File created: `14-Sep-89 10:03:02` **{DSK}/python2/aria/migration/interlisp/IL-RECORD.;2**

changes to: `(IL:FUNCTIONS MAKE-RECORD-ACCESSORS |fetch| |replace| |DO-create|)`

previous date: `2-Mar-89 13:12:40 {DSK}/users/eweaver/convert/IL-RECORD.;4`

Read Table: `XCL`

Package: `IL-CONVERT`

Format: `XCCS`


```
(IL:RPAQQ IL:IL-RECORDCOMS
          (
```

#|chapter 8|#

```
(IL:VARIABLES *RECORD-TYPES*)
(IL:FUNCTIONS ADD-EXPORTS ASSOCRECORD PROPRECORD ATOMRECORD BLOCKRECORD)
(IL:FUNCTIONS ARRAYRECORD DEFINE-ARRAYRECORD-STRUCTURE)
                                                        ; ^'(arrayrecord foo (a b c) b _ 3)
(IL:FUNCTIONS INTERLISP-COMMENT-P)
(IL:FUNCTIONS RECORD)
(IL:FUNCTIONS TYPERECORD FLATTEN MAKE-RECORD-ACCESSORS DEFINE-RECORD-STRUCTURE)
                                                        ; ^'(record foo (a b . c) b _ 3)

;;
;; ; this version defines a defstruct which is not really the same
;; ; as the IL record type.
;; (defun
;;   define-record-structure (record-name record-fields named record-tail)
;;   (let* ((name-string (symbol-name record-name))
;;          (struct-name (intern name-string))
;;          (*current-record-name* record-name)
;;          (slots nil))
;;     (declare (special *current-record-name*))
;;     (setq record-fields (make-true-list record-fields))
;;     (do ((fields record-fields (rest fields))
;;          field)
;;         ((null fields) (setq slots (nreverse slots)))
;;       (setq field (first fields))
;;       (cond
;;         ((null field )
;;          (warn "NIL as record field name not supported"))
;;         ((atom field) (push field slots))
;;         ((eq (first field) '*)) ;Ignore comments
;;         (t (setq slots (append (reverse (flatten field)) slots)))))
;;     (setf (gethash struct-name *record-types*) slots)
;;     (multiple-value-bind
;;      (record-tail-forms record-tail-inits)
;;      (process-record-tail record-tail)
;;      (add-exports
;;       `((defstruct
;;           ,struct-name
;;           (:type list)
;;           (:named ,named)
;;           ,@(mapcar
;;             #'(lambda (slot &aux pair)
;;                       (if (setq pair (assoc slot record-tail-inits))
;;                           `(,slot ,(cdr pair))
;;                         slot))
;;             slots))
;;       ,@record-tail-forms)))))

;; Returns two values: a list of forms to be generated, and a list of (slot . init-form) pairs.
(IL:FUNCTIONS PROCESS-RECORD-TAIL)

;; Define user-created access functions.  It doesn't matter if these fields are part of the structure or not.  If so, they will redefine the access
;; functions created by defstruct.
(IL:FUNCTIONS ACCESSFNS)

;;  (convert '(accessfns pilotbbt ((pbtsource foo1 foo2))))
(IL:FUNCTIONS DATATYPE FIELD-TO-SLOT-TYPE /DECLAREDATATYPE FIND-RECORD-TYPE FIND-RECORD-FIELDS
       |fetch| |replace| TYPE? |create| |DO-create|)
(IL:P (IL-COPYCONV |fetch| FETCH)
      (IL-COPYCONV |fetch| |ffetch|)
      (IL-COPYCONV |ffetch| FFETCH)
      (IL-COPYCONV |replace| REPLACE)
      (IL-COPYCONV |replace| |freplace|)
      (IL-COPYCONV |freplace| FREPLACE)
      (IL-COPYCONV TYPE? |type?|)
      (IL-COPYCONV |create| CREATE))
(IL:PROP (IL:MAKEFILE-ENVIRONMENT IL:FILETYPE)
       IL:IL-RECORD)))
```

#|chapter 8|#

```
(DEFVAR *RECORD-TYPES* (MAKE-HASH-TABLE :SIZE 100))

(DEFUN ADD-EXPORTS (FORMS &AUX (EXPORT-LIST NIL))
   (DOLIST (FORM FORMS)
       (AND (CONSP FORM)
            (MEMBER (FIRST FORM)
                   '(DEFUN DEFMACRO)
                   :TEST
                   #'EQ)
            (PUSH (SECOND FORM)
                  EXPORT-LIST)))
   (IF EXPORT-LIST
       `(PROGN (EXPORT ',(REVERSE EXPORT-LIST))
               ,@FORMS)
       (PROGN-IF-NEEDED FORMS)))

(IL-DEFCONV ASSOCRECORD (RECORD-NAME RECORD-FIELDS &REST RECORD-TAIL)
                        (DECLARE (IGNORE RECORD-NAME RECORD-FIELDS RECORD-TAIL))
                        (WARN "ASSOCRECORD not supported")
                        ;;
                        ;; (setf
                        ;;  (gethash record-name *record-types*)
                        ;;  (mapcar #'car record-fields))
                        ;; (process-record-tail record-tail)


                        )

(IL-DEFCONV PROPRECORD (RECORD-NAME RECORD-FIELDS &REST RECORD-TAIL)
                       (DECLARE (IGNORE RECORD-NAME RECORD-FIELDS RECORD-TAIL))
                       (WARN "PROPRECORD not supported")
                       ;;
                       ;; (setf
                       ;;  (gethash record-name *record-types*)
                       ;;  (do ((fields record-fields (rest (rest fields)))
                       ;;       (slots nil))
                       ;;      ((endp fields) (nreverse slots))
                       ;;      (push (first fields) slots))
                       ;;  (process-record-tail record-tail))


                       )

(IL-DEFCONV ATOMRECORD (RECORD-NAME RECORD-FIELDS &REST RECORD-TAIL)
                       (DECLARE (IGNORE RECORD-NAME RECORD-FIELDS RECORD-TAIL))
                       (WARN "ATOMRECORD not supported"))

(IL-DEFCONV BLOCKRECORD (RECORD-NAME RECORD-FIELDS &REST RECORD-TAIL)
                        (DECLARE (IGNORE RECORD-TAIL))
                        (DECLARE (SPECIAL *ADD-TO-RECORD-DEFN*))
                        (WARN "BLOCKRECORD not supported")
                        (DO ((FIELDS RECORD-FIELDS (REST FIELDS))
                             (SLOTS NIL)
                             FIELD)
                            ((ENDP FIELDS)
                             (SETF (GETHASH RECORD-NAME *RECORD-TYPES*)
                                   (IF (BOUNDP '*ADD-TO-RECORD-DEFN*)
                                       (APPEND (NREVERSE SLOTS)
                                               (GETHASH RECORD-NAME *RECORD-TYPES*))
                                       (NREVERSE SLOTS))))
                          (SETQ FIELD (FIRST FIELDS))
                          (WHEN (CONSP FIELD)
                              (SETQ FIELD (FIRST FIELD)))
                          (WHEN (AND FIELD (NOT (INTEGERP FIELD)))
                                (PUSH FIELD SLOTS)))
                        NIL)

(IL-DEFCONV ARRAYRECORD (RECORD-NAME RECORD-FIELDS &REST RECORD-TAIL)
                        (DEFINE-ARRAYRECORD-STRUCTURE RECORD-NAME RECORD-FIELDS RECORD-TAIL))


(DEFUN DEFINE-ARRAYRECORD-STRUCTURE (RECORD-NAME RECORD-FIELDS RECORD-TAIL)
   (LET
    ((*CURRENT-RECORD-NAME* RECORD-NAME))
    (DECLARE (SPECIAL *CURRENT-RECORD-NAME*))
    (MULTIPLE-VALUE-BIND (RECORD-TAIL-FORMS RECORD-TAIL-INITS)
        (PROCESS-RECORD-TAIL RECORD-TAIL)
      (LET ((NAME-STRING (SYMBOL-NAME RECORD-NAME))
            (FIELD-FNS NIL)
            (INITS NIL)
            (KEYS NIL)
```

```
           CREATE-FN
           (LENGTH 0))
          (DO ((I 0 (1+ I))
               (FIELDS RECORD-FIELDS (REST FIELDS))
               FIELD)
              ((ENDP FIELDS)
               (SETQ FIELD-FNS (NREVERSE FIELD-FNS))
               (SETQ INITS (NREVERSE INITS))
               (SETQ KEYS (NREVERSE KEYS)))
            ;; Define accessor functions.  We don't need to define
            ;; setf methods because the accessors are actually
            ;; macros which generate calls to svref, and setf          ; already knows how to handle svref.
            (SETQ FIELD (FIRST FIELDS))
            (INCF LENGTH)
            (COND
               ((INTEGERP FIELD)
                (INCF I (1- FIELD))
                (INCF LENGTH (1- FIELD)))
               ((NULL FIELD))
               (T (PUSH `(DEFMACRO ,(INTERN (CONCATENATE 'STRING NAME-STRING "-" (SYMBOL-NAME FIELD))) (X)
                           ,(MAKE-BQ `(SVREF ,(MAKE-MACRO-ARG :ELEMENT 'X)
                                             ,I)))
                        FIELD-FNS)
                  (LET ((SVAR (INTERN (CONCATENATE 'STRING (SYMBOL-NAME FIELD)
                                                   "-SET"))))
                       (PUSH `(WHEN ,SVAR
                                    (SETF (SVREF $X$ ,I)
                                          ,FIELD))
                             INITS)
                       (PUSH `(,FIELD ,(CDR (ASSOC FIELD RECORD-TAIL-INITS))
                                      ,SVAR)
                             KEYS)))))
          (SETQ CREATE-FN `(DEFUN ,(INTERN (CONCATENATE 'STRING "MAKE-" NAME-STRING)) (&KEY ,@KEYS)
                                  (LET (($X$)
                                        (MAKE-ARRAY ,LENGTH))
                                       ,@INITS $X$)))
          (ADD-EXPORTS `(,CREATE-FN ,@FIELD-FNS ,@RECORD-TAIL-FORMS))))))

;; ^'(arrayrecord foo (a b c) b _ 3)

(DEFUN INTERLISP-COMMENT-P (X)
   (AND (CONSP X)
        (EQ (FIRST X)
            '*)))

(IL-DEFCONV RECORD (&REST ARGS)
                   (SETQ ARGS (REMOVE-IF #'INTERLISP-COMMENT-P ARGS))
                   (DEFINE-RECORD-STRUCTURE (FIRST ARGS)
                          (SECOND ARGS)
                          NIL
                          (REST (REST ARGS))))

(IL-DEFCONV TYPERECORD (&REST ARGS)
                       (SETQ ARGS (REMOVE-IF #'INTERLISP-COMMENT-P ARGS))
                       (DEFINE-RECORD-STRUCTURE (FIRST ARGS)
                              (SECOND ARGS)
                              T
                              (REST (REST ARGS))))


(DEFUN FLATTEN (X)
   (COND
      ((CONSP X)
       (APPEND (FLATTEN (CAR X))
               (FLATTEN (CDR X))))
      ((NULL X)
       NIL)
      (T (CONS X NIL))))


(DEFUN MAKE-RECORD-ACCESSORS (RECORD-NAME TREE PATH)
   (COND
      ((NULL TREE)
       NIL)
      ((ATOM TREE)
       (LET ((ACCESSOR-NAME (INTERN (CONCATENATE 'STRING RECORD-NAME "-" (SYMBOL-NAME TREE)))))
            `((DEFSETF ,ACCESSOR-NAME (X) (VAL)
                 (LIST 'SETF ,(MAKE-BQ (SUBST (MAKE-MACRO-ARG :ELEMENT 'X)
                                              T PATH :TEST #'EQ))
                       VAL))
              (DEFMACRO ,ACCESSOR-NAME (X)
                 ,(MAKE-BQ (SUBST (MAKE-MACRO-ARG :ELEMENT 'X)
                                  T PATH :TEST #'EQ))))))
      ((EQ (CAR TREE)
```

```
           '*)
       NIL)
      (T (APPEND (MAKE-RECORD-ACCESSORS RECORD-NAME (CAR TREE)
                        `(CAR ,PATH))
                 (MAKE-RECORD-ACCESSORS RECORD-NAME (CDR TREE)
                        `(CDR ,PATH))))))


(DEFUN DEFINE-RECORD-STRUCTURE (RECORD-NAME RECORD-FIELDS NAMED RECORD-TAIL)
   (LET*
    ((NAME-STRING (SYMBOL-NAME RECORD-NAME))
     (STRUCT-NAME (INTERN NAME-STRING))
     (*CURRENT-RECORD-NAME* RECORD-NAME)
     (SLOTS (REMOVE-IF #'NULL (FLATTEN RECORD-FIELDS)))
     (ACCESSORS (MAKE-RECORD-ACCESSORS NAME-STRING RECORD-FIELDS (IF NAMED
                                                                    '(CDR T)
                                                                    T))))
    (DECLARE (SPECIAL *CURRENT-RECORD-NAME*))
    (SETF (GETHASH STRUCT-NAME *RECORD-TYPES*)
          SLOTS)
    (MULTIPLE-VALUE-BIND (RECORD-TAIL-FORMS RECORD-TAIL-INITS)
        (PROCESS-RECORD-TAIL RECORD-TAIL)
      (ADD-EXPORTS
       `((DEFUN ,(INTERN (CONCATENATE 'STRING "MAKE-" NAME-STRING)) (&KEY
                                                                    ,@(MAPCAR
                                                                       #'(LAMBDA
                                                                          (SLOT &AUX PAIR)
                                                                          (IF (SETQ PAIR (ASSOC SLOT
                                                                                                RECORD-TAIL-INITS
                                                                                                :TEST
                                                                                                #'EQ))
                                                                              (LIST SLOT (CDR PAIR))
                                                                              SLOT))
                                                                       SLOTS))
           ,(MAKE-BQ (LET ((FORM (SUBLIS (MAPCAR #'(LAMBDA (SLOT)
                                                   (CONS SLOT (MAKE-MACRO-ARG :ELEMENT SLOT)))
                                         SLOTS)
                                 RECORD-FIELDS)))
                       (IF NAMED
                           (CONS RECORD-NAME FORM)
                           FORM))))
         (DEFMACRO ,(INTERN (CONCATENATE 'STRING "COPY-" NAME-STRING)) (X)
            ,(MAKE-BQ `(COPY-TREE ,(MAKE-MACRO-ARG :ELEMENT 'X))))
         ,@ACCESSORS
         ,@RECORD-TAIL-FORMS)))))

;;  ^'(record foo (a b . c) b _ 3)
;;
;; ; this version defines a defstruct which is not really the same
;; ; as the IL record type.
;; (defun
;;  define-record-structure (record-name record-fields named record-tail)
;;  (let* ((name-string (symbol-name record-name))
;;         (struct-name (intern name-string))
;;         (*current-record-name* record-name)
;;         (slots nil))
;;   (declare (special *current-record-name*))
;;   (setq record-fields (make-true-list record-fields))
;;   (do ((fields record-fields (rest fields))
;;        field)
;;       ((null fields) (setq slots (nreverse slots)))
;;       (setq field (first fields))
;;       (cond
;;         ((null field )
;;          (warn "NIL as record field name not supported"))
;;         ((atom field) (push field slots))
;;         ((eq (first field) '*)) ;Ignore comments
;;         (t (setq slots (append (reverse (flatten field)) slots)))))
;;   (setf (gethash struct-name *record-types*) slots)
;;   (multiple-value-bind
;;    (record-tail-forms record-tail-inits)
;;    (process-record-tail record-tail)
;;    (add-exports
;;     '((defstruct
;;         ,struct-name
;;         (:type list)
;;         (:named ,named)
;;         ,@(mapcar
;;            #'(lambda (slot &aux pair)
;;                (if (setq pair (assoc slot record-tail-inits))
;;                    `(,slot ,(cdr pair))
;;                    slot))
;;            slots))
;;       ,@record-tail-forms)))))

;; Returns two values: a list of forms to be generated, and a list of (slot . init-form) pairs.
```

```
(DEFUN PROCESS-RECORD-TAIL (RECORD-TAIL)
   (DECLARE (SPECIAL *CURRENT-RECORD-NAME*))
   (DO ((SPECS RECORD-TAIL (REST SPECS))
        SPEC
        (FORMS NIL)
        (INITS NIL))
       ((ENDP SPECS)
        (VALUES FORMS (REVERSE INITS)))
     (COND
        ((AND (ATOM (FIRST SPECS))
              (REST SPECS)
              (EQ (SECOND SPECS)
                  'IL:_))
         (IF (EQ *CURRENT-RECORD-NAME* (FIRST SPECS))
             (WARN "implicit CREATE record spec (by assignment to record name) not supported")
             (PUSH (CONS (FIRST SPECS)
                         (CONVERT (THIRD SPECS)))
                   INITS))
         ;; A "field-name _ form" spec is not a list -- it is
         ;; three separate entries in the record-tail.
         (POP SPECS)
         (POP SPECS))
        (T                                                    ; All others are lists.
           (SETQ SPEC (FIRST SPECS))
           (CASE (FIRST SPEC)
               ((IL:CREATE IL:INIT IL:SUBRECORD IL:SYSTEM) (WARN "~:@(~s~) record spec not supported"
                                                                 (FIRST SPEC)))
               (IL:TYPE? (PUSH `(DEFUN ,(INTERN (CONCATENATE 'STRING (SYMBOL-NAME *CURRENT-RECORD-NAME*)
                                                             "-P")) (DATUM)
                                  (LET ((*LOCALS* (ACONS 'DATUM :LOCAL *LOCALS*)))
                                       ,@(MAPCONVERT (REST SPEC))))
                               FORMS))
               ((IL:ACCESSFNS IL:BLOCKRECORD) (LET ((*ADD-TO-RECORD-DEFN* T))
                                                   (DECLARE (SPECIAL *ADD-TO-RECORD-DEFN*))
                                                   (SETQ FORMS (APPEND FORMS (LIST (CONVERT SPEC))))))
               (T (WARN "unknown record spec  ~s ignored" SPEC)))))))
```

;; Define user-created access functions. It doesn't matter if these fields are part of the structure or not. If so, they will redefine the access functions
;; created by defstruct.

```
(IL-DEFCONV ACCESSFNS (RECORD-NAME &OPTIONAL RECORD-FIELDS &REST RECORD-TAIL)
                      (DECLARE (SPECIAL *CURRENT-RECORD-NAME*))
                      (DECLARE (SPECIAL *LOCALS*))
                      ;; The manual says the record name is the first argument, but it appears that sometimes it is missing when this is a
                      ;; subdeclaration, so we get it from a special variable which is set while processing the main declaration.
                      (UNLESS (ATOM RECORD-NAME)
                              (SETQ RECORD-FIELDS RECORD-NAME RECORD-NAME *CURRENT-RECORD-NAME*))
                      (WHEN)
                      (DO ((FORMS NIL)
                           FIELD FIELD-NAME ACCESSOR-NAME (FIELDS (IF (AND (= (LENGTH RECORD-FIELDS)
                                                                              2)
                                                                           (ATOM (FIRST RECORD-FIELDS)))
                                                                      ;; Pidgin single accessfn declaration...
                                                                      (LIST RECORD-FIELDS)
                                                                      RECORD-FIELDS)
                                                                  (REST FIELDS)))
                          ((ENDP FIELDS)
                           (ADD-EXPORTS (REVERSE FORMS)))
                        (SETQ FIELD (FIRST FIELDS))
                        (SETQ FIELD-NAME (POP FIELD))
                        (SETQ ACCESSOR-NAME (INTERN (CONCATENATE 'STRING (SYMBOL-NAME RECORD-NAME)
                                                                 "_"
                                                                 (SYMBOL-NAME FIELD-NAME))))
                                                                 ; Define the accessor function
                        (WHEN FIELD
                            ;; Also remember that we know about this field
                            (PUSH FIELD-NAME (GETHASH RECORD-NAME *RECORD-TYPES*))
                            (PUSH `(DEFUN ,ACCESSOR-NAME (DATUM)
                                      ,(LET ((*LOCALS* (ACONS 'DATUM :LOCAL *LOCALS*)))
                                            (CONVERT (POP FIELD))))
                                  FORMS)                     ; Define the function to set a new value
                            (WHEN FIELD
                                (PUSH `(DEFSETF ,ACCESSOR-NAME (DATUM) (NEWVALUE)
                                          ,(LET ((*LOCALS* (ACONS 'NEWVALUE :LOCAL (ACONS 'DATUM :LOCAL
                                                                                          *LOCALS*))))
                                                (CONVERT (POP FIELD))))
                                      FORMS)))))
```

;; (convert '(accessfns pilotbbt ((pbtsource foo1 foo2))))

```
(IL-DEFCONV DATATYPE (RECORD-NAME RECORD-FIELDS &REST RECORD-TAIL)
                     (LET*
                      ((NAME-STRING (SYMBOL-NAME RECORD-NAME))
                       (STRUCT-NAME (INTERN NAME-STRING))
                       (*CURRENT-RECORD-NAME* RECORD-NAME)
                       RECORD-TAIL-FORMS RECORD-TAIL-INITS (SLOTS NIL)
                       (SLOT-DEFNS NIL)
                       (FIELD-TYPES NIL))
                      (DECLARE (SPECIAL *CURRENT-RECORD-NAME*))
                      (DO ((FIELDS RECORD-FIELDS (REST FIELDS))
                           SLOT-NAME FIELD-TYPE FIELD)
                          ((ENDP FIELDS)
                           (SETQ SLOTS (NREVERSE SLOTS)))
                        (SETQ FIELD (FIRST FIELDS))
                        (SETQ SLOT-NAME (COND
                                          ((CONSP FIELD)
                                           (CASE (FIRST FIELD)
                                                 ((NIL)
                                                     ;; Some code has field specs like "(nil 5 word))"
                                                     (WARN "record spec ~s ignored -- NIL not allowed as field
                                                           name" FIELD)
                                                     NIL)
                                                 (IL:* NIL)    ; Ignore comments
                                                 (T (SETQ FIELD-TYPE (REST FIELD))
                                                    (FIRST FIELD))))
                                          (T (SETQ FIELD-TYPE NIL)
                                             FIELD)))
                        (WHEN SLOT-NAME
                              (PUSH SLOT-NAME SLOTS)
                              (PUSH FIELD-TYPE FIELD-TYPES)))
                      ;; Have to set the field names defined here before calling
                      ;; process-record-tail since it will add to them.
                      (SETF (GETHASH STRUCT-NAME *RECORD-TYPES*)
                            SLOTS)
                      (MULTIPLE-VALUE-SETQ (RECORD-TAIL-FORMS RECORD-TAIL-INITS)
                             (PROCESS-RECORD-TAIL RECORD-TAIL))
                      ;; This could be changed to a mapcar.  Previous definitions of il-defconv
                      ;; for some reason did not correctly handle lambda's.
                      (DO ((SLOTS SLOTS (REST SLOTS))
                           (FIELD-TYPES FIELD-TYPES (REST FIELD-TYPES))
                           SLOT-NAME FIELD-TYPE)
                          ((ENDP SLOTS)
                           (SETQ SLOT-DEFNS (NREVERSE SLOT-DEFNS)))
                        (SETQ SLOT-NAME (FIRST SLOTS)
                              FIELD-TYPE
                              (FIRST FIELD-TYPES))
                        (PUSH `(,SLOT-NAME ,(CDR (ASSOC SLOT-NAME RECORD-TAIL-INITS))
                                           :TYPE
                                           ,(FIELD-TO-SLOT-TYPE FIELD-TYPE SLOT-NAME))
                              SLOT-DEFNS))
                      (LET
                       ((NAME-STRING (SYMBOL-NAME STRUCT-NAME)))
                       (PROGN-IF-NEEDED
                        `((EXPORT ',(,(INTERN (CONCATENATE 'STRING "MAKE-" NAME-STRING))
                                     ,(INTERN (CONCATENATE 'STRING "COPY-" NAME-STRING))
                                     ,(INTERN (CONCATENATE 'STRING NAME-STRING "-P"))
                                     ,@(MAPCAR #'(LAMBDA (SLOT)
                                                         (INTERN (CONCATENATE 'STRING NAME-STRING "-"
                                                                              (SYMBOL-NAME SLOT))))
                                               SLOTS)))
                          (DEFSTRUCT ,STRUCT-NAME
                             ,@SLOT-DEFNS)
                          ,@RECORD-TAIL-FORMS)))))

(DEFUN FIELD-TO-SLOT-TYPE (TYPE &OPTIONAL SLOT-NAME)
   (IF (NULL TYPE)
       T
       (CASE (FIRST TYPE)
           (INTEGER 'INTEGER)
           ((IL:FIXP IL:SIGNEDWORD) 'FIXNUM)
           ((IL:FLOATING IL:FLOATP) 'FLOAT)
           (IL:FLAG '(OR NIL T))
           (IL:BITS (IF (<= (1- (EXPT 2 (SECOND TYPE)))
                            MOST-POSITIVE-FIXNUM)
                        'FIXNUM
                        'INTEGER))
           (BYTE 'FIXNUM)
           (IL:WORD 'FIXNUM)
           ((IL:POINTER IL:XPOINTER IL:FULLPOINTER IL:FULLXPOINTER) T)
           (T (WARN "Unknown type spec ~:@(~a~)~:[~; for slot ~:*~:@(~a~)~]" (FIRST TYPE)
                    SLOT-NAME)
              T))))
```

```
(IL-DEFCONV /DECLAREDATATYPE (&REST ARGS)
                             (WARN "/DECLAREDATATYPE ignored")
                             NIL)


(DEFUN FIND-RECORD-TYPE (FIELDNAME)
   (LET ((RECORD-TYPES NIL))
        (MAPHASH #'(LAMBDA (RECORD-NAME FIELDS)
                           (WHEN (MEMBER FIELDNAME FIELDS :TEST #'EQ)
                                 (PUSH RECORD-NAME RECORD-TYPES)))
                 *RECORD-TYPES*)
        (CASE (LENGTH RECORD-TYPES)
            (0 (WARN "no record is defined with a field named ~s, using a dummy function XXXXX-~a" FIELDNAME
                     FIELDNAME)
               'XXXXX)
            (1 (CAR RECORD-TYPES))
            (T (CERROR "use ~a" "~*multiple record types have a field named ~s: ~s" (CAR RECORD-TYPES)
                       FIELDNAME RECORD-TYPES)
               (CAR RECORD-TYPES)))))


(DEFUN FIND-RECORD-FIELDS (RECORD-TYPE)
   (MULTIPLE-VALUE-BIND (RECORD FOUND)
       (GETHASH RECORD-TYPE *RECORD-TYPES*)
     (IF FOUND
         RECORD
         (PROGN (WARN "no record type ~a, initializations may not be done" RECORD-TYPE)
                NIL))))

(IL-DEFCONV |fetch| (FIELD-NAME OF &OPTIONAL X &AUX RECORD-TYPE)
                    (DECLARE (SPECIAL IL:USERRECLST))
                    (WHEN (NOT (STRING-EQUAL OF "of"))
                          (SETQ X OF))
                    (IF (CONSP FIELD-NAME)
                        (SETQ RECORD-TYPE (FIRST FIELD-NAME)
                              FIELD-NAME
                              (SECOND FIELD-NAME))
                        (LET ((M (IL:\\RECORDBLOCK/RECFIELDLOOK IL:USERRECLST FIELD-NAME)))
                             (UNLESS M (WARN "no record is defined with a field named ~s, using a dummy function
                                             XXXXX-~a" FIELD-NAME FIELD-NAME))
                             (UNLESS (NULL (CDR M))
                                    (ERROR "More than one record with ~:@(~a~)." FIELD-NAME))
                             (SETQ RECORD-TYPE (IF (NULL M)
                                                   'XXXXX
                                                   (SECOND (FIRST M))))))
                    `(,(INTERN (CONCATENATE 'STRING (SYMBOL-NAME RECORD-TYPE)
                                            "-"
                                            (SYMBOL-NAME FIELD-NAME)))
                      ,(CONVERT X)))

(IL-DEFCONV |replace| (FIELD-NAME OF X WITH Y &AUX RECORD-TYPE)
                      (COND
                         ((NOT (STRING-EQUAL OF "OF"))
                          (CERROR "Skip this form" "Missing |of| in |replace|")
                          *CURRENT-FORM*)
                         ((NOT (STRING-EQUAL WITH "WITH"))
                          (CERROR "Skip this form" "Missing |with| in |replace|")
                          *CURRENT-FORM*)
                         (T (IF (CONSP FIELD-NAME)
                                (SETQ RECORD-TYPE (FIRST FIELD-NAME)
                                      FIELD-NAME
                                      (SECOND FIELD-NAME))
                                (LET ((M (IL:\\RECORDBLOCK/ACCESSDEF FIELD-NAME)))
                                     (UNLESS M (WARN "no record is defined with a field named ~s, using a dummy
                                                     function XXXXX-~a" FIELD-NAME FIELD-NAME))
                                     (UNLESS (NULL (CDR M))
                                            (ERROR "More than one record with ~:@(~a~)." FIELD-NAME))
                                     (SETQ RECORD-TYPE (IF (NULL M)
                                                           'XXXXX
                                                           (SECOND (FIRST M))))))
                            `(SETF (,(INTERN (CONCATENATE 'STRING (SYMBOL-NAME RECORD-TYPE)
                                                          "-"
                                                          (SYMBOL-NAME FIELD-NAME)))
                                    ,(CONVERT X))
                                   ,(CONVERT Y)))))

(IL-DEFCONV TYPE? (RECORD-NAME FORM)
                  `(,(INTERN (CONCATENATE 'STRING (SYMBOL-NAME RECORD-NAME)
                                          "-P"))
                    ,(CONVERT FORM)))

(IL-DEFCONV |create| (RECORD-NAME &REST ASSIGNMENTS)
                     (|DO-create| RECORD-NAME ASSIGNMENTS))


(DEFUN |DO-create| (RECORD-NAME ASSIGNMENTS)
```

```
   (LET
    ((NAME-STRING (SYMBOL-NAME RECORD-NAME))
     (INITS NIL)
     (SMASHING NIL)
     (USING NIL)
     (VAR (MAKE-FAKE-SYMBOL (STRING (GENSYM "G")))))
    (DO ((ASSIGNMENTS ASSIGNMENTS (REST ASSIGNMENTS)))
        ((ENDP ASSIGNMENTS)
         (SETQ INITS (REVERSE INITS)))
      (COND
         ((AND (CONSP (FIRST ASSIGNMENTS))
               (STRING-EQUAL (CAAR ASSIGNMENTS)
                      '"*")))
         ((AND (SYMBOLP (SECOND ASSIGNMENTS))
               (STRING-EQUAL (SECOND ASSIGNMENTS)
                      "_"))
          (PUSH (CONS (FIRST ASSIGNMENTS)
                      (CONVERT (THIRD ASSIGNMENTS)))
                INITS)
          (SETQ ASSIGNMENTS (CDDR ASSIGNMENTS)))
         (T (CASE (FIRST ASSIGNMENTS)
                ((IL:USING IL:|using|) (SETQ USING (CONVERT (SECOND ASSIGNMENTS))))
                ((IL:COPYING IL:|copying|) (WARN "COPYING assignment not supported"))
                ((IL:REUSING IL:|reusing|) (WARN "REUSING assignment not supported"))
                ((IL:SMASHING IL:|smashing|) (SETQ SMASHING (CONVERT (SECOND ASSIGNMENTS))))
                (T (WARN "unknown assignment ~s" (FIRST ASSIGNMENTS))))
            (POP ASSIGNMENTS))))
    (COND
       (USING
        `(LET ((,VAR (,(INTERN (CONCATENATE 'STRING "COPY-" NAME-STRING))
                      ,USING)))
              (SETF ,@(MAPCAN #'(LAMBDA (INIT)
                                        (LIST `(,(INTERN (CONCATENATE 'STRING NAME-STRING "-" (SYMBOL-NAME
                                                                                                (CAR INIT))))
                                                  ,VAR)
                                              (CDR INIT)))
                              INITS))
              ,VAR))
       (SMASHING
        (IF INITS
            `(LET ((,VAR ,SMASHING))
                  (SETF ,@(MAPCAN #'(LAMBDA (INIT)
                                            (LIST `(,(INTERN (CONCATENATE 'STRING NAME-STRING "-"
                                                                          (SYMBOL-NAME (CAR INIT))))
                                                      ,VAR)
                                                  (CDR INIT)))
                                  INITS))
                  ,VAR)
            SMASHING))
       (T `(,(INTERN (CONCATENATE 'STRING "MAKE-" NAME-STRING))
            ,@(MAPCAN #'(LAMBDA (INIT)
                                `(,(INTERN (STRING (CAR INIT))
                                           'KEYWORD)
                                  ,(CDR INIT)))
                      INITS))))))

(IL-COPYCONV |fetch| FETCH)

(IL-COPYCONV |fetch| |ffetch|)

(IL-COPYCONV |ffetch| FFETCH)

(IL-COPYCONV |replace| REPLACE)

(IL-COPYCONV |replace| |freplace|)

(IL-COPYCONV |freplace| FREPLACE)

(IL-COPYCONV TYPE? |type?|)

(IL-COPYCONV |create| CREATE)

(IL:PUTPROPS IL:IL-RECORD IL:MAKEFILE-ENVIRONMENT (:PACKAGE "IL-CONVERT" :READTABLE "XCL"))

(IL:PUTPROPS IL:IL-RECORD IL:COPYRIGHT ("ENVOS Corporation" 1989))
```

FUNCTION INDEX

PROPERTY INDEX

VARIABLE INDEX