RPC

SUN REMOTE PROCEDURE CALLS

By: JFinger

Supported by Atty Mullins (Mullins.pa@Xerox.com) and Bill Van Melle (vanMelle.pa@Xerox.com).

This document last edited on August 1, 1988.

INTRODUCTION

This module implements SUN remote procedure calls as specified in the Remote Procedure Call Protocol Specification. The syntax is oriented toward Lisp users, differing greatly from Sun's C-like syntax.

RPC2 Package

All functions and variables mentioned in this document are defined as external variables in the package RPC2, unless otherwise stated.

REMOTE PROCEDURE DEFINITITION

Remote programs are defined via calls to define-remote-program.

define-remote-program	name number version protocol &key :constants [Function] :types :inherits :procedures Defines parameters and result types of the procedures of remote program (number, version, protocol) . If successful, returns name, otherwise nil.
name	a string or symbol that may be used by other procedures (for example, remote-procedure-call) to uniquely specify this remote program.
number	is the program number of this program on the remote machine. As specified in Sun's Remote Procedure Call Programming Guide, programs 0 - #x1fffffff are defined by Sun, #x20000000 - #x3fffffff are reserved for users, and #x40000000 - #x5fffffff are designated as transient.
version	a number, is the desired version of remote program.
protocol	an atom,UDP or TCP.(At the moment TCP is not supported under Medley 1.0-S).
constants	a list of pairs (<constant-name> <constant-def>), where <constant-name> is a symbol and a <constant-def> is an XDR constant (See XDR Constant Definitions below) .
inherits	a list of name 's of other remote programs from which types and constants are inherited. Inherited types and constants are resolved by searching this list in order.
types	a list of pairs (<type-name> <typedef>) , where a <type-name> is a symbol and a <typedef> is an XDR type definition (defined below).

procedures — a list of 4-tuples of the form (<procname> <procnumber> <arg-types> <result-types>), where <procname> is a symbol or string naming the procedure, <procnumber> is the procedure number on the remote machine, <arg-types> is a (possibly empty) list of XDR type definitions (see below) of the arguments to be sent to the remote procedure, and <result-types> is a (possibly empty) list of XDR type definitions of data to be returned from this remote procedure.

XDR (EXTERNAL DATA REPRESENTATION) TYPE DEFINITIONS

Because the client and server machines may represent data in different ways, a data representation common to both machines is necessary Remote procedure calls pass data between machines in 'External Data Representation' (XDR). The XDR language implemented here is oriented toward Lisp in its syntax and is not identical to the language spelled out in the Sun XDR Protocol Specification.

XDR data types may be defined in the :types keyword argument for later reference in the :types or :procedures of this or later remote programs. When a remote program is defined (usually at load time), the needed reading and writing functions are compiled for each constructed type referenced. Note that all XDR calls are eventually resolved to a composition of Primitive and Constructed XDR Type Definitions (see below).

SYNTAX

The keywords of the XDR language may be specified as symbols of the Keyword package.

All XDR Data Types Definitions (notated here as a <typedef>), used in Remote Procedure Calls are from the following language:

1) **Primitive Definition**: One of the types in *xdr-primitive-types*,
:integer
:boolean
:unsigned
:hyperinteger
:hyperunsigned
:string
:float (not yet implemented)
:double (not yet implemented).

2) **Constructed Definition**:
One of the types in *xdr-constructed-types*,
(:enumeration (<symbol-1> <constant-1>) ...
(<symbol-n> <constant-n>))
(:union <enumeration-type> <typedef-1> ... <typedef-n>)
(:fixed-array <typedef> <constant>)
(:counted-array <typedef>)
(:opaque <constant>)
(:struct <defstruct-type> (<field-name-1> <typedef-1>) ...
(<field-name n> <typedef-n>))
(:sequence <typedef>)
(:list <typedef-1> ... <typedef-n>).

3) **Local Definition**: A symbol defined previously in the same remote program definition.

Example: :types ((nrec :unsigned)...) says that type 'nrec' is really only of type ':unsigned'.

4) **Qualified Definition**: A dotted pair of the form (<RPC program name> . <type>), where <type> is an XDR type local to <RPC program name>.

Example: :types ((count (myprog . nrec))...) says that a 'count' is really whatever myprog defines a 'nrec' to be.

5) **Inherited Definition**: A symbol defined in the :types argument of a remote program R such that R is on the list of remote programs passed as the :inherits argument to the current remote program definition. The first such type definition found is used, that is, the list of inherited programs is scanned from left to right.

XDR CONSTANT DEFINITIONS

Constants in XDR are defined by the following grammar:

<**constant-def**> ::= <**integer**> | <**defined-constant**>

<**defined constant**> ::= <**locally defined constant**>
; Defined in the Remote Program currently being defined.

| <**inherited constant**>
; Defined in a remote program inherited by the current Remote Program (searched from left to right).

| <**qualified constant**>
; A dotted pair (<rp> . <constant>), where <constant> is defined in remote program <rp>.

SEMANTICS

An XDR type can be defined by a bidirectional filter mapping a subset of Lisp onto a byte stream and vice-versa.

For the XDR primitive type's filter, a description is given of its argument on the Lisp and XDR sides.

:integer
Lisp: an integer in range -2,147,483,648 to 2,147,483,648 inclusive.
XDR: a 4 byte two's complement integer, high order to low order.

:unsigned
Lisp: an integer in range 0 to 4,294,967,295 inclusive.
XDR: a 4 byte non-negative integer, high order to low order.

:boolean
Lisp: NIL for false, non-NIL for true. (The Lisp symbol T is returned when decoding a 1 from the XDR side.)
XDR: 0 for false, 1 for true.

:hyperinteger
Lisp: an integer in range -(263) to 263 -1 inclusive.
XDR: a 8 byte two's complement integer, high order to low order.

:hyperunsigned
Lisp: an integer in range 0 to 264-1 inclusive.
XDR: a 8 byte non-negative integer, high order to low order.

:string
Lisp: a string of any length.
XDR: Suppose the string is of length n. The XDR representation is an :unsigned (the string length n) , followed by the n bytes of the string, followed by enough 0 bytes to make a multiple of 4 bytes.

:string-pointer
(UDP only)
Lisp: a dotted pair (addr . nbytes), where addr is a buffer's address and nbytes is the number of bytes in the buffer. (Should I add an offset

argument?). This is a speed hack to avoid having to copy VMEMPAGEP's twice.
XDR: An XDR :string, as above.

:float
Lisp: A floating point number. (NOT YET IMPLEMENTED).
XDR: A 4 byte floating point number in IEEE format.

:double
Lisp: A floating point number. (NOT YET IMPLEMENTED).
XDR: A double precision floating point number in IEEE format.

:void
Lisp: null
XDR: no bytes.

For each constructed XDR type, the declaration syntax is given along with its corresponding mapping.

(:enumeration (<symbol> <integer>) ... (<symbol> <integer>))
Lisp: a symbol
XDR: an XDR :integer.
The Lisp symbol (Each symbol is the "discriminant" for that value of the enumeration) and the XDR integer will be from a corresponding pair in the declaration. It is an error to try to encode a symbol not in the declaration or to try to decode an XDR integer for which there is not a corresponding symbol in the declaration.

(:union <enumeration-type> (<symbol-1> <typedef-1>) ... (<symbol-n> <typedef-n>))
Lisp: A list of two elements, the first being a discriminant for the enumeration type, and the second the appropriate Lisp input/output for the typedef corresponding to that discriminant's type..
XDR: An :integer discriminant followed by the XDR input/output for the typedef corresponding to that discriminant's type.

(:fixed-array <typedef> <constant>)
Lisp: An array of length <constant>, each element of which is an object of type <typedefLisp>. Note that since the function elt is used in encoding, any Lisp sequence could be used in place of an array.
XDR: A sequence of <constant> objects of type <typedefXDR>.

(:counted-array <typedef>)
Lisp: A list of two elements, the first of which is an integer (the number of objects to be encoded/decoded), and the second of which is an array of objects of type <typedefLisp>.
XDR: An integer (the number of objects to be encoded/decoded) followed by that number of objects of type <typedefXDR>.

(:opaque <constant>)
Lisp: A string of length <constant>.
XDR: A sequence of <constant> bytes followed by enough null bytes to round <constant> up to a multiple of four.

(:struct <defstruct-type> (<field-name-1> <typedef-1>) ...(<field-name n> <typedef-n>))

Lisp: A struct of type <defstruct-type> such that each field mentioned in the this XDR declaration has a value. Note that a separate defstruct must be executed. The fields need not be named here in the same order as those in the defstruct, nor must all the fields named in the defstruct be used here.

XDR: A sequence of objects of types <typedef1 XDR>...<typedefn XDR>.

(:sequence <typedef>) This is fashioned after Courier 's method for encoding/decoding linked lists. This type can often be used to get around clumsy recursive definitions involving :union's of enumeration type :boolean.

Lisp: A list of objects of type <typedefLisp>.

XDR: A sequence of objects, each preceded by an XDR :boolean encoding of true. The last object in the sequence is followed by the XDR :boolean encoding of false.

Note: (:sequence <typedef>) produces the same encoding (but not the same decoding) as
(defstruct astructure this-element the-rest)
along with the declaration

```
(:recursive (:union      :boolean
                         (T (:struct astructure      (this-
                         element <typedef>)
                                                     (the-rest
                         astructure)))
                         (NIL :void))),
```

(:list <typedef-1> ... <typedef-n>)

Lisp: A list , the ith element of which is of type <typedefi Lisp>.

XDR: A sequence of objects, the ith of which is of type <typedefi XDR>.

(:skip <unsigned>) (For decoding only)

Lisp: Nothing

XDR: Any n bytes of data, where <unsigned> = n.

Note: This is a klooge for not having to decode the fattr's that NFS returns with every single cotton-pickin' memory read.

EXAMPLE OF A REMOTE PROGRAM DEFINITION

The following call to define-remote-program defines the portmapper remote procedures described in Sun's Remote Procedure Call Specification. Note that there are two definitions of procedure 4 given. Since remote procedures may be invoked by name, it is reasonable for there to be more than one definition for how to decode and encode the arguments to a given routine. In this case, both a recursive and non-recursive definition is given for the values returned from procedure 4. Note also that mapstruct and mapsequence must be defstruct'ed before this call to define-remote-procedure.

```
(define-remote-program 'portmapper 100000 2 'udp
            :types    '(  (mapstruct (:union  :boolean
                                              (nil :void)
                                              (t (:struct mapstruct
                                                          (program :unsigned)
                                                          (vers :unsigned)
                                                          (prot :unsigned)
                                                          (port :unsigned)
                                                          (therest mapstruct)))))
                          (mapsequence (:sequence (:struct mapsequence
                                                           (program :unsigned)
```

```
                                                    (vers :unsigned)
                                                    (protocol :unsigned)
                                                    (port :unsigned)))))
        :procedures
                '(  (null 0 nil nil)
                    (lookup 3 (:unsigned :unsigned :unsigned :unsigned)
                                        (:unsigned))
                    (gooddump 4 nil (mapsequence))
                    (dump 4 nil (mapstruct))
                    (indirect 5 (:unsigned :unsigned :unsigned
                                        :string)
                                        (:unsigned :string))))
```

UNDEFINING REMOTE PROGRAMS

undefine-remote-program name number version [Function]

MAKING REMOTE PROCEDURE CALLS

remote-procedure-call destination program procid arglist [Function]
&key destsocket version credentials protocol
dynamic-prognum dynamic-version
msec-until-timeout msec-between-tries noerrorflg

Performs a remote procedure call to program on destination. Returns a list of the returned values.

destination — Designates the host to which the procedure call is made. If Destination is a number it is interpreted to be the il:iphostadress of the host; if a symbol or string, it is a name from which the net address of the host may be found.

program — Designates the remote program to be called. If Program is a number, it is interpreted to be the remote program number. If a symbol, in which case it is assumed to be the name of the remote procedure (as defined in define-remote-procedure. If :version is non-nil, then program is treated as a number rather than as a name. If version is nil and program is a number, then the latest version of that program is used.

procid — Designates the procedure number from program to be called. If Procid is a number it is interpreted to be the remote procedure number; if a symbol, it is the name given that procedure in define-remote-procedure.

arglist — A list of the arguments to be serialized into XDR representation and passed as the arguments of the remote procedure call.

:destsocket — Normally, the remote socket must be looked up in the local caches (See *rpc-socket-cache* and *rpc-well-known-sockets*) or else found by making a call to the Portmapper on the remote machine. If :destsocket is non-nil, its value is used as the remote socket.

:version — If non-nil designated the desired version of program as well as causing program to be interpreted as a number rather than a name. See program above.

:credentials — An object of type authentication to be passed as the credentials of the remote procedure call. (See create-unix-authentication).

:protocol — A symbol specifying the transport protocol. Currently only UDP is implemented. Defaults to UDP. The only reason for using this parameter is to specify (along with the program and version), which known remote program is to be used.

:dynamic-prognum — If you really can't live without it, dynamic-prognum is used as the remote program number in spite of treating the arglist and returned values exactly as in program. Don't ask why.

:dynamic-version — If you really can't live without it, dynamic-version is used as the remote program version in spite of treating the arglist and returned values exactly as specified in program (and possibly version). Don't ask why. Defaults to 1.

:msec-until-timeout — Total number of milliseconds of waiting for a reply packet before giving up on this remote procedure call. Defaults to value of *rpc-msec-until-timeout*.

:msec-between-tries — Number of milliseconds between outgoing UDP packets. Defaults to *rpc-msec-between-tries*.

:errorflg — If :noerrors, ignores remote procedure call errors. If :returnerrors, returns the error as an s-expression. Otherwise, signals a Lisp error. Default t.

LOW-LEVEL REMOTE PROCEDURE CALL FUNCTIONS

setup-rpc — destination program procid &optional destsocket version protocol dynamic-prognum dynamic-version [Function]

Returns four values destaddr, socket, program and procedure (Yes, this is real, live multiple value return requiring a multiple-value-bind or something similar.) for consumption by perform-rpc. The arguments to setup-rpc are identical in meaning to the identically named arguments to remote-procedure-call.

open-rpc-stream — protocol destaddr destsocket [Function]

Returns an rpcstream for use by perform-rpc. Destaddr and destsocket are as returned by setup-rpc and protocol is identical to the protocol argument to remote-procedure-call.

close-rpc-stream — rpcstream protocol [Function]

Closes rpcstream, an rpc-stream of protocol protocol created by open-rpc-stream .

perform-rpc — destaddr destsocket program procedure rpcstream arglist credentials protocol &key errorflg leave-stream-open msec-until-timeout msec-between-tries [Function]

Performs a remote procedure call returning a list of the values retruned by the remote procedure.

LISTING REMOTE PROGRAMS CURRENTLY DEFINED

list-remote-programs [Function]

Returns a list of 4-tuples (name number version protocol) for each remote program currently defined.

CREATION OF CREDENTIALS

create-unix-authentication
stamp machine-name uid gid gids [Function]

Returns a Unix-type authentication suitable for use as the credentials of a call to remote-procedure-call or perform-rpc.

stamp	An arbitrary unsigned integer.
machine-name	A string containing the name of the calling machine.
uid	User id number on the remote machine.
gid	Group id number on the machine.
gids	A list or array of group id numbers (on the remote machine) that contain the caller as a member.

GLOBAL VARIABLES

xdr-primitive-types	An a-list of keywords and the corresponding function that implements that XDR primitive type.
xdr-constructed-types	An a-list of keywords and the corresponding function that generates code to implement that XDR constructed type.
msec-until-timeout	Number of milliseconds before giving up on receiving a reply packet. Default 1000.
msec-between-tries	Number of milliseconds to wait before resending UDP packet. Default 100.
rpc-ok-to-cache	If non-nil, uses *rpc-socket-cache* as a cache of socket numbers found to date.
rpc-well-known-sockets	A list of well-known sockets. Format is (<host address> <remote program number> <remote program version> <protocol> <socket>)
rpc-socket-cache	A list of non-well-known sockets. Format is same as *rpc-well-known-sockets*.
debug	If non-nil prints out debugging information. If a number, the higher the number, the more information is printed. Default nil.

RPC FILES

RPC	Sets up the RPC2 Package and loads other RPC files. Loads Portmapper remote program definition and executes it.
RPCLOWLEVEL	Super low-level UDP/TCP functions added to Eric Schoen's TCPUDP code.
RPCOS	Low-level interface to Sun OS networking code .
RPCSTRUCT	Structure definitions used by the other files. These are in a separate file because they take so long to compile.
RPCCOMMON	Common lookup functions and stream i/o functions used by the other files.
RPCXDR	External Data Representation (XDR). Code Generation for XDR constructed types and XDR primitive functions.
RPCRPC	Remote program definition and remote procedure calls.
RPCPORTMAPPER	Definition of portmapper in UDP and TCP.

KNOWN DEFICIENCIES

Floating point XDR types are not implemented.

The view-packet utility is not documented and needs to be smarter about authentications.

Fall through cases of XDR types UNION and ENUMERATE should be added.

TCP is not supported under Medley 1.0-S, this should be in the next release.

COPYRIGHT INFORMATION

Written by Jeff Finger under support of National Institutes of Health Grant NIH 5P41 RR00785

to the SUMEX-AIM Computing Resource at Stanford University.

Modified to work under Medley 1.0-S by Atty Mullins.