT-SETFS (RESULT-ARG SLOT-NAMES PS)
  (LET ((STRUCTURE-TYPE (PS-TYPE PS))
        (XCL:WITH-COLLECTION (LET (SLOT-PLACE SLOT-NAME SLOT-ARGUMENT)
          (DOLIST (SLOT (PS-ALL-SLOTS PS))
            (SETQ SLOT-NAME (PSLOT-NAME SLOT))
            (SETQ SLOT-PLACE (ACCESSOR-BODY SLOT RESULT-ARG STRUCTURE-TYPE T))
            (SETQ SLOT-ARGUMENT (ASSOC SLOT-NAME SLOT-NAMES :TEST #'EQ))
            (XCL:COLLECT (IF SLOT-ARGUMENT
              (LET ((SUPPLIED-P (CADDR SLOT-ARGUMENT))
                    (IF SUPPLIED-P
                      `(IF , SUPPLIED-P
                        (SETF , SLOT-PLACE , SLOT-NAME))
                      `(SETF , SLOT-PLACE , SLOT-NAME)))
                `(SETF , SLOT-PLACE , (PSLOT-INITIAL-VALUE SLOT))))))))))

```

```

(DEFUN FIND-SLOT (NAME SLOTS &OPTIONAL (DONT-ERROR NIL))
  (DOLIST (SLOT SLOTS (OR DONT-ERROR (ERROR "slot ~s not found." NAME)))
    (IF (EQ NAME (PSLOT-NAME SLOT))
      (RETURN SLOT))))

```

```

(DEFUN RAW-CONSTRUCTOR (PS)
  ;; Returns a form which will make an instance of this structure w/o initialisation
  (ECASE (PS-TYPE PS)
    (DATATYPE `(IL:NCREATE ' , (PS-NAME PS)))
    (LIST `(MAKE-LIST , (PS-NUMBER-OF-SLOTS PS)))
    (VECTOR `(MAKE-ARRAY ' (, (PS-NUMBER-OF-SLOTS PS))
      :ELEMENT-TYPE
      ' , (PS-VECTOR-TYPE PS))))))

```

```

(DEFUN BUILD-CONSTRUCTOR-ARGLIST (SLOTS)
  ;; Gathers the keywords and initial-values for (non BOA) constructors
  (MAPCAN #'(LAMBDA (SLOT)
    (LET* ((INIT-FORM (PSLOT-INITIAL-VALUE SLOT))
          (ARG-NAME (PSLOT-NAME SLOT))
          (KEYWORD-PAIR `( (, (VALUES (INTERN (SYMBOL-NAME ARG-NAME)
            ' KEYWORD))
            , (GENSYM))))
          (COND
            ((NOT (PSLOT-ACCESSOR SLOT))
              ;; this is an invisible slot (name, initial-offset, etc.) don't generate a keyword arg
              NIL)
            (INIT-FORM
              ;; specify an initial value for the keyword arg
              ` ((, KEYWORD-PAIR , INIT-FORM)))
            (T ` ((, KEYWORD-PAIR NIL , (GENSYM)))))))
    SLOTS))

```

```

(DEFUN BUILD-CONSTRUCTOR-SLOT-SETFS (SLOTS ARGUMENT-LIST PS)
  ;; Builds the setfs that initialize the slots in a constructor
  (LET ((STRUCTURE-TYPE (PS-TYPE PS))
        (OBJECT-NAME (PS-NAME PS))
        (ARGUMENT-LIST ARGUMENT-LIST))
    ;; The argument list does not have arguments for "invisible" slots.
    (MAPCAR #'(LAMBDA (SLOT)
      (COND
        ((NOT (PSLOT-ACCESSOR SLOT))
          ;; invisible slot, so generate a self to it's initial-value

```



```

` (SETF , (ACCESSOR-BODY SLOT OBJECT-NAME STRUCTURE-TYPE T)
, (PSLOT-INITIAL-VALUE SLOT))
(T (LET* ((ARGUMENT (POP ARGUMENT-LIST))
(KEYWORD-VAR-NAME (CADAR ARGUMENT))
(INITIAL-VALUE-FORM (CADR ARGUMENT)))
;; since slots can be read-only, we self the raw accessor, not the slot accessor.
;; Also, since we built the object in which we are setting fields, we use the internal-accessor without
;; typecheck
(IF INITIAL-VALUE-FORM
` (SETF , (ACCESSOR-BODY SLOT OBJECT-NAME STRUCTURE-TYPE T)
, KEYWORD-VAR-NAME)
` (IF , (CADDR ARGUMENT)
(SETF , (ACCESSOR-BODY SLOT OBJECT-NAME STRUCTURE-TYPE T)
, KEYWORD-VAR-NAME))))))
SLOTS)))

```

```

(DEFUN BOA-CONSTRUCTOR-P (CONSTRUCTOR)
;; Returns t if the constructor is a By Order of Arguments constructor
(CONSP CONSTRUCTOR))

```

```

(DEFUN DEFAULT-CONSTRUCTOR-NAME (STRUCTURE-NAME)
(VALUE (INTERN (CONCATENATE 'STRING "MAKE-" (STRING STRUCTURE-NAME))))))

```

;;; copiers

```

(DEFUN DEFINE-COPIERS (PS)
;; Returns the form that when evaluated, defines the copier
(IF (NOT (PS-TEMPLATE PS))
(LET ((COPIER (PS-COPIER PS))
(RESET-ARG 'NEW)
(FROM-ARG (PS-NAME PS)))
(IF COPIER
(MULTIPLE-VALUE-BIND (FROM-ARG-TYPE-CHECK TYPE-CHECK-SLOTS?)
(BUILD-COPIER-TYPE-CHECK PS FROM-ARG)
(LET ((SLOT-SETFS (BUILD-COPIER-SLOT-SETFS (PS-ALL-SLOTS PS)
(PS-TYPE PS)
FROM-ARG RESULT-ARG TYPE-CHECK-SLOTS?))
(EXPORTNAME (PS-EXPORT PS)))
(IF (OR (EQ EXPORTNAME T)
(MEMBER :COPIER EXPORTNAME))
(EXPORT (PS-COPIER PS))) ; Edited by TT(13-June-90) Export Option follow up
;; Since we just built the object we're setting fields of, we don't need to type check it.
`(DEFUN , (PS-COPIER PS) (, FROM-ARG)
,@FROM-ARG-TYPE-CHECK (LET ((, RESULT-ARG , (RAW-CONSTRUCTOR PS))
,@SLOT-SETFS
, RESULT-ARG))))))))))

```

```

(DEFUN BUILD-COPIER-SLOT-SETFS (SLOTS STRUCTURE-TYPE FROM-ARGUMENT TO-ARGUMENT TYPE-CHECK-SLOTS?)
"constructs the forms that copy each individual slot."
;; build a series of forms that look like
;; (setf (structure-slot to-arg) (structure-slot from-arg))
(MAPCAR #'(LAMBDA (SLOT)
` (SETF , (ACCESSOR-BODY SLOT TO-ARGUMENT STRUCTURE-TYPE T)
, (ACCESSOR-BODY SLOT FROM-ARGUMENT STRUCTURE-TYPE T)))
SLOTS))

```

```

(DEFUN BUILD-COPIER-TYPE-CHECK (PS FROM-ARG)
;; Constructs the type checking form at the beginning of the copier and decides whether individual slots need to be type-checked.
(COND
((EQ (PS-TYPE PS)
'DATATYPE)
;; If something is a datatype type check the from-arg once at the beginning. Don't check the individual accesses.
(VALUES `((CHECK-TYPE ,FROM-ARG , (PS-NAME PS)))
NIL))
((PS-PREDICATE PS)
;; if the structure has a predicate ,then call the predicate.
(VALUES `((OR (, (PS-PREDICATE PS)
, FROM-ARG)
(ERROR , (FORMAT NIL "Arg not ~s: ~~S" (PS-NAME PS))
, FROM-ARG)))
NIL))

```

```
(T ;; Otherwise, just use the type-checked slot access, so that at least the argument is assured to be a vector/list.
  (VALUES NIL T)))
```

;;; print functions

```
(DEFVAR %DEFAULT-PRINT-FUNCTION 'DEFAULT-STRUCTURE-PRINTER "print function used when none is specified in
  a defstruct")
```

;;; internal stuff.

```
(DEFSETF IL:FFETCHFIELD IL:FREPLACEFIELD)
```

;;; utilities

```
(DEFMACRO DEFSTRUCT-ASSERT-SUBTYPEP (TYPE1 TYPE2 (ERROR-STRING . ERROR-ARGS)
  &REST CERROR-ACTIONS)
```

;; Provides an interface for places where the implementor isn't sure that subtypep can be trusted

```
(LET ((ERROR-STRING (OR ERROR-STRING "~S is not a subtype of ~S"))
      (ERROR-ARGS (OR ERROR-ARGS (LIST TYPE1 TYPE2))))
  `(MULTIPLE-VALUE-BIND (SUBTYPE? CERTAIN?)
    (SUBTYPEP ,TYPE1 ,TYPE2)
    (COND
     (SUBTYPE? ; it's ok, continue
      T)
     (CERTAIN? ; subtypep says it sure, so blow up
      (ERROR ,ERROR-STRING ,@ERROR-ARGS))
     (T ; subtypep isn't sure, so raise a continuable error
      (CERROR "Assume subtypep should return t" , (FORMAT NIL "Perhaps, ~a" ERROR-STRING)
        ,@ERROR-ARGS)
      ,@CERROR-ACTIONS T))))))
```

;;; inspecting structures

```
(DEFUN STRUCTURE-OBJECT-P (OBJECT)
  (TYPEP OBJECT 'STRUCTURE-OBJECT))
```

```
(DEFUN INSPECT-STRUCTURE-OBJECT (STRUCTURE OBJECTTYPE WHERE)
  "calls the system facilities with the appropriate slots and functions."
  (IL:INSPECTW.CREATE STRUCTURE (PS-ALL-SLOTS (PARSED-STRUCTURE (TYPE-OF STRUCTURE)))
    'STRUCTURE-OBJECT-INSPECT-FETCHFN
    'STRUCTURE-OBJECT-INSPECT-STOREFN
    'STRUCTURE-OBJECT-PROPCOMMANDFN NIL NIL (LET ((XCL:*PRINT-STRUCTURE* NIL))
      (CONCATENATE 'STRING (PRINC-TO-STRING STRUCTURE)
        " Inspector"))
    NIL WHERE 'STRUCTURE-OBJECT-INSPECT-PROPPRINTFN))
```

```
(DEFUN STRUCTURE-OBJECT-INSPECT-FETCHFN (OBJECT PROPERTY)
  (IF (PSLOT-ACCESSOR PROPERTY)
    (FUNCALL (PSLOT-ACCESSOR PROPERTY)
      OBJECT)
    (IL:FETCHFIELD (PSLOT-FIELD-DESCRIPTOR PROPERTY)
      OBJECT)))
```

```
(DEFUN STRUCTURE-OBJECT-INSPECT-PROPPRINTFN (PROPERTY DATUM)
  (PSLOT-NAME PROPERTY))
```

```
(DEFUN STRUCTURE-OBJECT-INSPECT-STOREFN (OBJECT PROPERTY NEWVALUE)
  ;; this effectively does (eval '(setf ,(pslot-accessor property) object) newvalue))
  (IF (PSLOT-ACCESSOR PROPERTY)
    (EVAL `(SETF (, (PSLOT-ACCESSOR PROPERTY)
      ',OBJECT)
      ',NEWVALUE))
    (IL:REPLACEFIELD (PSLOT-FIELD-DESCRIPTOR PROPERTY)
      OBJECT NEWVALUE)))
```

```
(DEFUN STRUCTURE-OBJECT-PROPCOMMANDFN (PROPERTY DATUM INSPECTOR-WINDOW)
  (IF (AND (TYPEP DATUM 'STRUCTURE-OBJECT)
    (PSLOT-READ-ONLY PROPERTY))
    (IL:PROMPTPRINT "Can't set a read-only slot.")
    (IL:DEFAULT.INSPECTW.PROPCOMMANDFN PROPERTY DATUM INSPECTOR-WINDOW)))
```

:: Defined last so functions required to load a defstruct are loaded first

```
(DEFSTRUCT (PS (:TYPE LIST)
              :NAMED)
```

::: Contains the parsed information for a SINGLE structure type

:: most values are not defaulted here, because the defaults depend on other slot values (e.g. predicate depends on type and named.) These
:: defaults are installed in ensure-consistent-ps.

```
(NAME) ; The name of the structure
(STANDARD-CONSTRUCTOR) ; Contains the constructor to be used by the #s reader.
(ALL-SLOT-NAMES) ; The slot-name list used by the inspector.
(TYPE %DEFAULT-DEFSTRUCT-TYPE) ; Is this structure a datatype, list or vector.
(VECTOR-TYPE) ; If its a vector, this is the element-type of the vector
(INCLUDE NIL) ; The included structure, if any.
(CONC-NAME) ;
(CONSTRUCTORS %NO-CONSTRUCTOR) ; A list of the constructors for this structure. Boas have the
; argument list, not just the name.

(PREDICATE %NO-PREDICATE)
(PRINT-FUNCTION)
(COPIER %NO-COPIER)
(NAMED NIL)
(INITIAL-OFFSET 0)
(LOCAL-SLOTS NIL) ; The slot descriptors for slots present locally (not included).
(ALL-SLOTS) ; The list of slot descriptors for every slot present in an instance
; of this slot.
; Slots specified in the :include option.

(INCLUDED-SLOTS)
;; Redundant
(DOCUMENTATION-STRING)
;; Unused
(FIELD-SPECIFIERS) ; The position of each slot in the structure. For vectors and list
; structures, it is just an offset. For datatypes, it is a
; field-specifier for fetchfield.

;; Unused
(POINTER-DESCRIPTORS) ; the descriptors for all fields which the circle-printer must scan.
; It is filled in the first time it is needed.
(INLINE T) ; Flag telling whether or not functions built by defstruct are inline
; or not.
(FAST-ACCESSORS NIL) ; Flag telling whether or not accessor functions should check the
; type of the object before slot accesses.
(TEMPLATE NIL) ; As in IL:BLOCKRECORD. Implies type datatype, no copier,
; predicate or constructors, and fast accessors. No datatype is
; declared for this option.
(EXPORT NIL) ; EXPORT indicates export of Structure's functions
)
```

```
(DEFSTRUCT (PARSED-SLOT (:CONC-NAME PSLOT-)
                       (:TYPE LIST))
  "describes a single slot in a structure"
  (NAME NIL :TYPE SYMBOL)
  (INITIAL-VALUE NIL)
  (TYPE %DEFAULT-SLOT-TYPE)
  (READ-ONLY NIL)
  FIELD-DESCRIPTOR ACCESSOR)
```

:: Mapping between names of generated functions and their associated structures

```
(DEFUN STRUCTURE-FUNCTION-P (SYMBOL)
  (CATCH 'FOUND
    (MAPHASH #'(LAMBDA (KEY PS)
      (IF (OR (AND (CONSP (PS-CONSTRUCTORS PS))
                  (MEMBER SYMBOL (PS-CONSTRUCTORS PS))
                  :TEST #'EQ))
            (EQ SYMBOL (PS-PREDICATE PS))
            (EQ SYMBOL (PS-COPIER PS))
            (DOLIST (SLOT (PS-ALL-SLOTS PS))
              (IF (EQ SYMBOL (PSLOT-ACCESSOR SLOT))
                  (RETURN (PS-NAME PS))))
              (THROW 'FOUND KEY)))
      *PARSED-DEFSTRUCTS*)))
```

```
(DEFUN STRUCTURE-FUNCTIONS (STRUCTURE-NAME)
  (LET ((PS (PARSED-STRUCTURE STRUCTURE-NAME))
        (CONS (PS-CONSTRUCTORS PS))
        (CONS (LET ((PREDICATE (PS-PREDICATE PS))
                    (IF PREDICATE (LIST PREDICATE)))
                  (LET ((COPIER (PS-COPIER PS)))
```

```

      (IF COPIER (LIST COPIER)))
    ,. (MAPCAN #' (LAMBDA (SLOT)
      (LET ((ACCESSOR (PSLOT-ACCESSOR SLOT))
            (AND ACCESSOR (LIST ACCESSOR))))
        (PS-ALL-SLOTS PS))))))

```

::: Editing structures

```

(DEFUN STRUCTURES.HASDEF (NAME &OPTIONAL TYPE SOURCE SPELLFLG)
  (OR (IL:GETDEF NAME 'IL:STRUCTURES 'IL:CURRENT '(IL:NODWIM IL:NOCOPY IL:NOERROR IL:HASDEF))
      (STRUCTURE-FUNCTION-P NAME)))

```

```

(DEFUN STRUCTURES.EDITDEF (NAME TYPE SOURCE EDITCOMS OPTIONS)
  "From accessor function or structure name, edit the structure."
  (IF (PARSED-STRUCTURE NAME T)
      (IL:DEFAULT.EDITDEF NAME 'IL:STRUCTURES SOURCE EDITCOMS OPTIONS)
      (LET ((STRUCTURE-NAME (STRUCTURE-FUNCTION-P NAME)))
          (IF STRUCTURE-NAME
              (IL:DEFAULT.EDITDEF STRUCTURE-NAME 'IL:STRUCTURES SOURCE EDITCOMS OPTIONS)
              (IL:DEFAULT.EDITDEF NAME TYPE SOURCE EDITCOMS OPTIONS))))
      NAME)

```

```

(IL:FILEPKGTYPE 'IL:STRUCTURES 'IL:HASDEF 'STRUCTURES.HASDEF 'IL:EDITDEF 'STRUCTURES.EDITDEF)

```

```

(IL:ADDTOVAR IL:SHADOW-TYPES (IL:STRUCTURES IL:FNS))

```

```

(IL:DECLARE\ : IL:DCOPY IL:DONTEVAL@LOAD

```

```

(IL:ADDTOVAR IL:INSPECTMACROS ((IL:FUNCTION STRUCTURE-OBJECT-P) . INSPECT-STRUCTURE-OBJECT))
)

```

::: file properties

```

(IL:PUTPROPS IL:DEFSTRUCT IL:FILETYPE :COMPILE-FILE)

```

```

(IL:PUTPROPS IL:DEFSTRUCT IL:MAKEFILE-ENVIRONMENT (:READTABLE "XCL" :PACKAGE "LISP"))

```

```

(IL:PUTPROPS IL:DEFSTRUCT IL:COPYRIGHT ("Venue & Xerox Corporation" 1986 1987 1900 1988 1989 1990 1992 1993))

```

FUNCTION INDEX

%MAKE-ACCESSOR-CLOSURE	13	DEFSTRUCT-SHARED-SETF-EXPANDER	13
%MAKE-ARRAY-ACCESSOR	14	DUMMY-SLOT	8
%MAKE-BIT-ACCESSOR	14	ENSURE-CONSISTENT-PS	5
%MAKE-FIXP-ACCESSOR	14	ENSURE-VALID-TYPE	3
%MAKE-FLAG-ACCESSOR	14	ESTABLISH-ACCESSORS	12
%MAKE-FLOAT-ACCESSOR	14	FIND-SLOT	16
%MAKE-LIST-ACCESSOR	13	INSERT-INCLUDED-SLOT	8
%MAKE-POINTER-ACCESSOR	14	INSPECT-STRUCTURE-OBJECT	18
%MAKE-SMALL-FIXP-ACCESSOR	14	MERGE-SLOTS	8
%MAKE-WORD-ACCESSOR	14	NAME-SLOT	8
%STRUCTURE-TYPE-TO-FIELDSPEC	10	OFFSET-SLOT	8
ACCESSOR-BODY	12	PARSE-SLOT	3
ARGUMENT-NAMES	15	PREDICATE-BODY	11
ASSIGN-FIELD-DESCRIPTORS	10	PROCESS-ACCESSORS	12
ASSIGN-SLOT-ACCESSOR	3	PROCESS-TYPE	10
ASSIGN-SLOT-OFFSET	7	PS-NAME-SLOT-POSITION	11
ASSIGN-STRUCTURE-REPRESENTATION	9	PS-NUMBER-OF-SLOTS	7
BOA-ARG-LIST-WITH-INITIAL-VALUES	15	PS-TYPE-SPECIFIER	7
BOA-CONSTRUCTOR-P	17	RAW-CONSTRUCTOR	16
BOA-SLOT-SETFS	16	RECORD-DOCUMENTATION	3
BUILD-CONSTRUCTOR-ARGLIST	16	REMOVE-DOCUMENTATION	3
BUILD-CONSTRUCTOR-SLOT-SETFS	16	RESOLVE-SLOTS	7
BUILD-COPIER-SLOT-SETFS	17	SETF-NAME	12
BUILD-COPIER-TYPE-CHECK	17	STRUCTURE-FUNCTION-P	19
CACHE-PREDICATE-INFO	11	STRUCTURE-FUNCTIONS	19
CACHE-SLOT-INFO	13	STRUCTURE-OBJECT-INSPECT-FETCHFN	18
COERCE-TYPE	9	STRUCTURE-OBJECT-INSPECT-PROPPRINTFN	18
DEFAULT-CONSTRUCTOR-NAME	17	STRUCTURE-OBJECT-INSPECT-STOREFN	18
DEFAULT-PREDICATE-NAME	11	STRUCTURE-OBJECT-P	18
DEFINE-ACCESSORS	12	STRUCTURE-OBJECT-PROPCOMMANDFN	18
DEFINE-BOA-CONSTRUCTOR	15	STRUCTURE-POINTER-SLOTS	10
DEFINE-CONSTRUCTORS	14	STRUCTURES.EDITDEF	20
DEFINE-COPIERS	17	STRUCTURES.HASDEF	20
DEFSTRUCT-PARSE-OPTIONS	4	TYPE-EXPAND-NAMED-STRUCTURE	11
DEFSTRUCT-SHARED-ACCESSOR-OPTIMIZER	12	TYPE-EXPAND-STRUCTURE	11
DEFSTRUCT-SHARED-PREDICATE-OPTIMIZER	11		

VARIABLE INDEX

%DEFAULT-DEFSTRUCT-TYPE	2	%DEFSTRUCT-CONSP-OPTIONS	3	IL:INSPECTMACROS	20
%DEFAULT-PRINT-FUNCTION	18	%DEFSTRUCT-EXPORT-OPTIONS	3	IL:SHADOW-TYPES	20
%DEFAULT-SLOT-TYPE	3	%DEFSTRUCT-OPTIONS	3		

CONSTANT INDEX

%DEFAULT-STRUCTURE-INCLUDE	3	%NO-CONSTRUCTOR	3	%NO-PREDICATE	3
%FUNCTION-DEFINING-FORM-KEYWORDS	12	%NO-COPIER	3		

STRUCTURE INDEX

PARSED-SLOT	19	PS	19
-------------	----	----	----

PROPERTY INDEX

IL:DEFSTRUCT	20
--------------	----

DEFINE-TYPE INDEX

IL:STRUCTURES	2
---------------	---

MACRO INDEX

DEFSTRUCT-ASSERT-SUBTYPEP	18
---------------------------	----

SETF INDEX

IL:FFETCHFIELD	18
----------------	----

```
{MEDLEY}<sources>DEFSTRUCT.;1
```

DEFINER INDEX

DEFSTRUCT2
