

File created: 13-Oct-93 18:50:03 {Pele:mv:envos}<LispCore>Sources>CLTL2>CL-ERROR.;2

previous date: 28-Feb-92 11:46:47 {Pele:mv:envos}<LispCore>Sources>CLTL2>CL-ERROR.;1

Read Table: XCL

Package: XEROX-COMMON-LISP

Format: XCCS

; Copyright (c) 1986, 1987, 1988, 1990, 1991, 1992, 1993 by Venue & Xerox Corporation. All rights reserved.

(IL:RPAQQ **IL:CL-ERRORCOMS**

((IL:COMS

;; Internal stuff.

(IL:FUNCTIONS CONDITIONS::EXPAND-WITH-COLLECTION-SITES CONDITIONS::WITH-COLLECTION-SITES)
(IL:FUNCTIONS DEFAULT-PROCEED-REPORT CONDITIONS::DEFAULT-RESTART-REPORT)
(IL:FUNCTIONS IL:WITH-GENSYMS IL:WITH-ERR-LOOP-VARS IL:STRIP-KEYWORDS IL:MAKE-REPORT-FUNCTION
CONDITIONS::NORMALIZE-SLOT-DESCRIPTION IL:CHECK-*CASE-SELECTOR
IL:COLLECT-CASE-SELECTORS IL:%SUFFIX-SYMBOL IL:PROCEED-ARG-COLLECTOR
SI::EXPAND-CONDITION-CASE SI::PROCESS-PROCEED-KEYWORDS SI::SPLIT-PROCEED-CLAUSES
SI::EXPAND-PROCEED-CASE CONDITIONS::PARSE-RESTART-CASE
CONDITIONS::CONVERT-RESTART-CASES CONDITIONS::EXPAND-RESTART-CASE
CONDITIONS::EXPAND-WCS))
(OPTIMIZERS CONDITION-CASE CATCH-ABORT PROCEED-CASE RESTART-CASE WITH-CONDITION-RESTARTS)
(IL:COMS (IL:FUNCTIONS DEFINE-CONDITION CHECK-TYPE ETYPESCASE CTYPESCASE ECASE CCASE ASSERT
HANDLER-BIND CONDITION-BIND CONDITION-CASE HANDLER-CASE IGNORE-ERRORS PROCEED-CASE
RESTART-CASE RESTART-BIND WITH-SIMPLE-RESTART DEFINE-PROCEED-FUNCTION CATCH-ABORT
WITH-CONDITION-RESTARTS))

;; Conversion functions for translating old code

(IL:FUNCTIONS CONDITIONS::CONVERT-CONDITION-CASE CONDITIONS::CONVERT-OLD-DEFINE-CONDITION)
(IL:PROP (IL:FILETYPE IL:MAKEFILE-ENVIRONMENT)
IL:CL-ERROR)
(IL:DECLARE\ : IL:DONTEVAL@LOAD IL:DOEVAL@COMPILE IL:DONTCOPY IL:COMPILEVARS (IL:ADDVARS (IL:NLAMA)
(IL:NLAML)
(IL:LAMA)))))

;; Internal stuff.

(DEFUN **CONDITIONS::EXPAND-WITH-COLLECTION-SITES** (CONDITIONS::NEW-SITES CONDITIONS::BODY
CONDITIONS::OLD-SITES)

(LET ((CONDITIONS::NEW-SITES-AND-TAILS NIL))
'(LET , (MAPCAN #'(LAMBDA (CONDITIONS::SITE)
(UNLESS (GETF CONDITIONS::OLD-SITES CONDITIONS::SITE)
'((, (CAR (PUSH (IL:GENSYM
CONDITIONS::OLD-SITES))
(LAST , (CAR (PUSH CONDITIONS::SITE CONDITIONS::OLD-SITES))))))
CONDITIONS::NEW-SITES)
(MACROLET ((**CONDITIONS::WITH-COLLECTION-SITES** ((&REST CONDITIONS::SITES)
&BODY CONDITIONS::BODY)
(**CONDITIONS::EXPAND-WITH-COLLECTION-SITES** CONDITIONS::SITES CONDITIONS::BODY
' , CONDITIONS::OLD-SITES))
(CONDITIONS::COLLECT-INTO (CONDITIONS::SITE CONDITIONS::FORM)
;; written in this way to take advantage of RPLCONS. The FORM is evaluated first so that COLLECT-INTO
;; nests properly, i.e., the test to determine if this is the first value collected should be done after the value
;; itself is generated in case it does collection as well.
(LET ((CONDITIONS::TAIL (GETF ' , CONDITIONS::OLD-SITES CONDITIONS::SITE)))
(WHEN (NULL CONDITIONS::TAIL)
(ERROR "~S is not a valid site for ~S." CONDITIONS::SITE
' CONDITIONS::COLLECT-INTO))
'(LET ((SI:: \$WITH-COLLECTION-VALUE\$, CONDITIONS::FORM)
(IF , CONDITIONS::SITE
(RPLACD , CONDITIONS::TAIL (SETQ , CONDITIONS::TAIL (LIST
SI:: \$WITH-COLLECTION-VALUE\$
)))
(SETQ , CONDITIONS::SITE (SETQ , CONDITIONS::TAIL (LIST
SI:: \$WITH-COLLECTION-VALUE\$
))))
SI:: \$WITH-COLLECTION-VALUE\$))))
, @CONDITIONS::BODY))))

(DEFMACRO **CONDITIONS::WITH-COLLECTION-SITES** ((&REST CONDITIONS::SITES)
&BODY CONDITIONS::BODY)
(**CONDITIONS::EXPAND-WITH-COLLECTION-SITES** CONDITIONS::SITES CONDITIONS::BODY NIL))

(DEFMACRO **DEFAULT-PROCEED-REPORT** (PROCEED-TYPE)
'(GET , PROCEED-TYPE ' IL:%DEFAULT-PROCEED-REPORT ' IL:DEFAULT-PROCEED-REPORTER))

(DEFMACRO **CONDITIONS::DEFAULT-RESTART-REPORT** (CONDITIONS::RESTART-TYPE)
'(GET , CONDITIONS::RESTART-TYPE ' IL:%DEFAULT-PROCEED-REPORT))

```

(DEFMACRO IL:WITH-GENSYMS (IL:VARS IL:PREFIX &BODY IL:BODY)
  `(LET , (IL:MAPCAR IL:VARS (IL:FUNCTION (LAMBDA (IL:VAR)
    ` (, IL:VAR (IL:GENSYM , IL:PREFIX))))
    , @IL:BODY))

(DEFMACRO IL:WITH-ERR-LOOP-VARS (IL:PREFIX &BODY IL:BODY)
  `(IL:WITH-GENSYMS (IL:VAL IL:BLOCK-NAME IL:AGAIN)
    , IL:PREFIX
    , @IL:BODY))

(DEFUN IL:STRIP-KEYWORDS (IL:ARGS)
  (VALUES (IL:FOR IL:OLD IL:ARGS IL:ON IL:ARGS IL:BY CDDR IL:WHILE (KEYWORDP (FIRST IL:ARGS))
    IL:COLLECT (LIST (FIRST IL:ARGS)
      (SECOND IL:ARGS)))
    IL:ARGS))

(DEFUN IL:MAKE-REPORT-FUNCTION (IL:DATUM IL:BOUND-VAR &OPTIONAL IL:TYPE-NAME)
  (ETYPESCASE IL:DATUM
    (STRING IL:DATUM)
    (LIST `(LAMBDA (, IL:BOUND-VAR *STANDARD-OUTPUT*)
      , (IF IL:TYPE-NAME
        `(IL:WITH , IL:TYPE-NAME , IL:BOUND-VAR , IL:DATUM)
        IL:DATUM))))))

(DEFUN CONDITIONS::NORMALIZE-SLOT-DESCRIPTION (CONDITIONS::SLOT-DESC)
  (ETYPESCASE CONDITIONS::SLOT-DESC
    (CONS `(, (FIRST CONDITIONS::SLOT-DESC)
      , (SECOND CONDITIONS::SLOT-DESC)
      :READ-ONLY T))
    (SYMBOL `(, CONDITIONS::SLOT-DESC NIL :READ-ONLY T))))

(DEFUN IL:CHECK-*CASE-SELECTOR (IL:SELECTOR IL:NAME)
  (IF (OR (EQ IL:SELECTOR 'T)
    (EQ IL:SELECTOR 'OTHERWISE))
    (ERROR "~A not allowed in the ~A form." IL:SELECTOR IL:NAME)
    IL:SELECTOR))

(DEFUN IL:COLLECT-CASE-SELECTORS (IL:CLAUSES IL:NAME)
  (IL:MAPCONC IL:CLAUSES (IL:FUNCTION (LAMBDA (IL:CLAUSE)
    (IL:IF (AND (CONSP (CAR IL:CLAUSE))
      (IL:FMEMB IL:NAME ' (ECASE CCASE))))
    IL:THEN (COPY-LIST (CAR IL:CLAUSE))
    IL:ELSE (LIST (IL:CHECK-*CASE-SELECTOR (CAR IL:CLAUSE)
      IL:NAME))))))

(DEFUN IL:%SUFFIX-SYMBOL (IL:SYMBOL IL:SUFFIX PACKAGE)
  (INTERN (CONCATENATE 'STRING (SYMBOL-NAME IL:SYMBOL)
    IL:SUFFIX)
    PACKAGE))

(DEFMACRO IL:PROCEED-ARG-COLLECTOR (IL:NAME)
  "Function that collects user-specified optional args (excluding the condition) for a named proceed case."
  `(GET , IL:NAME ' IL:%PROCEED-ARG-COLLECTOR))

(DEFUN SI::EXPAND-CONDITION-CASE (SI::FORM SI::CLAUSES SI::ENV SI::CTX SI::OPTIMIZE?)
  (MACROLET
    ((SI::BOUND-TYPES (SI::CLAUSE)
      `(FIRST , SI::CLAUSE)))
    (IF (NULL SI::CLAUSES)
      SI::FORM
      ;; First, precompute the handler for this condition-case. We can use a constant catch tag because of the nice dynamic nesting properties of
      ;; CONDITION-CASE.
      (LET*
        ((SI::VALUE-SLOT (IF IL:*BYTECOMPILER-IS-EXPANDING*
          (IL:GENSYM)
          'SI::CONDITION-CASE-VALUES))
          (SI::NO-ERROR-CLAUSE (ASSOC ' :NO-ERROR SI::CLAUSES :TEST 'EQ))
          (SI::ONLY-ONE-VALUE? (AND (NULL SI::NO-ERROR-CLAUSE)
            SI::OPTIMIZE? SI::CTX (EQL (COMPILER:CONTEXT-VALUES-USED SI::CTX)
              1))))
          (DECLARE (SPECIAL IL:*BYTECOMPILER-IS-EXPANDING*))
          (FLET
            ((SI::CONSTRUCT-NO-ERROR-CODE
              (SI::VALUE-FORM)
              (DESTRUCTURING-BIND (SI::SELECTOR SI::BOUND-VARS &REST SI::BODY)

```



```

      (SETF SI::FILTER SI::VALUE))
    (:FILTER
      (IF SI::FILTER (ERROR "Duplicate filter specified for proceed type ~S."
                            SI::NAME))
      (SETF SI::FILTER `(LAMBDA NIL ,SI::VALUE)))
    (:CONDITION
      (IF SI::FILTER (ERROR "Duplicate test form specified for proceed type ~S."
                            SI::NAME))
      (SETF SI::FILTER
        ;; consider using a closure here.
        `(LAMBDA NIL (TYPEP *CURRENT-CONDITION* ',SI::VALUE))))
    (:REPORT-FUNCTION
      (IF SI::REPORT (ERROR "Duplicate report form specified for proceed type ~S."
                            SI::NAME))
      (SETF SI::REPORT SI::VALUE))
    (:REPORT
      (IF SI::REPORT (ERROR "Duplicate report form specified for proceed type ~S."
                            SI::NAME))
      (SETF SI::REPORT (ETYPESCASE SI::VALUE
                                (STRING SI::VALUE)
                                (LIST `(LAMBDA (*STANDARD-OUTPUT*
                                                ,SI::VALUE))))))
      (OTHERWISE (CERROR "Ignore key/value pair" "Illegal keyword ~S in proceed case
                          ~S." SI::KEY SI::NAME))))
    (VALUES SI::FILTER SI::REPORT SI::TAIL)))

```

```

(DEFUN SI::SPLIT-PROCEED-CLAUSES (SI::CLAUSES SI::ENV SI::OPTIMIZE?)
  (LET
    (SI::CASES SI::BODIES)
    (IL:FOR SI::CLAUSE IL:IN SI::CLAUSES IL:AS SI::SELECTOR IL:FROM 0
      IL:DO (DESTRUCTURING-BIND
        (SI::NAME SI::VARS)
        SI::CLAUSE
        (MULTIPLE-VALUE-BIND (SI::FILTER SI::REPORT SI::TAIL)
          (SI::PROCESS-PROCEED-KEYWORDS SI::NAME (CDDR SI::CLAUSE)
            *PACKAGE*))
          (IF (NULL SI::NAME)
            (UNLESS SI::REPORT (ERROR "Unnamed proceed cases must have a report method: ~S" SI::CLAUSE)))
          )
        (MACROLET ((SI::CONSTANT-PROCEED-CASE? NIL
          '(AND (OR (NULL SI::FILTER)
            (AND (SYMBOLP SI::FILTER)
              (OR (NULL SI::ENV)
                (NOT (COMPILER:ENV-FBOUNDP SI::ENV SI::FILTER))))))
            (OR (NULL SI::REPORT)
              (STRINGP SI::REPORT)
              (AND (SYMBOLP SI::REPORT)
                (OR (NULL SI::ENV)
                  (NOT (COMPILER:ENV-FBOUNDP SI::ENV NIL)))))))
          (PUSH (IF (AND SI::OPTIMIZE? (SI::CONSTANT-PROCEED-CASE?))
            (IL:MAKE-PROCEED-CASE :NAME SI::NAME :SELECTOR SI::SELECTOR :TEST SI::FILTER
              :REPORT SI::REPORT)
            `(IL:MAKE-PROCEED-CASE :NAME ',SI::NAME :SELECTOR ,SI::SELECTOR :TEST
              , (AND SI::FILTER '#',SI::FILTER)
              :REPORT
              , (AND SI::REPORT (IF (STRINGP SI::REPORT)
                SI::REPORT
                '#',SI::REPORT))))
            SI::CASES))
          (PUSH (LIST* SI::SELECTOR SI::VARS SI::TAIL)
            SI::BODIES)))
    (VALUES (FLET ((SI::MAYBE-QUOTE (SI::X)
      (IF (IL:PROCEED-CASE-P SI::X)
        (IL:KWOTE SI::X)
        SI::X)))
      (COND
        ((EQL (IL:LENGTH SI::CASES)
          1)
          (SI::MAYBE-QUOTE (FIRST SI::CASES)))
          ((EVERY 'IL:PROCEED-CASE-P SI::CASES)
            (IL:KWOTE (NREVERSE SI::CASES)))
          (T `(LIST ,@(MAPCAR #'SI::MAYBE-QUOTE (NREVERSE SI::CASES))))))
      (REVERSE SI::BODIES))))

```

```

(DEFUN SI::EXPAND-PROCEED-CASE (SI::FORM SI::CLAUSES SI::ENV SI::CTX SI::OPTIMIZE?)
  (LET
    ((SI::VALUE-SLOT (IF IL:*BYTECOMPILER-IS-EXPANDING*
      (IL:GENSYM)
      'SI::PROCEED-CASE-NORMAL-VALUES))
    (SI::ONLY-ONE-VALUE? (AND SI::OPTIMIZE? SI::CTX (EQL (COMPILER:CONTEXT-VALUES-USED SI::CTX)
      1))))
    (DECLARE (SPECIAL IL:*BYTECOMPILER-IS-EXPANDING*))
    (MULTIPLE-VALUE-BIND (SI::CASES SI::BODIES)
      (SI::SPLIT-PROCEED-CLAUSES SI::CLAUSES SI::ENV SI::OPTIMIZE?))

```

```
(IF (NULL SI::CASES)
  SI::FORM
  `(LET* (,SI::VALUE-SLOT (SI::PROCEED-CASE-SELECTOR-AND-VALUES
    (LET ((IL:*PROCEED-CASES* (CONS ,SI::CASES IL:*PROCEED-CASES*)))
      (CATCH IL:*PROCEED-CASES*
        (SETF ,SI::VALUE-SLOT ,(IF SI::ONLY-ONE-VALUE?
          SI::FORM
          `(MULTIPLE-VALUE-LIST ,SI::FORM)))
        :NORMAL))))))
  (IF (EQ SI::PROCEED-CASE-SELECTOR-AND-VALUES :NORMAL)
    ,(IF SI::ONLY-ONE-VALUE?
      SI::VALUE-SLOT
      `(VALUES-LIST ,SI::VALUE-SLOT))
    ,(FLET ((SI::CREATE-A-CASE (SI::X)
      (DESTRUCTURING-BIND (CASE SI::ARGS
        &REST
        SI::BODY)
        SI::X
        (IF (NULL SI::ARGS)
          `(,CASE ,@SI::BODY)
          `(,CASE (DESTRUCTURING-BIND (&OPTIONAL ,@SI::ARGS)
            SI::PROCEED-CASE-VALUES
            ,@SI::BODY))))))
      (IF (EVERY #'(LAMBDA (SI::X)
        (NULL (SECOND SI::X)))
        SI::BODIES)
        `(CASE (CAR SI::PROCEED-CASE-SELECTOR-AND-VALUES)
          (IL:\\\\,@ (MAPCAR #'SI::CREATE-A-CASE SI::BODIES)))
        `( (LAMBDA (SI::PROCEED-CASE-SELECTOR SI::PROCEED-CASE-VALUES)
          (CASE SI::PROCEED-CASE-SELECTOR
            (IL:\\\\,@ (MAPCAR #'SI::CREATE-A-CASE SI::BODIES)))
            (CAR SI::PROCEED-CASE-SELECTOR-AND-VALUES)
            (CDR SI::PROCEED-CASE-SELECTOR-AND-VALUES))))))))))
```

(DEFUN **CONDITIONS::PARSE-RESTART-CASE** (CONDITIONS::NAME CONDITIONS::CLAUSE)
 ; Edited 24-Feb-92 11:49 by jrb:

```
(LET (CONDITIONS::FILTER CONDITIONS::REPORT CONDITIONS::INTERACTIVE)
  (IL:WHILE (KEYWORDP (FIRST CONDITIONS::CLAUSE))
    IL:DO (LET* ((CONDITIONS::KEY (POP CONDITIONS::CLAUSE))
      (CONDITIONS::VALUE (POP CONDITIONS::CLAUSE))
      (CASE CONDITIONS::KEY
        (:FILTER (UNLESS CONDITIONS::FILTER (SETF CONDITIONS::FILTER CONDITIONS::VALUE)))
        (:TEST
          ;; This is actually not too bad as a back-patch goes; the compiler will correctly unroll (FUNCALL (FUNCTION
          ;; <drek>) <args>) into as direct a call to <drek> as possible
          (UNLESS CONDITIONS::FILTER
            (SETF CONDITIONS::FILTER `(LAMBDA NIL (FUNCALL #' ,CONDITIONS::VALUE
              *CURRENT-CONDITION*))))))
        (:CONDITION (UNLESS CONDITIONS::FILTER
          (SETF CONDITIONS::FILTER `(LAMBDA NIL (TYPEP *CURRENT-CONDITION*
            ',CONDITIONS::VALUE))))))
        (:REPORT (UNLESS CONDITIONS::REPORT (SETF CONDITIONS::REPORT CONDITIONS::VALUE)))
        (:INTERACTIVE (UNLESS CONDITIONS::INTERACTIVE (SETF CONDITIONS::INTERACTIVE
          CONDITIONS::VALUE)))
        (OTHERWISE (CERROR "Ignore key/value pair" "Illegal keyword ~S in restart named ~S."
          CONDITIONS::KEY CONDITIONS::NAME))))))
  (VALUES CONDITIONS::FILTER CONDITIONS::REPORT CONDITIONS::INTERACTIVE CONDITIONS::CLAUSE)))
```

(DEFUN **CONDITIONS::CONVERT-RESTART-CASES** (CONDITIONS::CLAUSES CONDITIONS::ENV CONDITIONS::OPTIMIZE?)

```
(LET
  (CONDITIONS::CASES CONDITIONS::BODIES CONDITIONS::ANY-ARGLISTS? (CONDITIONS::ALL-CONSTANT? T))
  (CONDITIONS::WITH-COLLECTION-SITES
    (CONDITIONS::CASES CONDITIONS::BODIES)
    (IL:FOR CONDITIONS::CLAUSE IL:IN CONDITIONS::CLAUSES IL:AS CONDITIONS::SELECTOR IL:FROM 0
      IL:DO
        (LET* ((CONDITIONS::NAME (POP CONDITIONS::CLAUSE))
          (CONDITIONS::ARGLIST (POP CONDITIONS::CLAUSE))
          (WHEN CONDITIONS::ARGLIST (SETF CONDITIONS::ANY-ARGLISTS? T))
          (MULTIPLE-VALUE-BIND (CONDITIONS::FILTER CONDITIONS::REPORT CONDITIONS::INTERACTIVE
            CONDITIONS::CODE)
            (CONDITIONS::PARSE-RESTART-CASE CONDITIONS::NAME CONDITIONS::CLAUSE)
            (MACROLET ((CONDITIONS::CONSTANT-RESTART? NIL
              '(AND (OR (NULL CONDITIONS::FILTER)
                (AND (SYMBOLP CONDITIONS::FILTER)
                  (OR (NULL CONDITIONS::ENV)
                    (NOT (COMPILER:ENV-FBOUNDP CONDITIONS::ENV
                      CONDITIONS::FILTER))))))
                (OR (NULL CONDITIONS::REPORT)
                  (STRINGP CONDITIONS::REPORT)
                  (AND (SYMBOLP CONDITIONS::REPORT)
                    (OR (NULL CONDITIONS::ENV)
                      (NOT (COMPILER:ENV-FBOUNDP CONDITIONS::ENV
                        CONDITIONS::REPORT))))))
                (OR (NULL CONDITIONS::INTERACTIVE)
```



```

        'SI::PROCEED-CASE-NORMAL-VALUES))
(CONDITIONS::ONLY-ONE-VALUE? (AND CONDITIONS::OPTIMIZE? CONDITIONS::CTX (EQL (COMPILER:CONTEXT-VALUES-USED
                                                                           CONDITIONS::CTX
                                                                           1))))

(CONDITIONS::COND-BINDING (GENSYM))
(CONDITIONS::RESTARTS-BINDING (GENSYM))
(DECLARE (SPECIAL IL:*BYTECOMPILER-IS-EXPANDING*))
;; This is way weird, but I think it should work. The idea is we bind copies of the restarts which have their filter tests munged to insure that they only
;; will respond to the appropriate condition object. I have NO idea why anyone would want this level of control over the system, but it's here now.
` (LET
  ((,CONDITIONS::COND-BINDING ,CONDITIONS::CONDITION-FORM))
  (DECLARE (SPECIAL ,CONDITIONS::COND-BINDING))
  (LET* ((,CONDITIONS::RESTARTS-BINDING
         (MAPCAR #'(LAMBDA (CONDITIONS::R)
                   (LET ((CONDITIONS::NEWRS (CONDITIONS::COPY-RESTART CONDITIONS::R)))
                       (SETF (CONDITIONS::RESTART-TEST CONDITIONS::NEWRS)
                             (IF (CONDITIONS::RESTART-TEST CONDITIONS::R)
                                 `(LAMBDA NIL (AND (EQ *CURRENT-CONDITION*
                                                       ',,CONDITIONS::COND-BINDING)
                                                  (FUNCALL ',(CONDITIONS::RESTART-TEST
                                                         CONDITIONS::R))))
                                 `(LAMBDA NIL (EQ *CURRENT-CONDITION* ',,CONDITIONS::COND-BINDING)
                                       ))
                               ))
                   ))
        ,CONDITIONS::NEWRS))
    ,CONDITIONS::RESTARTS-FORM))
  ,CONDITIONS::VALUE-SLOT
  (SI::PROCEED-CASE-SELECTOR-AND-VALUES (LET ((IL:*PROCEED-CASES* (CONS
                                                                    ,CONDITIONS::RESTARTS-BINDING
                                                                    IL:*PROCEED-CASES*))
      (CATCH IL:*PROCEED-CASES*
        (SETF ,CONDITIONS::VALUE-SLOT
              , (IF CONDITIONS::ONLY-ONE-VALUE?
                    CONDITIONS::BODY
                    `(MULTIPLE-VALUE-LIST ,CONDITIONS::BODY))
              )
          :NORMAL))))
    (IF (EQ SI::PROCEED-CASE-SELECTOR-AND-VALUES :NORMAL)
      , (IF CONDITIONS::ONLY-ONE-VALUE?
            CONDITIONS::VALUE-SLOT
            `(VALUES-LIST ,CONDITIONS::VALUE-SLOT))
      (APPLY (CONDITIONS::RESTART-FUNCTION (NTH (CAR SI::PROCEED-CASE-SELECTOR-AND-VALUES)
                                                ,CONDITIONS::RESTARTS-BINDING))
            (CDR SI::PROCEED-CASE-SELECTOR-AND-VALUES))))))

(DEFOPTIMIZER CONDITION-CASE (FORM &REST CLAUSES &ENVIRONMENT ENV &CONTEXT CTX)
  (SI::EXPAND-CONDITION-CASE FORM CLAUSES ENV CTX T))

(DEFOPTIMIZER CATCH-ABORT (PRINT-FORM &BODY FORMS &ENVIRONMENT ENV &CONTEXT CTX)
  (IF (AND CTX (EQL (COMPILER:CONTEXT-VALUES-USED CTX)
                    1))
    `(PROCEED-CASE (PROGN ,@FORMS)
                   (ABORT NIL :REPORT ,PRINT-FORM NIL))
    'COMPILER:PASS))

(DEFOPTIMIZER PROCEED-CASE (FORM &REST CLAUSES &ENVIRONMENT ENV &CONTEXT CTX)
  (SI::EXPAND-PROCEED-CASE FORM CLAUSES ENV CTX T))

(DEFOPTIMIZER RESTART-CASE (CONDITIONS::FORM &REST CONDITIONS::CLAUSES &ENVIRONMENT CONDITIONS::ENV &CONTEXT
  CONDITIONS::CTX)
  (CONDITIONS::EXPAND-RESTART-CASE CONDITIONS::FORM CONDITIONS::CLAUSES
  CONDITIONS::ENV CONDITIONS::CTX T))

(DEFOPTIMIZER WITH-CONDITION-RESTARTS (CONDITIONS::CONDITION-FORM CONDITIONS::RESTARTS-FORM &REST
  CONDITIONS::BODY &ENVIRONMENT CONDITIONS::ENV &CONTEXT
  CONDITIONS::CTX)
  (CONDITIONS::EXPAND-WCS CONDITIONS::CONDITION-FORM
  CONDITIONS::RESTARTS-FORM CONDITIONS::BODY CONDITIONS::ENV
  CONDITIONS::CTX T))

(DEFDEFINER DEFINE-CONDITION IL:STRUCTURES (CONDITIONS::NAME (CONDITIONS::PARENT-TYPE)
  &OPTIONAL CONDITIONS::SLOTS &REST CONDITIONS::OPTIONS)
  (LET ((CONDITIONS::PARENT-SLOTS (AND CONDITIONS::PARENT-TYPE (CL::STRUCTURE-SLOT-NAMES
  CONDITIONS::PARENT-TYPE T)))
        CONDITIONS::REPORTER CONDITIONS::HANDLER CONDITIONS::SLOT-DESCRIPTIONS
        CONDITIONS::SHADOWED-SLOT-DESCRIPTIONS CONDITIONS::CLASS-OPTIONS CONDITIONS::DOC)
    (SETQ CONDITIONS::SLOT-DESCRIPTIONS (WITH-COLLECTION (DOLIST (CONDITIONS::SLOT CONDITIONS::SLOTS)
  (SETQ CONDITIONS::SLOT (
  CONDITIONS::NORMALIZE-SLOT-DESCRIPTION
  CONDITIONS::SLOT))
  (IF (MEMBER (FIRST CONDITIONS::SLOT)
              CONDITIONS::PARENT-SLOTS :TEST

```

```

'EQ)
(PUSH CONDITIONS::SLOT
  CONDITIONS::SHADOWED-SLOT-DESCRIPTIONS
)
(COLLECT CONDITIONS::SLOT))))
(SETQ CONDITIONS::CLASS-OPTIONS (WITH-COLLECTION (DOLIST (CONDITIONS::OPTION CONDITIONS::OPTIONS)
  (MACROLET ((CONDITIONS::MULTIPLE-OPTION-ERROR
    NIL
    '(CERROR "Ignore the later ~*~S
      option" "~S is a duplicate ~S
      option for ~S."
      CONDITIONS::OPTION (FIRST
        CONDITIONS::OPTION
      )
      'DEFINE-CONDITION)))
  (ECASE (FIRST CONDITIONS::OPTION)
    ((:CONC-NAME :INLINE) (COLLECT
      CONDITIONS::OPTION
    ))
    (:DOCUMENTATION
      (AND CONDITIONS::DOC
        (SETQ CONDITIONS::DOC
          (SECOND CONDITIONS::OPTION
        )))
      (:REPORT (IF (NULL CONDITIONS::REPORTER)
        (SETQ CONDITIONS::REPORTER
          (SECOND
            CONDITIONS::OPTION
          ))
        (
          CONDITIONS::MULTIPLE-OPTION-ERROR
        )))
      (:HANDLE (IF (NULL CONDITIONS::HANDLER)
        (SETQ CONDITIONS::HANDLER
          (SECOND
            CONDITIONS::OPTION
          ))
        (
          CONDITIONS::MULTIPLE-OPTION-ERROR
        ))))))))
'(PROGN (DEFSTRUCT (,CONDITIONS::NAME ,@(AND CONDITIONS::PARENT-TYPE
  ; hook for CONDITION
  '((:INCLUDE ,CONDITIONS::PARENT-TYPE ,.(NREVERSE
    CONDITIONS::SHADOWED-SLOT-DESCRIPTIONS
  )))
  ,.CONDITIONS::CLASS-OPTIONS
  (:PRINT-FUNCTION IL:%PRINT-CONDITION)
  (:CONSTRUCTOR ,(IL:%SUFFIX-SYMBOL CONDITIONS::NAME " constructor" (
    SYMBOL-PACKAGE
    CONDITIONS::NAME
  )))
  (:COPIER NIL)
  (:PREDICATE NIL))
, .CONDITIONS::SLOT-DESCRIPTIONS)
(EVAL-WHEN (LOAD EVAL)
  ,@(AND CONDITIONS::DOC `((SETF (DOCUMENTATION ',CONDITIONS::NAME 'TYPE)
  ',CONDITIONS::DOC)))
  ,@(IF (CONSP CONDITIONS::REPORTER)
    (LET ((CONDITIONS::REPORTER-NAME (IL:%SUFFIX-SYMBOL CONDITIONS::NAME " report
      method" (SYMBOL-PACKAGE CONDITIONS::NAME)))
    )
      (PROG1 `((SETF (SYMBOL-FUNCTION ',CONDITIONS::REPORTER-NAME)
        #' ,CONDITIONS::REPORTER))
        (SETQ CONDITIONS::REPORTER CONDITIONS::REPORTER-NAME))))
    (SETF (CONDITION-REPORTER ',CONDITIONS::NAME)
      , (TYPECASE CONDITIONS::REPORTER
        (NULL NIL)
        (STRING CONDITIONS::REPORTER)
        (T `#' ,CONDITIONS::REPORTER)))
    ,@(IF (CONSP CONDITIONS::HANDLER)
      (LET ((CONDITIONS::HANDLER-NAME (IL:%SUFFIX-SYMBOL CONDITIONS::NAME " default
        handler" (SYMBOL-PACKAGE CONDITIONS::NAME)))
      )
        (PROG1 `((SETF (SYMBOL-FUNCTION ',CONDITIONS::HANDLER-NAME)
          #' ,CONDITIONS::HANDLER))
          (SETQ CONDITIONS::HANDLER CONDITIONS::HANDLER-NAME))))
      (SETF (CONDITION-HANDLER ',CONDITIONS::NAME)
        , (AND CONDITIONS::HANDLER `#' ,CONDITIONS::HANDLER))))))

```

```

(DEFMACRO CHECK-TYPE (CL::PLACE CL::TYPESPEC &OPTIONAL STRING)
  (IL:WITH-ERR-LOOP-VARS "CHECK-TYPE" `(BLOCK ,IL:BLOCK-NAME
    (TAGBODY ,IL:AGAIN (LET ((,IL:VAL ,CL::PLACE))
      (WHEN (TYPEP ,IL:VAL ',CL::TYPESPEC)
        (RETURN-FROM ,IL:BLOCK-NAME))
      (SETF ,CL::PLACE (IL:CHECK-TYPE-FAIL

```



```

T
',CL::PLACE
,IL:VAL
',CL::TYPESPEC
,STRING))
(GO ,IL:AGAIN))))))

(DEFMACRO ETYPECASE (CL::KEYFORM &BODY CL::CLAUSES)
  (IL:WITH-GENSYMS (CL::VALUE)
    "ETYPECASE"
    (LET ((CL::CASE-SELECTORS (CONS 'OR (IL:COLLECT-CASE-SELECTORS CL::CLAUSES 'ETYPECASE)))
          `(LET ((,CL::VALUE ,CL::KEYFORM)
                (TYPECASE ,CL::VALUE
                  (IL:\\,@ CL::CLAUSES)
                  (T (IL:CHECK-TYPE-FAIL NIL ',CL::KEYFORM ,CL::VALUE ',CL::CASE-SELECTORS NIL)))))))

```

```

(DEFMACRO CTYPECASE (CL::KEYPLACE &BODY CL::CLAUSES)
  (LET ((CL::CASE-SELECTORS (CONS 'OR (IL:COLLECT-CASE-SELECTORS CL::CLAUSES 'CTYPECASE)))
        (IL:WITH-ERR-LOOP-VARS
          "CTYPECASE"
          `(BLOCK ,IL:BLOCK-NAME
              (TAGBODY ,IL:AGAIN (LET ((,IL:VAL ,CL::KEYPLACE)
                                      (RETURN-FROM ,IL:BLOCK-NAME
                                        (TYPECASE ,IL:VAL
                                          (IL:\\,@ CL::CLAUSES)
                                          (T (SETF ,CL::KEYPLACE (IL:CHECK-TYPE-FAIL T
                                                                    ',CL::KEYPLACE
                                                                    ,IL:VAL
                                                                    ',CL::CASE-SELECTORS NIL))
                                              (GO ,IL:AGAIN))))))))))

```

```

(DEFMACRO ECASE (CL::KEYFORM &REST CL::CLAUSES)
  (IL:WITH-GENSYMS (CL::VALUE)
    "ECASE"
    (LET ((CL::CASE-SELECTORS (IL:COLLECT-CASE-SELECTORS CL::CLAUSES 'ECASE))
          (IF CL::CASE-SELECTORS
              `(LET ((,CL::VALUE ,CL::KEYFORM)
                    (CASE ,CL::VALUE
                      (IL:\\,@ CL::CLAUSES)
                      (T (IL:ECASE-FAIL NIL ',CL::KEYFORM ,CL::VALUE ',CL::CASE-SELECTORS))))
              (ERROR "Empty case statement."))))

```

```

(DEFMACRO CCASE (CL::KEYFORM &BODY CL::CLAUSES)
  (LET ((CL::CASE-SELECTORS (IL:COLLECT-CASE-SELECTORS CL::CLAUSES 'CCASE))
        (UNLESS CL::CASE-SELECTORS (ERROR "Empty CCASE."))
        (IL:WITH-ERR-LOOP-VARS
          "CCASE"
          `(BLOCK ,IL:BLOCK-NAME
              (TAGBODY ,IL:AGAIN (LET ((,IL:VAL ,CL::KEYFORM)
                                      (RETURN-FROM ,IL:BLOCK-NAME
                                        (CASE ,IL:VAL
                                          (IL:\\,@ CL::CLAUSES)
                                          (T (SETF ,CL::KEYFORM (IL:ECASE-FAIL T ',CL::KEYFORM
                                                                    ,IL:VAL
                                                                    ',CL::CASE-SELECTORS))
                                              (GO ,IL:AGAIN))))))))))

```

```

(DEFMACRO ASSERT (CL::TEST-FORM &OPTIONAL CL::PLACES CL::DATUM &REST CL::ARGS)
  (UNLESS (LISTP CL::PLACES)
    (ERROR "~S should be a list of places." CL::PLACES))
  (IL:WITH-ERR-LOOP-VARS "ASSERT" `(BLOCK ,IL:BLOCK-NAME
                                  (TAGBODY ,IL:AGAIN (WHEN ,CL::TEST-FORM
                                                          (RETURN-FROM ,IL:BLOCK-NAME NIL)
                                                          (IL:ASSERT-FAIL ,CL::DATUM ,@CL::ARGS)
                                                          (GO ,IL:AGAIN))))))

```

```

(DEFMACRO HANDLER-BIND (BINDINGS &REST FORMS)
  "Eval forms under temporary new condition handlers."
  (IF (NULL BINDINGS)
      `(PROGN ,@FORMS)
      `(LET
          ((IL:*CONDITION-HANDLER-BINDINGS*
            (CONS ,(IF (NULL (REST BINDINGS))
                      `(CONS ',(FIRST (FIRST BINDINGS))
                              ,(SECOND (FIRST BINDINGS)))
                      `(LIST :MULTIPLE-HANDLER-BINDINGS
                              ,(WITH-COLLECTION (DOLIST (BINDING BINDINGS)
                                                    (COLLECT ',(FIRST BINDING))
                                                    (COLLECT (SECOND BINDING))))))
            IL:*CONDITION-HANDLER-BINDINGS*))
        ,@FORMS)))

```

```
(DEFMACRO CONDITION-BIND (BINDINGS &REST FORMS)
  "Eval forms under temporary new condition handlers; synonym for HANDLER-BIND"
  `(HANDLER-BIND ,BINDINGS ,@FORMS))
```

```
(DEFMACRO CONDITION-CASE (FORM &REST CLAUSES)
  "Eval form under condition handlers that provide alternate continuations."
  (SI::EXPAND-CONDITION-CASE FORM CLAUSES NIL NIL NIL))
```

```
(DEFMACRO HANDLER-CASE (CONDITIONS::FORM &REST CONDITIONS::CLAUSES)
  "Eval form under condition handlers that provide alternate continuations."
  (SI::EXPAND-CONDITION-CASE CONDITIONS::FORM CONDITIONS::CLAUSES NIL NIL NIL)
  `(CONDITION-CASE ,CONDITIONS::FORM ,@CONDITIONS::CLAUSES))
```

```
(DEFMACRO IGNORE-ERRORS (&BODY IL:FORMS)
  "Eval forms with handler for any condition of type ERROR."
  `(CONDITION-CASE (PROGN ,@IL:FORMS)
    (ERROR (CONDITION)
      (VALUES NIL CONDITION))))
```

```
(DEFMACRO PROCEED-CASE (FORM &REST CLAUSES &ENVIRONMENT ENV)
  "Eval forms, establishing a place to proceed from errors."
  (SI::EXPAND-PROCEED-CASE FORM CLAUSES ENV NIL NIL))
```

```
(DEFMACRO RESTART-CASE (CONDITIONS::FORM &REST CONDITIONS::CLAUSES &ENVIRONMENT CONDITIONS::ENV)
  (CONDITIONS::EXPAND-RESTART-CASE CONDITIONS::FORM CONDITIONS::CLAUSES CONDITIONS::ENV NIL NIL))
```

```
(DEFMACRO RESTART-BIND (CONDITIONS::BINDINGS &BODY CONDITIONS::BODY)
  ;; This should also be optimized along the lines of RESTART-BIND. Not as important since this one will be rare.
  (IF (NULL CONDITIONS::BINDINGS)
    `(PROGN ,@CONDITIONS::BODY)
    (LET ((CONDITIONS::CASES (MAPCAR #'(LAMBDA (CONDITIONS::BINDING)
      (DESTRUCTURING-BIND (CONDITIONS::NAME &KEY
        CONDITIONS::INTERACTIVE-FUNCTION
        CONDITIONS::REPORT-FUNCTION
        CONDITIONS::FILTER-FUNCTION)
        CONDITIONS::BINDING
        `(CONDITIONS::MAKE-RESTART :NAME ',CONDITIONS::NAME
          :SELECTOR 'SI::COMPLEX-RESTART-MARKER :FUNCTION
            ,FUNCTION :INTERACTIVE-FN
            ,CONDITIONS::INTERACTIVE-FUNCTION :REPORT
            ,CONDITIONS::REPORT-FUNCTION :TEST
            ,CONDITIONS::FILTER-FUNCTION))))
        CONDITIONS::BINDINGS)))
      `(LET ((IL:*PROCEED-CASES* (CONS ,(IF (NULL (REST CONDITIONS::CASES))
        (FIRST CONDITIONS::CASES)
        `(LIST ,@CONDITIONS::CASES))
        IL:*PROCEED-CASES*)))
        ,@CONDITIONS::BODY))))))
```

```
(DEFMACRO WITH-SIMPLE-RESTART ((RESTART-NAME CONDITIONS::FORMAT-STRING &REST CONDITIONS::FORMAT-ARGS)
  &BODY CONDITIONS::BODY)
  `(RESTART-CASE (PROGN ,@CONDITIONS::BODY)
    (,RESTART-NAME NIL :REPORT (LAMBDA (STREAM)
      (FORMAT STREAM ,CONDITIONS::FORMAT-STRING
        ,@CONDITIONS::FORMAT-ARGS))
      (VALUES NIL T))))
```

```
(DEFDEFINER DEFINE-PROCEED-FUNCTION IL:FUNCTIONS (NAME &REST TAIL)
  (MULTIPLE-VALUE-BIND (FILTER REPORT ARGLIST)
    (SI::PROCESS-PROCEED-KEYWORDS NAME TAIL (SYMBOL-PACKAGE NAME))
    (LET ((VARS (IL:MAPCAR ARGLIST (IL:FUNCTION (IL:LAMBDA (X)
      (IF (SYMBOLP X)
        X
        (CAR X)))))))
      (UNLESS REPORT
        (SETF REPORT 'IL:DEFAULT-PROCEED-REPORTER))
      `(PROGN ,@(IF (CONSP FILTER)
        (LET ((FILTER-FUNCTION (IL:%SUFFIX-SYMBOL NAME " proceed case default test" (
          SYMBOL-PACKAGE
          NAME))))
          (PROG1 `((SETF (SYMBOL-FUNCTION ',FILTER-FUNCTION)
            #' ,FILTER))
            (SETF FILTER-FUNCTION)))
          (SETF (DEFAULT-PROCEED-TEST ',NAME)
            ,(AND FILTER #' ,FILTER)))
        ,@(IF (CONSP REPORT)
          (LET ((REPORTER (IL:%SUFFIX-SYMBOL NAME " proceed case default report method"
```



```

, (WALK-FORM CONDITIONS::VALUE :WALK-FUNCTION
  #' (LAMBDA (CONDITIONS::FORM CONDITIONS::CONTEXT)
    (IF (AND (SYMBOLP CONDITIONS::FORM)
              (MEMBER CONDITIONS::FORM CONDITIONS::ALL-SLOTS))
        (VALUES (LIST (IL:%SUFFIX-SYMBOL CONDITIONS::NAME
                      (CONCATENATE 'STRING "-" (SYMBOL-NAME
                                   CONDITIONS::FORM
                                   )))
                  *PACKAGE*)
                CONDITION)
          T)
      CONDITIONS::FORM))
  :COPY T :LEXICAL-VARIABLES (LIST CONDITION)))))))))
(:HANDLER-FUNCTION (COLLECT `(:HANDLE ,CONDITIONS::VALUE)))
(:HANDLE
 (COLLECT
  `(:HANDLE
    , (LET
      ((CONDITION (INTERN "CONDITION" *PACKAGE*))
        (CONDITIONS::ALL-SLOTS (APPEND (CL::STRUCTURE-SLOT-NAMES CONDITIONS::PARENT-TYPE T)
                                         (MAPCAR #'(LAMBDA (CONDITIONS::X)
                                                    (IF (CONSP CONDITIONS::X)
                                                        (CAR CONDITIONS::X)
                                                        CONDITIONS::X)))
                                         CONDITIONS::SLOTS))))
        (LAMBDA (,CONDITION)
          , (WALK-FORM CONDITIONS::VALUE :WALK-FUNCTION
            #' (LAMBDA (CONDITIONS::FORM CONDITIONS::CONTEXT)
              (IF (AND (SYMBOLP CONDITIONS::FORM)
                        (MEMBER CONDITIONS::FORM CONDITIONS::ALL-SLOTS))
                  (VALUES (LIST (IL:%SUFFIX-SYMBOL CONDITIONS::NAME
                                (CONCATENATE 'STRING "-" (SYMBOL-NAME
                                             CONDITIONS::FORM
                                             )))
                              *PACKAGE*)
                            CONDITION)
                    T)
                CONDITIONS::FORM))
            :COPY T :LEXICAL-VARIABLES (LIST CONDITION)))))))))
` (DEFINE-CONDITION ,CONDITIONS::NAME (,CONDITIONS::PARENT-TYPE)
  (, @CONDITIONS::SLOTS)
  , @CONDITIONS::OPTIONS))))))

```

(IL:PUTPROPS IL:CL-ERROR IL:FILETYPE :COMPILE-FILE)

(IL:PUTPROPS IL:CL-ERROR IL:MAKEFILE-ENVIRONMENT (:READTABLE "XCL" :PACKAGE "XCL"))

(IL:DECLARE\ : IL:DONTEVAL@LOAD IL:DOEVAL@COMPILE IL:DONTCOPY IL:COMPILEVAR)

(IL:ADDTOVAR IL:NLAMA)

(IL:ADDTOVAR IL:NLAML)

(IL:ADDTOVAR IL:LAMA)

)

(IL:PUTPROPS IL:CL-ERROR IL:COPYRIGHT ("Venue & Xerox Corporation" 1986 1987 1988 1990 1991 1992 1993))

FUNCTION INDEX

IL:%SUFFIX-SYMBOL	2	CONDITIONS::EXPAND-WCS	6
IL:CHECK-*CASE-SELECTOR	2	CONDITIONS::EXPAND-WITH-COLLECTION-SITES	1
IL:COLLECT-CASE-SELECTORS	2	IL:MAKE-REPORT-FUNCTION	2
CONDITIONS::CONVERT-CONDITION-CASE	11	CONDITIONS::NORMALIZE-SLOT-DESCRIPTION	2
CONDITIONS::CONVERT-OLD-DEFINE-CONDITION	11	CONDITIONS::PARSE-RESTART-CASE	5
CONDITIONS::CONVERT-RESTART-CASES	5	SI::PROCESS-PROCEED-KEYWORDS	3
SI::EXPAND-CONDITION-CASE	2	SI::SPLIT-PROCEED-CLAUSES	4
SI::EXPAND-PROCEED-CASE	4	IL:STRIP-KEYWORDS	2
CONDITIONS::EXPAND-RESTART-CASE	6		

MACRO INDEX

ASSERT	9	HANDLER-CASE	10
CATCH-ABORT	11	IGNORE-ERRORS	10
CCASE	9	IL:PROCEED-ARG-COLLECTOR	2
CHECK-TYPE	8	PROCEED-CASE	10
CONDITION-BIND	10	RESTART-BIND	10
CONDITION-CASE	10	RESTART-CASE	10
CTYPECASE	9	CONDITIONS::WITH-COLLECTION-SITES	1
DEFAULT-PROCEED-REPORT	1	WITH-CONDITION-RESTARTS	11
CONDITIONS::DEFAULT-RESTART-REPORT	1	IL:WITH-ERR-LOOP-VARS	2
ECASE	9	IL:WITH-GENSYMS	2
ETYPECASE	9	WITH-SIMPLE-RESTART	10
HANDLER-BIND	9		

OPTIMIZER INDEX

CATCH-ABORT	7	PROCEED-CASE	7	WITH-CONDITION-RESTARTS	7
CONDITION-CASE	7	RESTART-CASE	7		

DEFINER INDEX

DEFINE-CONDITION	7	DEFINE-PROCEED-FUNCTION	10
------------------------	---	-------------------------------	----

PROPERTY INDEX

IL:CL-ERROR	12
-------------------	----
