File created: 28-Apr-88 18:39:29 **{ERIS}<CUTTING>RPC>RPCXDR.;6**

changes to: (IL:FUNCTIONS XDR-STRING-POINTER)
(IL:VARS IL:RPCXDRCOMS)
(FILE-ENVIRONMENTS IL:RPCXDR)

previous date: 28-Apr-88 18:34:44 {ERIS}<CUTTING>RPC>RPCXDR.;5

Read Table: XCL

Package: RPC2

Format: XCCS


```
(IL:RPAQQ IL:RPCXDRCOMS
          ((IL:PROPS (IL:RPCXDR IL:MAKEFILE-ENVIRONMENT IL:FILETYPE))

;;; Useful Constants

           (IL:VARIABLES TWOTO31MINUSONE TWOTO31ST TWOTO32ND TWOTO32MINUSONE TWOTO63MINUSONE TWOTO64MINUSONE
                  TWOTO64TH MINUS2TO31 MINUS2TO63)
           (IL:VARIABLES *XDR-PRIMITIVE-TYPES* *XDR-CONSTRUCTED-TYPES* *XDR-CODEGEN-RECURSIVELST*)
           (IL:STRUCTURES TYPSTK)
                                                                  ; Miscellaneous XDR Utility Functions
           (IL:FUNCTIONS ACCESS-FCN-NAME CONSTRUCTOR-FCN-NAME FIND-IN-TYPE-STACK)

;;; Type Declarations and Predicates

           (IL:TYPES XDR-INTEGER XDR-UNSIGNED XDR-HYPERINTEGER XDR-HYPERUNSIGNED)
           (IL:FUNCTIONS XDR-INTEGER-P XDR-UNSIGNED-P XDR-HYPERINTEGER-P XDR-HYPERUNSIGNED-P)

;;; XDR Code Generation for Constructed Functions

           (IL:FUNCTIONS XDR-CODEGEN-COMMENT XDR-CODEGEN XDR-CODEGEN-1 XDR-CODEGEN-2 XDR-CODEGEN-3
                  XDR-CODEGEN-RECURSION XDR-CODEGEN-PRIMITIVE XDR-CODEGEN-INHERITED XDR-CODEGEN-QUALIFIED
                  XDR-CODEGEN-LOCAL XDR-CODEGEN-CONSTRUCTED XDR-CODEGEN-CONSTANT XDR-CODEGEN-ENUMERATION
                  XDR-CODEGEN-UNION XDR-CODEGEN-LIST XDR-CODEGEN-STRUCT XDR-CODEGEN-FIXED-ARRAY
                  XDR-CODEGEN-COUNTED-ARRAY XDR-CODEGEN-OPAQUE XDR-CODEGEN-SKIP XDR-CODEGEN-SEQUENCE)

;;; XDR PRIMITIVES

           (IL:FUNCTIONS XDR-BOOLEAN XDR-INTEGER XDR-UNSIGNED XDR-HYPERINTEGER XDR-HYPERUNSIGNED
                  XDR-OPAQUE-PRIMITIVE XDR-SKIP-PRIMITIVE XDR-STRING XDR-STRING-POINTER XDR-FLOAT XDR-VOID)))

(IL:PUTPROPS IL:RPCXDR IL:MAKEFILE-ENVIRONMENT (:READTABLE "XCL" :PACKAGE "RPC2"))

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

;;; Useful Constants

(DEFCONSTANT TWOTO31MINUSONE 2147483647
   "NIL")

(DEFCONSTANT TWOTO31ST 2147483648)

(DEFCONSTANT TWOTO32ND 4294967296
   "NIL")

(DEFCONSTANT TWOTO32MINUSONE 4294967295
   "NIL")

(DEFCONSTANT TWOTO63MINUSONE 9223372036854775807
   "NIL")

(DEFCONSTANT TWOTO64MINUSONE 18446744073709551615
   "NIL")

(DEFCONSTANT TWOTO64TH 18446744073709551616
   "NIL")

(DEFCONSTANT MINUS2TO31 -2147483648
   "NIL")
```

```
(DEFCONSTANT MINUS2TO63 -9223372036854775808
   "NIL")


(DEFPARAMETER *XDR-PRIMITIVE-TYPES*
   '((:INTEGER . XDR-INTEGER)
     (:BOOLEAN . XDR-BOOLEAN)
     (:UNSIGNED . XDR-UNSIGNED)
     (:HYPERINTEGER . XDR-HYPERINTEGER)
     (:HYPERUNSIGNED . XDR-HYPERUNSIGNED)
     (:STRING . XDR-STRING)
     (:VOID . XDR-VOID)
     (:FLOAT . XDR-FLOAT)
     (:DOUBLE . XDR-DOUBLE)
     (:STRING-POINTER . XDR-STRING-POINTER))
   "An alist of XDR primitive types and the function that encodes/decodes that type")


(DEFPARAMETER *XDR-CONSTRUCTED-TYPES*
   '((:ENUMERATION . XDR-CODEGEN-ENUMERATION)
     (:UNION . XDR-CODEGEN-UNION)
     (:STRUCT . XDR-CODEGEN-STRUCT)
     (:LIST . XDR-CODEGEN-LIST)
     (:FIXED-ARRAY . XDR-CODEGEN-FIXED-ARRAY)
     (:COUNTED-ARRAY . XDR-CODEGEN-COUNTED-ARRAY)
     (:OPAQUE . XDR-CODEGEN-OPAQUE)
     (:SKIP . XDR-CODEGEN-SKIP)
     (:SEQUENCE . XDR-CODEGEN-SEQUENCE))
   "
   Association list of XDR constructed types and the functions that create functions to read/write them")


(DEFGLOBALVAR *XDR-CODEGEN-RECURSIVELST* NIL
   "
   Place for XDR-CODEGEN to save recursive functions it found in making an expansion.
   A list of TYPSTK structs
   ")


(DEFSTRUCT TYPSTK
   "Element on stack of types for which code already generated."
   PROG
   TYPE
   XDRPROC
   OPER
   ARGS)
```

;; Miscellaneous XDR Utility Functions

```
(DEFUN ACCESS-FCN-NAME (STRUCT FIELD)
   "
   Maps struct name and field name (strings or symbols) into the
   access function name for that slot."                          ;
   (INTERN (CONCATENATE 'STRING (STRING STRUCT)
                  "-"
                  (STRING FIELD))
          (SYMBOL-PACKAGE STRUCT)))


(DEFUN CONSTRUCTOR-FCN-NAME (STRUCT)
   "
   Maps a symbol or string naming a defstruct into the constructor function symbol
   for that defstruct type"
   (INTERN (CONCATENATE 'STRING "MAKE-" (STRING STRUCT))
          (SYMBOL-PACKAGE STRUCT)))


(DEFUN FIND-IN-TYPE-STACK (PRG TYP STACK)
   "
   Find the first element in a list of TYPSTK's such that PRG and TYP
   match the PROG and TYPE fields of the TYPSTK.
   "
   (DOLIST (EL STACK)
       (IF (AND (EQL PRG (TYPSTK-PROG EL))
                (EQL TYP (TYPSTK-TYPE EL)))
           (RETURN EL))))
```

;;; Type Declarations and Predicates

```
(DEFTYPE XDR-INTEGER ()
   '(AND INTEGER (SATISFIES XDR-INTEGER-P)))
```

```
(DEFTYPE XDR-UNSIGNED ()
   '(AND INTEGER (SATISFIES XDR-UNSIGNED-P)))

(DEFTYPE XDR-HYPERINTEGER ()
   '(AND INTEGER (SATISFIES XDR-HYPERINTEGER-P)))

(DEFTYPE XDR-HYPERUNSIGNED ()
   '(AND INTEGER (SATISFIES XDR-HYPERUNSIGNED-P)))

(DEFUN XDR-INTEGER-P (I)
   (AND (>= I MINUS2TO31)
        (< I TWOTO31ST)))

(DEFUN XDR-UNSIGNED-P (I)
   (OR (AND (TYPEP I 'FIXNUM)
            (>= (THE FIXNUM I)
                0))
       (AND (>= I 0)
            (< I TWOTO32ND))))

(DEFUN XDR-HYPERINTEGER-P (I)
   (AND (>= I MINUS2TO63)
        (<= I TWOTO63MINUSONE)))

(DEFUN XDR-HYPERUNSIGNED-P (I)
   (AND (>= I 0)
        (<= I TWOTO64MINUSONE)))
```

;;; XDR Code Generation for Constructed Functions

```
(DEFUN XDR-CODEGEN-COMMENT ()
   "
   ************************************************
   **** Code Generation for XCL Constructed Types ****
   ************************************************
   The following functions generate code for translating between Common Lisp
   and XDR. For each function,
       CONTEXT is an RPC-PROGRAM structure with respect to which a
               typedef is being constructed.
       TYPEDEF is an XDR type definition, and
       OPER is either READ (decode) or WRITE (encode).
   For all functions except XDR-CODEGEN, a third argument ARGS is a list of
   arguments to the code being generated. It always begins with an XDR-stream argument
   and for OPER=WRITE is usually followed by the object to be written.

   WARNINGS:

   (1) DO NOT, REPEAT DO NOT pass an (XDR-CODEGEN-xxx) as the argument of an
   (XDR-CODEGEN-xxx). If you do, you might cause the  code generated for
   the argument to be evaluated multiple times in the code for the resulting
   expression.

   (2) The XDR-CODEGEN-xxx functions code in-line rather than wrap themselves
   in LET's or LAMBDA's or whatever. To avoid complications with functions
   that require a location-specifier (CHECK-TYPE or CCASE, for example), an
   XDR-CODEGEN-xxx function may ***not*** generate code that assumes that its
   arguments ARGS or various COUNTs are legitimate location-specifiers. If a
   CHECK-TYPE or similar function is to be done, a LET (or other binding
   mechanism) should be generated to create a legal location-specifier."
   NIL)

(DEFUN XDR-CODEGEN (CONTEXT TYPEDEF OPER)
   "
   Top-level XDR Code Generation function. Returns code to read/write
   an XDR element of type TYPEDEF.

   CONTEXT is an RPC-PROGRAM structure with respect to which the
   TYPEDEF is interpreted (in terms of inheritance).

   TYPEDEF is an XDR Type or Type definition.

   OPER is either 'RPC2::READ or 'RPC::WRITE.

   See documentation of XDR-CODEGEN-COMMENT.
   "
   (SETQ *XDR-CODEGEN-RECURSIVELST* NIL)
   (LET* ((ARGS (ECASE OPER
                  (READ '(XDR-STREAM))
```

```
                    (WRITE '(XDR-STREAM XDR-TOPLEVEL-ITEM))))
          (FCN (XDR-CODEGEN-1 CONTEXT TYPEDEF OPER ARGS NIL)))
         (IF FCN
             (IF (NULL *XDR-CODEGEN-RECURSIVELST*)
                 (LIST 'LAMBDA ARGS FCN)
                 (LIST 'LAMBDA ARGS `(LABELS ,(XDR-CODEGEN-3 *XDR-CODEGEN-RECURSIVELST*)
                                             ,FCN)))
             (ERROR "Could not parse XDR Type ~S" TYPEDEF))))

(DEFUN XDR-CODEGEN-1 (CONTEXT TYPEDEF OPER ARGS STK)
   "
   Generates code to read or write an element of type TYPEDEF.

   CONTEXT, TYPEDEF, and OPER are as in XDR-CODEGEN.

   ARGS is a list of the arguments forms for the generated code.
   For OPER=READ it will (<rpc-stream-name>), and
   For OPER=WRITE it will be (<rpc-stream-name> <element>).

   STK is a list of TYPSTK elements, one for each named type above
   this one in this expansion.
   "
   (OR (XDR-CODEGEN-PRIMITIVE CONTEXT TYPEDEF OPER ARGS STK)
       (XDR-CODEGEN-CONSTRUCTED CONTEXT TYPEDEF OPER ARGS STK)
       (XDR-CODEGEN-LOCAL CONTEXT TYPEDEF OPER ARGS STK)
       (XDR-CODEGEN-INHERITED CONTEXT TYPEDEF OPER ARGS STK)
       (XDR-CODEGEN-QUALIFIED CONTEXT TYPEDEF OPER ARGS STK)
       (ERROR "Could not resolve XDR Type Definition: ~a" TYPEDEF)))

(DEFUN XDR-CODEGEN-2 (CONTEXT TYPENAME OPER ARGS STK)
   "
   Expands named types.

   (1) Sees whether type already seen above here in this expansion.
   Otherwise,
   (2) Notes the name on TYPESTK,
   (3) Finds the definition of this type,
   (4) Calls XDR-CODEGEN-1 to expand the type definition.
   (5) Sees whether the XDR-CODEGEN-1 call found this type below,
        if so, notes this on *XDR-CODEGEN-RECURSIVELST* and returns
        call to the recursive function for this type.
        otherwise just returns the code.
   "
   ;; Every named type expansion passes through here and gets expanded. Since it is only named types that can be recursive,  this is the only place
   ;; we check for recursion
   (OR (XDR-CODEGEN-RECURSION CONTEXT TYPENAME OPER ARGS STK)
       (LET (TD CODE TOP)                                          ; No
            (PUSH (MAKE-TYPSTK :PROG CONTEXT :TYPE TYPENAME :OPER OPER :ARGS (IF (EQL OPER 'READ)
                                                                                 ARGS
                                                                                 '(RPCSTREAM RVALUE)))
                  STK)                                           ; Push type on stack
            (UNLESS (SETQ TD (FIND-RPC-TYPEDEF CONTEXT TYPENAME))
                (ERROR "Null type definition for Program ~A, Type ~A" (AND CONTEXT (RPC-PROGRAM-NAME CONTEXT))
                       TYPENAME))
            (SETQ CODE (XDR-CODEGEN-1 CONTEXT TD OPER ARGS STK))
                                                                   ; Generate code
            (SETQ TOP (CAR STK))                                   ; "Pop" stack
            (IF (NULL (TYPSTK-XDRPROC TOP))                        ; Was this type called recursively?
                CODE                                               ; No, just return code
                (PROGN (PUSH TOP *XDR-CODEGEN-RECURSIVELST*)       ; Yes, save recursive type
                       `(,(TYPSTK-XDRPROC TOP)
                         ,@ARGS)                                   ; Return call to recursive function

                       )))))

(DEFUN XDR-CODEGEN-3 (RLIST)
   ;; Generate the set of function definitions for LABELS. RLIST is a list of TYPSTK structs. .
   (MAP 'LIST #'(LAMBDA (TYPSTK)
                      `(,(TYPSTK-XDRPROC TYPSTK)
                        ,(TYPSTK-ARGS TYPSTK)
                        ,(XDR-CODEGEN-1 (TYPSTK-PROG TYPSTK)
                                (OR (FIND-RPC-TYPEDEF (TYPSTK-PROG TYPSTK)
                                          (TYPSTK-TYPE TYPSTK))
                                    (ERROR "No typedef for Program ~A, Type ~A" (RPC-PROGRAM-NAME (TYPSTK-PROG
                                                                                                   TYPSTK))
                                          (TYPSTK-TYPE TYPSTK)))
                                (TYPSTK-OPER TYPSTK)
                                (TYPSTK-ARGS TYPSTK)
                                RLIST)))
        RLIST))
```

```
(DEFUN XDR-CODEGEN-RECURSION (PRG TYP OPER ARGS STACK)
   ;; If type has already be seen, mark as recursive and return code calling that function
   (LET ((INSTACK (FIND-IN-TYPE-STACK PRG TYP STACK)))
        (WHEN INSTACK                                                    ; Seen it before
            (SETF (TYPSTK-XDRPROC INSTACK)
                  (OR (TYPSTK-XDRPROC INSTACK)
                      (INTERN (SYMBOL-NAME (GENSYM (CONCATENATE 'STRING "XDR-" (SYMBOL-NAME OPER)
                                                                "-"
                                                                (SYMBOL-NAME TYP)
                                                                "-"))))))
            `(,(TYPSTK-XDRPROC INSTACK)
              ,@ARGS))))

(DEFUN XDR-CODEGEN-PRIMITIVE (CONTEXT TYPEDEF OPER ARGS STK)
   "NIL"
   (LET (FCN)
        (IF (AND (SYMBOLP TYPEDEF)
                 (SETQ FCN (CDR (ASSOC TYPEDEF *XDR-PRIMITIVE-TYPES*))))
            `(,FCN ,@ARGS))))

(DEFUN XDR-CODEGEN-INHERITED (CONTEXT TYPEDEF OPER ARGS STK)
   "NIL"
   (AND (SYMBOLP TYPEDEF)
        (SOME #'(LAMBDA (PROGNAME)
                        (LET* ((PRG (FIND-RPC-PROGRAM :NAME PROGNAME))
                               (TD (FIND-RPC-TYPENAME PRG TYPEDEF)))
                              (IF (AND PRG TD)
                                  (XDR-CODEGEN-2 PRG TD OPER ARGS STK))))
              (RPC-PROGRAM-INHERITS CONTEXT))))

(DEFUN XDR-CODEGEN-QUALIFIED (CONTEXT TYPEDEF OPER ARGS STK)
   "NIL"
   (IF (AND (CONSP TYPEDEF)
            (SYMBOLP (CAR TYPEDEF))
            (SYMBOLP (CDR TYPEDEF)))
       (LET* ((PRG (FIND-RPC-PROGRAM :NAME (CAR TYPEDEF)))
              (TD (FIND-RPC-TYPEDEF PRG (CDR TYPEDEF))))
             (IF (AND PRG TD)
                 (XDR-CODEGEN-2 PRG TD OPER ARGS STK)
                 (ERROR "Could not find qualified XDR definition ~A from RPC program ~A" (CDR TYPEDEF)
                        (CAR TYPEDEF))))))

(DEFUN XDR-CODEGEN-LOCAL (CONTEXT TYPEDEF OPER ARGS STK)
   "NIL"
   (IF (SYMBOLP TYPEDEF)
       (LET ((TD (FIND-RPC-TYPENAME CONTEXT TYPEDEF)))
            (IF TD (XDR-CODEGEN-2 CONTEXT TD OPER ARGS STK)))))

(DEFUN XDR-CODEGEN-CONSTRUCTED (CONTEXT TYPEDEF OPER ARGS STK)
   "NIL"
   (LET (FCN)
        (IF (AND (CONSP TYPEDEF)
                 (SETQ FCN (CDR (ASSOC (CAR TYPEDEF)
                                       *XDR-CONSTRUCTED-TYPES*))))
            (FUNCALL FCN CONTEXT TYPEDEF OPER ARGS STK))))

(DEFUN XDR-CODEGEN-CONSTANT (CONTEXT CONSTANT)
   (COND
      ((NULL CONSTANT)
       (ERROR "Could not resolve nil constant definition from RPC program ~a~%" (RPC-PROGRAM-NAME CONTEXT)))
      ((INTEGERP CONSTANT)
       CONSTANT)                                                         ; Immediate Constant Definition
      ((AND (SYMBOLP CONSTANT)
            (OR (FIND-XDR-CONSTANT CONTEXT CONSTANT)                     ; Local Constant Definition
                (SOME #'(LAMBDA (CNTX)
                                (FIND-XDR-CONSTANT (FIND-RPC-PROGRAM :NAME CNTX)
                                                   CONSTANT))
                      (RPC-PROGRAM-INHERITS CONTEXT))                    ; Inherited Constant Definition

                )))
      ((AND (CONSP CONSTANT)                                             ; Qualified Constant Definition
            (SYMBOLP (CDR CONSTANT))
            (FIND-XDR-CONSTANT (FIND-RPC-PROGRAM :NAME (CAR CONSTANT))
                  (CDR CONSTANT))))
      ((ERROR "Could not resolve XDR constant ~a~%" CONSTANT))))

(DEFUN XDR-CODEGEN-ENUMERATION (CONTEXT TYPEDEF OPER ARGS STK)
   "NIL"
   (LET ((ALIST (MAP 'LIST #'(LAMBDA (X)
```

```
                                    (CONS (CAR X)
                                          (XDR-CODEGEN-CONSTANT CONTEXT (CADR X))))
                     (CDR TYPEDEF))))
        (IF (EQL OPER 'READ)
            `(CAR (RASSOC (XDR-INTEGER ,(CAR ARGS))
                          ',ALIST))
            `(XDR-INTEGER ,(CAR ARGS)
                    (CDR (ASSOC ,(CADR ARGS)
                                ',ALIST))))))


(DEFUN XDR-CODEGEN-UNION (CONTEXT TYPEDEF OPER ARGS STK)
   "
   (UNION <discriminant-type> (<enumeration-element> <arm-type>) ...(<> <>))
   Read Calling Sequence: XDR-UNION(xdrstream)
   Read Input: An integer followed by the encoding of that arm.
   Read Output: The enumeration element from the type of the discriminant
                The discriminant and arm are returned as a dotted pair.
   Write Input: An enumeration element and an unencoded arm.
   Write calling sequence: XDR-UNION(xdrstream,discriminant,arm)
   Write Output: The (integer) encoding of the discriminant and the encoded arm.
   "
   (LET
    ((DISCRIM-TYPE (SECOND TYPEDEF))
     (XDRSTREAM (FIRST ARGS))
     (UNIONLIST (SECOND ARGS)))
    (IF (EQL OPER 'READ)
        `(LET (DISCRIMINANT)
              (SETQ DISCRIMINANT ,(XDR-CODEGEN-1 CONTEXT DISCRIM-TYPE OPER ARGS STK))
              (LIST DISCRIMINANT (CASE DISCRIMINANT
                                       (IL:\\\,.
                                          (DO ((PAIRS (CDDR TYPEDEF)
                                                      (CDR PAIRS))
                                               (ARMS)
                                               (PAIR))
                                              ((NULL PAIRS)
                                               (NREVERSE ARMS))
                                             (SETQ PAIR (FIRST PAIRS))
                                             (PUSH `(,(IF (EQL (CAR PAIR)
                                                               'DEFAULT)
                                                          'OTHERWISE
                                                          `(,(CAR PAIR)))
                                                        ,(XDR-CODEGEN-1 CONTEXT (CADR PAIR)
                                                                        OPER ARGS STK))
                                                   ARMS))))))
        `(PROGN ,(XDR-CODEGEN-1 CONTEXT DISCRIM-TYPE OPER `(,XDRSTREAM (CAR ,UNIONLIST))
                        STK)
                (CASE (CAR ,UNIONLIST)
                    (IL:\\\,. (DO ((PAIRS (CDDR TYPEDEF)
                                          (CDR PAIRS))
                                   (ARMS)
                                   (PAIR))
                                  ((NULL PAIRS)
                                   (NREVERSE ARMS))
                                 (SETQ PAIR (CAR PAIRS))
                                 (PUSH `(,(IF (EQL (CAR PAIR)
                                                   'DEFAULT)
                                              'OTHERWISE
                                              `(,(CAR PAIR)))
                                            ,(XDR-CODEGEN-1 CONTEXT (CADR PAIR)
                                                    OPER
                                                    `(,XDRSTREAM (CADR ,UNIONLIST))
                                                    STK))
                                       ARMS))))))))


(DEFUN XDR-CODEGEN-LIST (CONTEXT TYPEDEF OPER ARGS STK)
   "TYPEDEF = (LIST <typedef-1> ... <typedef-n>)"
   (IF (EQL OPER 'READ)
       `(LIST ,.(MAP 'LIST #'(LAMBDA (TD)
                                     (XDR-CODEGEN-1 CONTEXT TD OPER ARGS STK))
                     (CDR TYPEDEF)))
       (LET ((XDRSTREAM (FIRST ARGS))
             (THELIST (SECOND ARGS)))
            `(PROGN ,.(DO ((TD (CDR TYPEDEF)
                               (CDR TD))
                           (INDX 0 (+ 1 INDX))
                           (CODE))
                          ((NULL TD)
                           (NREVERSE CODE))
                         (PUSH (XDR-CODEGEN-1 CONTEXT (CAR TD)
                                              OPER
                                              `(,XDRSTREAM (ELT ,THELIST ,INDX))
                                              STK)
                               CODE))))))
```

```
(DEFUN XDR-CODEGEN-STRUCT (CONTEXT TYPEDEF OPER ARGS STK)
   "(STRUCT <defstruct-type> (<field-name> <type>) ... (<field-name> <type>))"
   (LET ((STRUCT-TYPE (CADR TYPEDEF))
         (XDRSTREAM (FIRST ARGS))
         (THESTRUCT (SECOND ARGS)))
        (IF (EQL OPER 'READ)
            (LET ((NEWSTRUCT (INTERN (SYMBOL-NAME (GENSYM "XDR-")))))
                 `(LET ((,NEWSTRUCT (,(CONSTRUCTOR-FCN-NAME STRUCT-TYPE))))
                       ,@(MAP 'LIST #'(LAMBDA (X)
                                              `(SETF (,(ACCESS-FCN-NAME STRUCT-TYPE (CAR X))
                                                      ,NEWSTRUCT)
                                                     ,(XDR-CODEGEN-1 CONTEXT (CADR X)
                                                             OPER ARGS STK)))
                              (CDDR TYPEDEF))
                       ,NEWSTRUCT))
                 `(PROGN ,@(MAP 'LIST #'(LAMBDA (X)
                                                 (XDR-CODEGEN-1 CONTEXT (CADR X)
                                                        OPER
                                                        `(,XDRSTREAM (,(ACCESS-FCN-NAME STRUCT-TYPE (CAR X))
                                                                     ,THESTRUCT))
                                                        STK))
                                (CDDR TYPEDEF))))))


(DEFUN XDR-CODEGEN-FIXED-ARRAY (CONTEXT TYPEDEF OPER ARGS STK &OPTIONAL DONT-RESOLVE-COUNT)
   "typedef is (fixed-array elttype count)"                    ;
   "typedef is (fixed-array elttype count)"
   (LET* ((ELEMENT-TYPE (SECOND TYPEDEF))
          (COUNT (IF DONT-RESOLVE-COUNT
                     (THIRD TYPEDEF)                           ; This hack enables XDR-CODEGEN-FIXED-ARRAY to be used
                                                               ; by XDR-CODEGEN-COUNTED-ARRAY. Normally, the count
                                                               ; must be resolvable at codegen-time, but when called by
                                                               ; XDR-CODEGEN, COUNT is an expression that needs to be
                                                               ; evaluated at run time.
                     (XDR-CODEGEN-CONSTANT CONTEXT (THIRD TYPEDEF))))
          (XDRSTREAM (FIRST ARGS))
          (THEARRAY (SECOND ARGS))
          (NEWARRAY (INTERN (SYMBOL-NAME (GENSYM "XDR-ARRAY-"))))
          (THECOUNT (INTERN (SYMBOL-NAME (GENSYM "XDR-COUNT-")))))
         (IF (EQL OPER 'READ)
             `(LET ((,NEWARRAY (MAKE-ARRAY ,COUNT))
                    (,THECOUNT ,COUNT))
                   (CHECK-TYPE ,THECOUNT (INTEGER 0 *))
                   (DOTIMES (I ,THECOUNT ,NEWARRAY)
                       (SETF (ELT ,NEWARRAY I)
                             ,(XDR-CODEGEN-1 CONTEXT ELEMENT-TYPE OPER ARGS STK))))
             `(LET ((,THECOUNT ,COUNT))
                   (CHECK-TYPE ,THECOUNT (INTEGER 0 *))
                   (DOTIMES (I ,THECOUNT ,THEARRAY)
                       ,(XDR-CODEGEN-1 CONTEXT ELEMENT-TYPE OPER `(,XDRSTREAM (ELT ,THEARRAY I))
                                STK))))))


(DEFUN XDR-CODEGEN-COUNTED-ARRAY (CONTEXT TYPEDEF OPER ARGS STK)
   "typedef is (fixed-array element-type)"                     ;
   "typedef is (fixed-array element-type)"
   (LET ((ELEMENT-TYPE (SECOND TYPEDEF))
         (XDRSTREAM (FIRST ARGS))
         (THEARRAY (SECOND ARGS)))
        (IF (EQL OPER 'READ)
            `(LET ((THECOUNT ,(XDR-CODEGEN-1 CONTEXT :UNSIGNED OPER ARGS STK)))
                  (LIST THECOUNT ,(XDR-CODEGEN-FIXED-ARRAY CONTEXT `(:FIXED-ARRAY ,ELEMENT-TYPE THECOUNT)
                                        OPER ARGS STK T)))
            `(LET ((THECOUNT (LENGTH ,THEARRAY)))
                  ,(XDR-CODEGEN-1 CONTEXT :UNSIGNED OPER `(,XDRSTREAM THECOUNT)
                           STK)
                  ,(XDR-CODEGEN-FIXED-ARRAY CONTEXT `(:FIXED-ARRAY ,ELEMENT-TYPE THECOUNT)
                           OPER ARGS STK T))))))


(DEFUN XDR-CODEGEN-OPAQUE (CONTEXT TYPEDEF OPER ARGS STK)
   "Declaration is (opaque <bytecount>)"
   (LET ((BYTECOUNT (XDR-CODEGEN-CONSTANT CONTEXT (SECOND TYPEDEF)))
         (XDRSTREAM (FIRST ARGS))
         (BYTESTRING (SECOND ARGS)))
        (CHECK-TYPE BYTECOUNT (INTEGER 0 *))
        (IF (EQL OPER 'READ)
            `(XDR-OPAQUE-PRIMITIVE ,XDRSTREAM ,BYTECOUNT)
            `(XDR-OPAQUE-PRIMITIVE ,XDRSTREAM ,BYTECOUNT ,BYTESTRING))))


(DEFUN XDR-CODEGEN-SKIP (CONTEXT TYPEDEF OPER ARGS STK)
   (LET ((BYTECOUNT (XDR-CODEGEN-CONSTANT CONTEXT (SECOND TYPEDEF)))
         (XDRSTREAM (FIRST ARGS))
         (BYTESTRING (SECOND ARGS)))
        (CHECK-TYPE BYTECOUNT (INTEGER 0 *))
        (IF (EQL OPER 'READ)
```

```
            `(XDR-SKIP-PRIMITIVE ,XDRSTREAM ,BYTECOUNT)
            `(XDR-SKIP-PRIMITIVE ,XDRSTREAM ,BYTECOUNT ,BYTESTRING))))
```

```
(DEFUN XDR-CODEGEN-SEQUENCE (CONTEXT TYPEDEF OPER ARGS STK)
   (LET ((STREAM (FIRST ARGS))
         (ELEMENTS (SECOND ARGS))
         (ELTTYPE (SECOND TYPEDEF)))
        (IF (EQL OPER 'READ)
            `(DO ((ITEMS))
                 ()
                (ECASE (XDR-BOOLEAN ,STREAM)
                    ((NIL) (RETURN (NREVERSE ITEMS)))
                    ((T) (PUSH ,(XDR-CODEGEN-1 CONTEXT ELTTYPE OPER ARGS STK)
                               ITEMS))))
            `(DOLIST (EL ,ELEMENTS (XDR-BOOLEAN ,STREAM NIL))
                 (XDR-BOOLEAN ,STREAM T)
                 ,(XDR-CODEGEN-1 CONTEXT ELTTYPE OPER `(,STREAM EL)
                         STK)))))
```

;;; XDR PRIMITIVES

```
(DEFUN XDR-BOOLEAN (XDRSTREAM &OPTIONAL (VALUE T WRITEP))
   "NIL"
   (IF WRITEP
       (PUTCELL XDRSTREAM (IF (NULL VALUE)
                              0
                              1))
       (PROGN (SETQ VALUE (GETCELL XDRSTREAM))
              (CCASE VALUE (0 NIL)
                     (1 T)))))
```

```
(DEFUN XDR-INTEGER (XDRSTREAM &OPTIONAL VALUE)
   "NIL"
   (IF VALUE
       (PROGN (CHECK-TYPE VALUE INTEGER)
              (PUTCELL XDRSTREAM VALUE))
       (GETCELL XDRSTREAM)))
```

```
(DEFUN XDR-UNSIGNED (XDRSTREAM &OPTIONAL VALUE)
   "NIL"
   (IF VALUE
       (PUTUNSIGNED XDRSTREAM VALUE)
       (GETUNSIGNED XDRSTREAM)))
```

```
(DEFUN XDR-HYPERINTEGER (XDRSTREAM &OPTIONAL (VALUE T WRITEP))
   "NIL"
   (IF WRITEP
       (PROGN (CHECK-TYPE VALUE XDR-HYPERINTEGER)
              (IF (MINUSP VALUE)
                  (SETQ VALUE (+ TWOTO64TH VALUE)))
              (PUTUNSIGNED XDRSTREAM (ASH VALUE -32))
              (PUTUNSIGNED XDRSTREAM (LOGAND VALUE TWOTO32MINUSONE)))
       (PROGN (SETQ VALUE (+ (ASH (GETUNSIGNED XDRSTREAM)
                                  32)
                             (GETUNSIGNED XDRSTREAM)))
              (IF (> VALUE TWOTO63MINUSONE)
                  (SETQ VALUE (- VALUE TWOTO64TH))
                  VALUE))))
```

```
(DEFUN XDR-HYPERUNSIGNED (XDRSTREAM &OPTIONAL (VALUE T WRITEP))
   "NIL"
   (IF WRITEP
       (PROGN (CHECK-TYPE VALUE XDR-HYPERUNSIGNED)
              (PUTUNSIGNED XDRSTREAM (ASH VALUE -32))
              (PUTUNSIGNED XDRSTREAM (LOGAND VALUE TWOTO32MINUSONE)))
       (+ (ASH (GETUNSIGNED XDRSTREAM)
               32)
          (GETUNSIGNED XDRSTREAM))))
```

```
(DEFUN XDR-OPAQUE-PRIMITIVE (XDRSTREAM N &OPTIONAL (VALUE T WRITEP))
   "NIL"
   (IF WRITEP
       (PROGN (CHECK-TYPE N (INTEGER 0 *))
              (PUTBYTES XDRSTREAM VALUE)
              (SETQ N (LOGAND N 3))
              (DOTIMES (I (CASE N
                                ((0 2) N)
                                (1 3)
                                (3 1)))
```

```
                (PUTBYTE XDRSTREAM 0)))
        (PROG1 (SETQ VALUE (GETBYTES XDRSTREAM N))
            (SETQ N (LOGAND N 3))
            (DOTIMES (I (CASE N
                             ((0 2) N)
                             (1 3)
                             (3 1)))
                (GETBYTE XDRSTREAM)))))

(DEFUN XDR-SKIP-PRIMITIVE (XDRSTREAM N &OPTIONAL (VALUE T WRITEP))
   (IF WRITEP
       (ERROR "SKIP is currently defined for input only.")
       (PROGN (SETQ VALUE (GETOFFSET XDRSTREAM))
              (PUTOFFSET XDRSTREAM (+ N (CDR VALUE)))
              N)))

(DEFUN XDR-STRING (XDRSTREAM &OPTIONAL VALUE)
   "NIL"
   (IF VALUE
       (LET ((NBYTES (LENGTH VALUE)))
            (XDR-UNSIGNED XDRSTREAM NBYTES)
            (XDR-OPAQUE-PRIMITIVE XDRSTREAM NBYTES VALUE))
       (XDR-OPAQUE-PRIMITIVE XDRSTREAM (XDR-UNSIGNED XDRSTREAM))))

(DEFUN XDR-STRING-POINTER (XDRSTREAM &OPTIONAL (VALUE T WRITEP))
   "This is a hack to avoid copying 512 byte VMEMPAGEP's"
   (IF WRITEP
       (LET ((BUFFER (CAR VALUE))
             (NBYTES (CDR VALUE))
             (OUTSTREAM (RPC-STREAM-OUTSTREAM XDRSTREAM))
             (MOD4))
            ;; This only works for UDP!!
            (XDR-UNSIGNED XDRSTREAM NBYTES)
            (IF *USE-OS-NETWORKING*
                (IL:\\MOVEBYTES BUFFER 0 (IL:LOCF (IL:|fetch| (XDR-DATA-BLOCK XDR-PUBLIC)
                                                      IL:|of| (SETQ OUTSTREAM (IL:\\DTEST OUTSTREAM
                                                                                          'XDR-DATA-BLOCK))))
                                  (RPC-STREAM-OUTBYTEPTR XDRSTREAM)
                                  NBYTES)
                (IL:UDP.APPEND.BYTES OUTSTREAM BUFFER 0 NBYTES))
            (IF (NOT (= 0 (SETQ MOD4 (LOGAND 3 NBYTES))))        ; Pad to multiple of 4 with zeros.
                (DOTIMES (I (CASE MOD4
                                  ((0 2) MOD4)
                                  (1 3)
                                  (3 1)))
                    (PUTBYTE OUTSTREAM 0))))
       (LET* ((NBYTES (XDR-UNSIGNED XDRSTREAM))
              (PLACE (GETOFFSET XDRSTREAM))
              (PACKET (CAR PLACE))
              (BYTEOFFSET (CDR PLACE)))
             ;; Returns ((packet . byteoffset) . number-of-bytes))
             ;; Note that this does NOT update rpcstream ponter.
             (CONS (CONS PACKET BYTEOFFSET)
                   NBYTES))))

(DEFUN XDR-FLOAT (S &OPTIONAL (V T WRITEP))
   "NIL"
   (ERROR "Not yet implemented"))

(DEFUN XDR-VOID (XDRSTREAM &OPTIONAL (VALUE T WRITEP))
   NIL)

(IL:PUTPROPS IL:RPCXDR IL:COPYRIGHT ("Stanford University and Xerox Corporation" 1987 1988))
```

FUNCTION INDEX

CONSTANT INDEX

TYPE INDEX

VARIABLE INDEX

STRUCTURE INDEX

PROPERTY INDEX