Remote Procedure Call Programming Reference

Version 3.6

www.radisys.com

World Headquarters
5445 NE Dawson Creek Drive • Hillsboro, OR
97124 USA
Phone: 503-615-1100 • Fax: 503-615-1121
Toll-Free: 800-950-0044

International Headquarters
Gebouw Flevopoort • Televisieweg 1A
NL-1322 AC • Almere, The Netherlands
Phone: 31 36 5365595 • Fax: 31 36 5365620

RadiSys Microware Communications Software Division, Inc.
1500 N.W. 118th Street
Des Moines, Iowa 50325
515-223-8000

Revision A
November 2001

Copyright and publication information

This manual reflects version 3.6 of Remote Procedure Call.

Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, RadiSys Corporation will not be liable for any damages including indirect or consequential, from use of the OS-9 operating system, Microware-provided software, or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

Reproduction notice

The software described in this document is intended to be used on a single computer system. RadiSys Corporation expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of RadiSys Corporation and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.

November 2001

Table of Contents

Chapter 1: OS-9 RPC C Library

This chapter describes the structures used and the functions available for RPC programming.

Overview

The RPC library routines enable C programs to make procedure calls on machines across a network. The client first calls a procedure to send a data packet to the server. Upon receipt of the packet, the server calls a dispatch routine to perform the requested service and sends back a reply. Finally, the procedure call returns to the client.

RPC functions can be found in the `rpc.l` library.

RPCGEN is the compiler that generates client and server sides of an RPC program.

For More Information

See the RPCGEN Programming Guide chapter in the ***Using Network File System/Remote Procedure Call*** manual.

For More Information

See Remote Procedure Calls of the ***Using Network File System/Remote Procedure Call*** manual for RPC programming examples.

RPC Programming Structures

The main include file for all RPC files is `rpc.h`. It can be found in the directory `MWOS/SRC/DEFS/SPF/RPC`. This header file includes the following RPC include files:

```
RPC/xdr.h
RPC/auth.h
RPC/clnt.h
RPC/svc.h
RPC/pmap.clnt.h
RPC/rpc.msg.h
RPC/auth_unix.h
RPC/svc_auth.h
```

auth.h

The main structure in `RPC/auth.h` is described below.

```
/*
* Auth handle, interface to client side authenticators.
*/
typedef struct {
        struct  opaque_auth     ah_cred;
        struct  opaque_auth     ah_verf;
        struct auth_ops {
                void   (*ah_nextverf)();
                int    (*ah_marshal)();   /* nextverf & serialize */
                int    (*ah_validate)();  /* validate varifier */
                int    (*ah_refresh)();   /* refresh credentials */
                void   (*ah_destroy)();   /* destroy this structure */
        } *ah_ops;
        caddr_t ah_private;
} AUTH;
```

clnt.h

The main structure in RPC/clnt.h is described below.

```
/*
* Client rpc handle.
* Created by individual implementations, see e.g. rpc_udp.c.
* Client is responsible for initializing auth, see e.g. auth_none.c.
*/
typedef struct {
     AUTH    *cl_auth;                  /* authenticator */
     struct clnt_ops {
        enum clnt_stat  (*cl_call)();    /* call remote procedure */
        void            (*cl_abort)();   /* abort a call */
        void            (*cl_geterr)();  /* get specific error code */
        bool_t          (*cl_freeres)(); /* frees results */
        void            (*cl_destroy)(); /* destroy this structure */
        bool_t          (*cl_control)(); /* the ioctl() of rpc */
     } *cl_ops;
     caddr_t            cl_private;      /* private stuff */
} CLIENT;
```

svc.h

The main structure RPC/svc.h is described below.

```
* Server side transport handle
*/
typedef struct {
   int              xp_sock;
   u_short          xp_port;           /* associated port number */
   struct xp_ops {
       bool_t       (*xp_recv)();      /* receive incomming requests */
       enum xprt_stat (*xp_stat)();    /* get transport status */
       bool_t       (*xp_getargs)();   /* get arguments */
       bool_t       (*xp_reply)();     /* send reply */
       bool_t       (*xp_freeargs)();  /* free mem allocated for args */
       void         (*xp_destroy)();   /* destroy this struct */
   } *xp_ops;
   int              xp_addrlen;        /* length of remote address */
   struct sockaddr_in xp_raddr;        /* remote address */
   struct opaque_auth xp_verf;         /* raw response verifier */
   caddr_t          xp_p1;             /* private */
   caddr_t          xp_p2;             /* private */
} SVCXPRT;
```

xdr.h

The main structure in RPC/xdr.h is described below.

```
/*
* The XDR handle.
* Contains operation which is being applied to the stream,
* an operations vector for the paticular implementation
  (e.g. see xdr_mem.c),
* and two private fields for the use of the particular impelementation.
*/
typedef struct {
   enum xdr_op     x_op;        /* operation; fast additional param */
   struct xdr_ops {
      bool_t  (*x_getlong)();   /* get a long from underlying stream */
      bool_t  (*x_putlong)();   /* put a long to " */
      bool_t  (*x_getbytes)();  /* get some bytes from " */
      bool_t  (*x_putbytes)();  /* put some bytes to " */
      u_int   (*x_getpostn)();  /* returns bytes off from beginning */
      bool_t  (*x_setpostn)();  /* lets you reposition the stream */
      long *  (*x_inline)();    /* buf quick ptr to buffered data */
      void    (*x_destroy)();   /* free privates of this xdr_stream */
   } *x_ops;
   caddr_t         x_public;    /* users' data */
   caddr_t         x_private;   /* pointer to private data */
   caddr_t         x_base;      /* private used for position info */
   int             x_handy;     /* extra private word */
} XDR;
```

RPC C Library Functions

Table 1-1 on page 10 lists and briefly describes the RPC C library functions. Detailed descriptions follow.

Table 1-1 RPC C Library Functions

Function	Description
`auth_destroy()`	Destroy Authentication Information
`authnone_create()`	Create Authentication Handle
`authunix_create()`	Create Authentication Handle
`authunix_create_default()`	Create Authentication Handle
`callrpc()`	Call a Remote Procedure
`clnt_broadcast()`	Broadcast an RPC Call
`clnt_call()`	Call Remote Procedure
`clnt_control()`	Change Client Object
`clnt_create()`	Create Client Handle
`clnt_destroy()`	Destroy Client Handle
`clnt_freeres()`	Free Data Area Associated with Result
`clnt_geterr()`	Copy Client Error Structure
`clnt_pcreateerror()`	Print Message to Standard Error
`clnt_perrno()`	Print Message to Standard Error
`clnt_perror()`	Print Message to Standard Error
`clnt_spcreateerror()`	Encode Message to a Buffer

Table 1-1 RPC C Library Functions (continued)

Function	Description
clnt_sperrno()	Encode Message to a Buffer
clnt_sperror()	Encode Message to a Buffer
clntraw_create()	Create Loopback Client Handle
clnttcp_create()	Create Client Handle Using TCP
clntudp_create()	Create Client Handle Using UDP
get_myaddress()	Return Local Machine's Internet Address
pmap_getmaps()	Return List of Program to Port Mappings
pmap_getport()	Return Service Port Number
pmap_rmtcall()	Request Portmap to Make an RPC Call
pmap_set()	Establish Mapping for RPC Service
pmap_unset()	Destroy Mapping for RPC Service
registerrpc()	Register RPC Service with Portmap
svc_destroy()	Destroy Service Transport Handle
svc_freeargs()	Free Data Area for Parameters
svc_getargs()	Decode Parameters of a Service Request
svc_getcaller()	Get Network Address of Caller
svc_getreq()	Custom Asynchronous Event Processor

Table 1-1 RPC C Library Functions (continued)

Function	Description
svc_getreqset()	Custom Asynchronous Event Processor
svc_register()	Establish Service with Dispatch Routine
svc_run()	Process RPC Request
svc_sendreply()	Send RPC Results
svc_unregister()	Remove Mapping for RPC Service
svcerr_auth()	Report Authentication Error
svcerr_decode()	Report Decoding Error
svcerr_noproc()	Report Unknown Procedure Number
svcerr_noprog()	Report Unknown Program Number
svcerr_progvers()	Report Unknown Version Number
svcerr_systemerr()	Report System Error
svcerr_weakauth()	Report Weak Authentication
svcfd_create()	Create Service Transport on Open Descriptor
svcraw_create()	Create Loopback Service Transport
svctcp_create()	Create TCP Service Transport
svcudp_create()	Create UDP Service Transport

auth_destroy()

Destroy Authentication Information

Syntax

```
#include <RPC/rpc.h>
void auth_destroy(AUTH *auth)
```

Description

The `auth_destroy()` macro destroys the authentication information associated with `auth`. Destruction usually involves deallocation of private data structures.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`auth`	A pointer to the authentication handle. The use of `auth` is undefined after calling `auth_destroy()`.

authnone_create()

Create Authentication Handle

Syntax

```
#include <RPC/rpc.h>
AUTH * authnone_create(void)
```

Description

`authnone_create()` creates and returns an authentication handle that passes nonusable authentication information with each remote procedure call. This is the default authentication used by RPC.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

none

authunix_create()

Create Authentication Handle

Syntax

```
#include <RPC/rpc.h>
AUTH * authunix_create(
   char *machname,
   int uid,
   int gid,
   int len,
   int aup_gids)
```

Description

`authunix_create()` creates and returns an authentication handle that contains authentication information. The `len` and `aup_gids` parameters are ignored.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`machname`	Name of the machine the information was created on.
`uid`	User's OS-9 user ID.
`gid`	User's OS-9 group ID.
`len`	Number of elements in `aup_gids`.
`aup_gids`	Reference to a counted array of user's groups.

authunix_create_default()

Create Authentication Handle

Syntax

```
#include <RPC/rpc.h>
AUTH * authunix_create_default(void)
```

Description

`authunix_create_default()` calls `authunix_create()` using the appropriate parameters.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

none

callrpc()

Call a Remote Procedure

Syntax

```
#include <RPC/rpc.h>
int callrpc(
   char *host,
   int prognum,
   int versnum,
   int procnum,
   xdrproc_t inproc,
   char *in,
   xdrproc_t outproc,
   char *out)
```

Description

`callrpc()` calls the remote procedure associated with `prognum`, `versnum`, and `procnum` on the machine host. `callrpc()` returns a value of 0 if it succeeds or the value of the error cast to an integer if it fails. This routine is useful for translating failure statuses into messages.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`host`	Name of the machine the info was created on.
`prognum`	Program number.
`versnum`	Version number.
`procnum`	Procedure number.
`inproc`	Encodes the parameters.
`in`	Address of the parameter(s).

`outproc`	Decodes the results.
`out`	Address of where to place the result(s).

WARNING

Calling remote procedures with this routine uses TCP as a transport. You do not have control over time-outs or authentication when using this routine.

clnt_broadcast()

Broadcast an RPC Call

Syntax

```
#include <RPC/rpc.h>
enum clnt_stat clnt_broadcast(
   u_long prog,
   u_long vers,
   u_long proc,
   xdrproc_t xargs,
   caddr_t argsp,
   xdrproc_t xresults,
   caddr_t resultsp,
   resultproc_t eachresult)
```

Description

`clnt_broadcast()` calls a remote procedure by broadcasting the call message to all locally connected systems.

If `eachresult()` returns 0, `clnt_broadcast()` waits for more replies. Otherwise, it returns with the appropriate status.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`prog`	Program number.
`vers`	Version number.
`proc`	Procedure number.
`xargs`	Address of the parameter(s).
`argsp`	Address of where to place the result(s).
`xresults`	Encodes the parameters.

`resultsp`	Decodes the results.
`eachresult`	Each time `clnt_broadcast()` receives a response, it calls `eachresult()`. The form of `eachresult()` is: `eachresult(resp, addr)` where `resp` Address of where to place result. `addr` Address of sending machine.

WARNING

Broadcast packets are limited in size to the maximum transfer unit of the data link. For ethernet, this value is 1500 bytes.

clnt_call()

Call Remote Procedure

Syntax

```
#include <RPC/rpc.h>
enum clnt_stat clnt_call(
   CLIENT*rh
   u_long proc,
   xdrproc_t xargs,
   caddr_t argsp,
   xdrproc_t xres,
   caddr_t resp,
   struct timeval timeout)
```

Description

The `clnt_call()` macro calls the remote procedure associated with the client handle. The client handle is obtained from a client creation routine such as `clnt_create()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`rh`	Client handle.
`proc`	Procedure number.
`xargs`	Encodes the parameters.
`argsp`	Decodes the results.
`xres`	Address of the parameter(s).
`resp`	Address of where to place the result(s).
`timeout`	Time allowed for the results to return.

clnt_control()

Change Client Object

Syntax

```
#include <RPC/rpc.h>
bool_t clnt_control(
   CLIENT *cl,
   int request,
   char *info)
```

Description

The `clnt_control()` macro changes or retrieves information concerning a client object. These include the parameters timeout, retry count, and server address.

The requests are shown in **Table 1-2** on page 22.

Table 1-2 Requests

Request	Description
`CLSET_TIMEOUT`	Set timeout value.
`CLGET_TIMEOUT`	Get timeout value.
`CLSET_RETRY_TIMEOUT`	Set retry timeout value.
`CLGET_RETRY_TIMEOUT`	Get retry timeout value.
`CLGET_SERVER_ADDR`	Get server inet address.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`cl`	Client handle.
`request`	Indicates the type of operation.
`info`	Pointer to the information.

clnt_create()

Create Client Handle

Syntax

```
#include <RPC/rpc.h>
CLIENT * clnt_create (
   char *hostname,
   unsigned prog,
   unsigned vers,
   char *proto)
```

Description

`clnt_create()` is a generic client creation routine. Default time outs are set, but you can modify them using `clnt_call()` or `clnt_control()`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`hostname`	Identifies the remote host.
`prog`	Program number.
`vers`	Version number.
`proto`	Transport protocol to be used. The currently supported values for `proto` are `tcp` and `udp`.

WARNING

Using `clnt_create()` with UDP has its shortcomings. Because messages can only hold up to 8K of encoded data, you cannot use this transport for procedures that take large parameters or return huge results.

clnt_destroy()

Destroy Client Handle

Syntax

```
#include <RPC/rpc.h>
void clnt_destroy(CLIENT *rh)
```

Description

The `clnt_destroy()` macro destroys the client's handle. Destruction usually involves deallocation of private data structures, including its own structure. If the library opened the associated socket, it will close the socket. Otherwise, the socket remains open.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`rh`	Pointer to the client handle. Use of `rh` is undefined after calling `clnt_destroy()`.

clnt_freeres()

Free Data Area Associated with Result

Syntax

```
#include <RPC/rpc.h>
bool_t clnt_freeres(
   CLIENT *rh,
   xdrproc_t xres,
   char *resp);
```

Description

The `clnt_freeres()` macro frees any data allocated by the system when the results of a call were decoded. If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`rh`	Client handle.
`xres`	Routine describing results in simple primitive.
`resp`	Address of the results.

clnt_geterr()

Copy Client Error Structure

Syntax

```
#include <RPC/rpc.h>
void clnt_geterr(
   CLIENT *rh,
   struct rpc_err *errp);
```

Description

The `clnt_geterr()` macro copies the error structure out of the client handle to the structure at address `errp`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`rh`	Pointer to the client handle.
`errp`	Pointer to rpc_err structure.

clnt_pcreateerror()

Print Message to Standard Error

Syntax

```
#include <RPC/rpc.h>
void clnt_pcreateerror(char *msg)
```

Description

`clnt_pcreateerror()` prints a message to standard error indicating why a client handle could not be created. The message is prepended with string `msg` and a colon (:). `clnt_pcreateerror()` is used when a `clnt_create()` call fails.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`msg`	String to print.

clnt_perrno()

Print Message to Standard Error

Syntax

```
#include <RPC/rpc.h>
void clnt_perrno(enum clnt_stat num)
```

Description

`clnt_perrno()` prints a message to standard error corresponding to the condition indicated by `num`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`num`	Specifies client error status.

clnt_perror()

Print Message to Standard Error

Syntax

```
#include <RPC/rpc.h>
void clnt_perror(
   CLIENT *clnt,
   char *msg)
```

Description

`clnt_perror()` prints a message to standard error indicating why a call failed. `clnt` is the handle used to perform the call. The message is prepended with string `msg` and a colon (:).

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`clnt`	Client handle.
`msg`	Message to print.

clnt_spcreateerror()

Encode Message to a Buffer

Syntax

```
#include <RPC/rpc.h>
char * clnt_spcreateerror(char *msg)
```

Description

`clnt_spcreateerror()` operates like `clnt_pcreateerror()` except that it returns a pointer to a string instead of printing to standard error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`msg`	Message to encode.

WARNING

`clnt_spcreateerror()` returns a pointer to static data. This static data area is overwritten on each call.

clnt_sperrno()

Encode Message to a Buffer

Syntax

```
#include <RPC/rpc.h>
char * clnt_sperrno(enum clnt_stat num)
```

Description

`clnt_sperrno()` accepts the same parameters as `clnt_perrno()`. However, instead of sending a message to standard error indicating why a call failed, it returns a pointer to a string containing the message. The string ends with a newline character.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`num`	Client error status.

WARNING

`clnt_sperrno()` returns a pointer to static data. This static data area is overwritten on each call.

clnt_sperror()

Encode Message to a Buffer

Syntax

```
#include <RPC/rpc.h>
char * clnt_sperror(
   CLIENT *clnt,
   char *msg)
```

Description

`clnt_sperror()` is similar to `clnt_perror()`. However, `clnt_sperror()` returns a pointer to a string instead of printing to standard error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`clnt`	Client handle.
`msg`	Message to encode.

WARNING

`clnt_sperror()` returns a pointer to static data. This static data area is overwritten on each call.

clntraw_create()

Create Loopback Client Handle

Syntax

```
#include <RPC/rpc.h>
CLIENT * clntraw_create(
   u_long prog,
   u_long vers)
```

Description

`clntraw_create()` creates a local client for the remote program `prog` and version `vers`. The transport used to pass messages to the service is actually a buffer within the process's address space. Therefore, the corresponding server should be located in the same address space. This allows simulation and acquisition of overheads, such as round trip times.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`prog`	Program number.
`vers`	Version number.

clnttcp_create()

Create Client Handle Using TCP

Syntax

```
#include <RPC/rpc.h>
CLIENT * clnttcp_create(
   struct sockaddr_in *addr,
   u_long prog,
   u_long vers,
   int *sockp,
   u_int sendsz,
   u_int recvsz)
```

Description

`clnttcp_create()` creates a client for the remote program `prog` and version `vers`. The client uses TCP as a transport.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`addr`	Internet address of the remote program. If `addr->sin_port` is zero, it is set to the actual port on which the remote program is listening. The remote service is consulted for this information.
`prog`	Program number.
`vers`	Version number.
`sockp`	A pointer to a socket. If `sockp` is null, this routine opens a new socket.
`sendsz`	Size of the send buffer. If 0, the default is chosen.
`recvsz`	Size of the receive buffer. If 0, the default is chosen.

clntudp_create()

Create Client Handle Using UDP

Syntax

```
#include <RPC/rpc.h>
CLIENT * clntudp_create(
   struct sockaddr_in *addr,
   u_long prog,
   u_long vers,
   struct timeval wait,
   int *sockp)
```

Description

`clntudp_create()` creates a client handle for the remote program `prog` and version `vers`. The client uses UDP as a transport. `clntudp_create()` sends the call message periodically until a response is received or until the call times out.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`addr`	Internet address of the remote program. If `addr->sin_port` is zero, it is set to the actual port on which the remote program is listening. The remote service is consulted for this information.
`prog`	Program number.
`vers`	Version number.
`wait`	A time-out value used for this call.
`sockp`	A pointer to a socket. If `sockp` is null, this routine opens a new socket.

get_myaddress()

Return Local Machine's Internet Address

Syntax

```
#include <RPC/rpc.h>
void get_myaddress(struct sockaddr_in *addr)
```

Description

`get_myaddress()` returns the machine's address in `addr`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`addr`	Location to store the machine's socket address.

pmap_getmaps()

Return List of Program to Port Mappings

Syntax

```
#include <RPC/rpc.h>
struct pmaplist *pmap_getmaps(
  struct sockaddr_in *addr)
```

Description

`pmap_getmaps()` returns a list of the current program-to-port mappings on the host located at address `addr`. This routine can return null. The `rpcinfo` command uses this routine.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`addr`	Specifies the socket address of the host.

pmap_getport()

Return Service Port Number

Syntax

```
#include <RPC/rpc.h>
u_short pmap_getport(
   struct sockaddr_in *addr,
   u_long prog,
   u_long vers,
   u_int protocol)
```

Description

`pmap_getport()` returns the port number on which waits a service that supports program `prog` and version `vers` and speaks the transport protocol `protocol`. A return value of 0 indicates that the mapping does not exist or that the system failed to contact the remote portmap service.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`addr`	Socket address.
`prog`	Program number requested.
`vers`	Version number requested.
`protocol`	Protocol to be used. Supported values are `IPPROTO_UDP` and `IPPROTO_TCP`.

pmap_rmtcall()

Request Portmap to Make an RPC Call

Syntax

```
#include <RPC/rpc.h>
enum clnt_stat pmap_rmtcall(
   struct sockaddr_in *addr,
   u_long prog,
   u_long vers,
   u_long proc,
   xdrproc_t xdrargs,
   caddr_t argsp,
   xdrproc_t xdrves,
   caddr_t resp,
   struct timeval tout,
   u_long *port_ptr)
```

Description

`pmap_rmtcall()` instructs the port mapper on the host at address `addr` to make an RPC call for the user to a procedure on that host.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`addr`	Socket address.
`prog`	Program number.
`vers`	Version number.
`proc`	Procedure number.
`xdrargs`	Address of the parameter(s).
`argsp`	Address of where to place the result(s).

`xdrres`	Encodes the parameters.
`resp`	Decodes the results.
`tout`	Time allowed for the results to return.
`port_ptr`	Port number. `port_ptr` is modified to the program's port number if the procedure succeeds.

pmap_set()

Establish Mapping for RPC Service

Syntax

```
#include <RPC/rpc.h>
bool_t pmap_set(
   u_long prog,
   u_long vers,
   int protocol,
   u_short port)
```

Description

`pmap_set()` establishes a mapping between `port`, `prog`, `vers`, and `protocol` on the machine's `portmap` service. If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`prog`	Program number.
`vers`	Version number.
`protocol`	Protocol to be used. Supported values are `IPPROTO_UDP` and `IPPROTO_TCP`.
`port`	Port number to associate with the program, version, and protocol triple.

pmap_unset()

Destroy Mapping for RPC Service

Syntax

```
#include <RPC/rpc.h>
bool_t pmap_unset(
   u_long prog,
   u_long vers)
```

Description

`pmap_unset()` destroys all mappings involving program `prog` and version `vers` on the machine's `portmap` service. If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`prog`	Program number.
`vers`	Version number.

registerrpc()

Register RPC Service with Portmap

Syntax

```
#include <RPC/rpc.h>
int registerrpc(
   u_long prog,
   u_long vers,
   u_long proc,
   char *(*procname)(,)
   xdrproc_t inproc,
   xdrproc_t outproc)
```

Description

`registerrpc()` registers a procedure with the service package. If a request arrives for program `prog`, version `vers`, and procedure `proc`, `procname` is called with a pointer to its parameter(s). `procname` should return a pointer to its static result(s). This routine returns a value of 0 if the registration succeeded. Otherwise, it returns a value of -1.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`prog`	Program number.
`vers`	Version number.
`proc`	Procedure number.
`procname`	Name of a procedure.
`inproc`	Decodes the parameters.
`outproc`	Encodes the results.

svc_destroy()

Destroy Service Transport Handle

Syntax

```
#include <RPC/rpc.h>
void svc_destroy(SVCXPRT *xprt)
```

Description

`svc_destroy()` destroys the service transport handle `xprt`. Destruction usually involves deallocation of private data structures. This includes its own data structure.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xprt`	Service transport handle. Use of `xprt` is undefined after calling this routine.

svc_freeargs()

Free Data Area for Parameters

Syntax

```
#include <RPC/rpc.h>
bool_t svc_freeargs(
   SVCXPRT *xprt,
   xdrproc_t xargs,
   char *argsp)
```

Description

`svc_freeargs()` frees any data allocated by the system when the parameters to a service procedure using `xargs` were decoded. This routine returns a value of 1 if the results were successfully freed. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xprt`	Service transport handle.
`xargs`	Decodes the parameters.
`argsp`	Address where parameters are placed.

svc_getargs()

Decode Parameters of a Service Request

Syntax

```
#include <RPC/rpc.h>
bool_t svc_getargs(
   SVCXPRT *xprt,
   xdrproc_t xargs,
   char *argsp)
```

Description

`svc_getargs()` decodes the parameters of a request associated with `xprt`. This routine returns a value of 1 if decoding succeeds. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xprt`	Service transport handle.
`xargs`	Decodes the parameters.
`argsp`	Address where parameters are placed.

svc_getcaller()

Get Network Address of Caller

Syntax

```
#include <RPC/rpc.h>
struct sockaddr_in *svc_getcaller(SVCXPRT *xprt)
```

Description

`svc_getcaller()` gets the network address of the caller of a procedure associated with `xprt`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xprt`	Service transport handle.

svc_getreq()

Custom Asynchronous Event Processor

Syntax

```
#include <RPC/rpc.h>
void svc_getreq(int *rdfds)
```

Description

`svc_getreq()` is similar to `svc_getreqset()` but limited to 32 descriptors. `svc_run()` makes this interface obsolete.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`rdfds`	Resultant `read` file descriptor bit mask.

svc_getreqset()

Custom Asynchronous Event Processor

Syntax

```
#include <RPC/rpc.h>
void svc_getreqset(int *readfds)
```

Description

`svc_getreqset()` is only of interest if a service implementor does not call `svc_run()`, but instead implements custom asynchronous event processing. `svc_getreqset()` is called when the system has determined that a request has arrived on some socket(s). The routine returns when all sockets associated with the value of `readfds` have been serviced.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`readfds`	Resultant `read` file descriptor bit mask.

svc_register()

Establish Service with Dispatch Routine

Syntax

```
#include <RPC/rpc.h>
bool_t svc_register(
   SVCXPRT xprt,
   u_long prog,
   u_long vers,
   void (*dispatch)(,)
   int protocol)
```

Description

`svc_register()` associates `prog` and `vers` with the service dispatch procedure. If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xprt`	Service transport handle. If `xprt` is 0, the service is not registered with dispatch. If `xprt` is non-zero, a mapping of `prog`, `vers`, and `protocol` is established with the local service.
`prog`	Program number.
`vers`	Version number.

`dispatch` Service dispatch procedure. The dispatch procedure has the following form:

```
dispatch(request, xprt)
struct svc_req *request;
```

where:

`xprt` is the service transport handle.

`protocol` Protocol to be used. Supported values are `IPPROTO_UDP` and `IPPROTO_TCP`.

svc_run()

Process RPC Request

Syntax

```
#include <RPC/rpc.h>
void svc_run(void)
```

Description

`svc_run()` never returns. It waits for requests to arrive. When a request arrives, `svc_run()` calls the appropriate service procedure. This procedure is usually waiting for I/O.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

none

svc_sendreply()

Send RPC Results

Syntax

```
#include <RPC/rpc.h>
bool_t svc_sendreply(
   SVCXPRT *xprt,
   xdrproc_t xdr_results,
   caddr_t xdr_location)
```

Description

`svc_sendreply()` is called by a service dispatch routine to send the results of a remote procedure call. If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xprt`	Service transport handle.
`xdr_results`	Routine used to encode the results.
`xdr_location`	Address of the results.

svc_unregister()

Remove Mapping for RPC Service

Syntax

```
#include <RPC/rpc.h>
void svc_unregister(
   u_long prog,
   u_long vers)
```

Description

`svc_unregister()` removes all mapping of the service to dispatch routines and `portmap`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`prog`	Specifies a remote program.
`vers`	Specifies the version of a remote program.

svcerr_auth()

Report Authentication Error

Syntax

```
#include <RPC/rpc.h>
void svcerr_auth(
   SVCXPRT *xprt,
   enum auth_stat *why)
```

Description

`svcerr_auth()` reports an authentication error. It is called by a service dispatch routine that refuses to perform a remote procedure call due to an authentication error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xprt`	Service transport handle.
`why`	Authentication error status.

svcerr_decode()

Report Decoding Error

Syntax

```
#include <RPC/rpc.h>
void svcerr_decode(SVCXPRT *xprt)
```

Description

`svcerr_decode()` reports a decoding error. It is called by a service dispatch routine that cannot successfully decode its parameters.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xprt`	Service transport handle.

svcerr_noproc()

Report Unknown Procedure Number

Syntax

```
#include <RPC/rpc.h>
void svcerr_noproc(SVCXPRT *xprt)
```

Description

`svcerr_noproc()` reports an unknown procedure number. It is called by a service dispatch routine that does not implement the procedure number that the caller requests.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xprt`	Service transport handle.

svcerr_noprog()

Report Unknown Program Number

Syntax

```
#include <RPC/rpc.h>
void svcerr_noprog(SVCXPRT *xprt)
```

Description

`svcerr_noprog()` reports an unknown program number. It is called when the desired program is not registered with the package. Service implementors usually do not need to use this routine.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xprt`	Service transport handle.

svcerr_progvers()

Report Unknown Version Number

Syntax

```
#include <RPC/rpc.h>
void svcerr_progvers(SVCXPRT *xprt)
```

Description

`svcerr_progvers()` reports an unknown version number. It is called when the desired version of a program is not registered with the package. Service implementors usually do not need to use this routine.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xprt`	Service transport handle.

svcerr_systemerr()

Report System Error

Syntax

```
#include <RPC/rpc.h>
void svcerr_systemerr(SVCXPRT *xprt)
```

Description

`svcerr_systemerr()` reports a system error. It is called by a service dispatch routine when the routine detects a system error not covered by any particular protocol. For example, if a service can no longer allocate storage, it may call this routine.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xprt`	Service transport handle.

svcerr_weakauth()

Report Weak Authentication

Syntax

```
#include <RPC/rpc.h>
void svcerr_weakauth(SVCXPRT *xprt)
```

Description

`svcerr_weakauth()` reports weak authentication. It is called by a service dispatch routine that refuses to perform a remote procedure call due to insufficient (but correct) authentication parameters.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xprt`	Service transport handle.

svcfd_create()

Create Service Transport on Open Descriptor

Syntax

```
#include <RPC/rpc.h>
SVCXPRT *svcfd_create(
   int fd,
   u_int sendsize,
   u_int recvsize)
```

Description

`svcfd_create()` creates a service on top of an open path. Typically, this path is a connected socket for a stream protocol such as TCP.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`fd`	Path number.
`sendsize`	Size of the send buffer. If 0, defaults are chosen.
`recvsize`	Size of the receive buffer. If 0, defaults are chosen.

svcraw_create()

Create Loopback Service Transport

Syntax

```
#include <RPC/rpc.h>
SVCXPRT * svcraw_create(void)
```

Description

`svcraw_create()` creates a local service transport. This routine returns a pointer to the transport. The transport is a buffer within the process's address space. The corresponding client should live in the same address space. This routine allows for the simulation and acquisition of overheads, such as round trip times.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

none

svctcp_create()

Create TCP Service Transport

Syntax

```
#include <RPC/rpc.h>
SVCXPRT * svctcp_create(
   int sock,
   u_int sendsize,
   u_int recvsize)
```

Description

`svctcp_create()` creates a service transport and returns a pointer to it. The transport is associated with the socket `sock`. If the socket is not bound to a local port, this routine binds it to an arbitrary port. This routine chooses suitable defaults if a value of 0 is specified for `sendsize` and `recvsize`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`sock`	Transport's socket number. `sock` may be null. In this case, a new socket is created.
`sendsize`	Size of the send buffer.
`recvsize`	Size of the receive buffer.

svcudp_create()

Create UDP Service Transport

Syntax

```
#include <RPC/rpc.h>
SVCXPRT * svcudp_create(int sock)
```

Description

`svcudp_create()` creates a service transport and returns a pointer to it. The transport is associated with the socket `sock`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`sock`	The transport's socket number. `sock` may be null. In this case, a new socket is created. If the socket is not bound to a local port, this routine binds it to an arbitrary port.

Chapter 2: OS-9 XDR C Library

This chapter lists the RPC programming structures used for XDR and functions available for XDR C library programming.

Overview

The XDR C library routines enable C programmers to describe arbitrary data structures in a machine-independent fashion. Data for remote procedure calls are transmitted using these routines.

XDR functions can be found in the `rpc.l` library.

RPCGEN is the compiler that generates client and server sides of an RPC program.

For More Information

See the RPCGEN Programming Guide chapter in the ***Using Network File System/Remote Procedure Call*** manual.

For More Information

See ***Using Network File System/Remote Procedure Call*** manual for XDR programming examples.

XDR Programming Structures

The main include file for XDR is RPC/rpc.h, which includes RPC/xdr.h. It can be found in MWOS/SRC/DEFS/SPF/RPC. RPC/xdr.h defines the preliminary XDR structure:

```
/*
* The XDR handle.
* Contains operation which is being applied to the stream, an
* operations vector for the paticular implementation (e.g. see
* xdr_mem.c), and two private fields for the use of the particular
* impelementation.
*/
typedef struct {
   enum xdr_op     x_op;       /* operation; fast additional param */
   struct xdr_ops {
       bool_t  (*x_getlong)(); /* get a long from underlying stream */
       bool_t  (*x_putlong)(); /* put a long to " */
       bool_t  (*x_getbytes)();/* get some bytes from " */
       bool_t  (*x_putbytes)();/* put some bytes to " */
       u_int   (*x_getpostn)();/* returns bytes off from beginning */
       bool_t  (*x_setpostn)();/* lets you reposition the stream */
       long *  (*x_inline)();  /* buf quick ptr to buffered data */
       void    (*x_destroy)(); /* free privates of this xdr_stream */
   } *x_ops;
   caddr_t       x_public;     /* users' data */
   caddr_t       x_private;    /* pointer to private data */
   caddr_t       x_base;       /* private used for position info */
   int           x_handy;      /* extra private word */
} XDR;
```

XDR C Library Functions

Table 2-1 on page 72 lists and briefly describes the XDR C library functions. Detailed descriptions follow.

Table 2-1 XDR C Library Functions

Function	Description
`xdr_accepted_reply()`	Encode RPC Reply Messages
`xdr_array()`	Translate Arrays to/from XDR
`xdr_authunix_parms()`	Generate UNIX Credentials
`xdr_bool()`	Translate Booleans to/from XDR
`xdr_bytes()`	Translate Counted Bytes to/from XDR
`xdr_callhdr()`	Encode RPC Call Header
`xdr_callmsg()`	Encode RPC Call Message
`xdr_char()`	Translate Characters to/from XDR
`xdr_destroy()`	Destroy XDR Stream
`xdr_double()`	Translate Double Precision Numbers to/from XDR
`xdr_enum()`	Translate Enumerated Types to/from XDR
`xdr_float()`	Translate Floating Point Numbers to/from XDR
`xdr_free()`	Free XDR Structure
`xdr_getpos()`	Return Position in an XDR Stream
`xdr_inline()`	Invoke Inline XDR Function

Table 2-1 XDR C Library Functions (continued)

Function	Description
xdr_int()	Translate Integers to/from XDR
xdr_long()	Translate Long Integers to/from XDR
xdr_opaque()	Translate Opaque Data to/from XDR
xdr_opaque_auth()	Describe RPC Authentication Information
xdr_pmap()	Describe Procedure Parameters and Port Maps
xdr_pmaplist()	Describe Procedure Parameters and Port Maps
xdr_pointer()	Translate Pointer to/from XDR
xdr_reference()	Translate Pointers to/from XDR
xdr_rejected_reply()	Encode Rejected RPC Message
xdr_replymsg()	Encode RPC Reply Message
xdr_setpos()	Set Position within XDR Stream
xdr_short()	Translate Short Integers to/from XDR
xdr_string()	Translate Strings to/from XDR
xdr_u_char()	Translate Unsigned Characters to/from XDR
xdr_u_int()	Translate Unsigned Integers to/from XDR
xdr_u_long()	Translate Unsigned Long Integers to/from XDR
xdr_u_short()	Translate Unsigned Short Integers to/from XDR
xdr_union()	Translate Discriminated Union to/from XDR

Table 2-1 XDR C Library Functions (continued)

Function	Description
xdr_vector()	Translate Fixed-Length Arrays to/from XDR
xdr_void()	Translate Nothing to/from XDR
xdr_wrapstring()	Package RPC Message
xdrmem_create()	Initialize XDR Stream Object
xdrrec_create()	Initialize XDR Stream
xdrrec_endofrecord()	Mark End of Record in XDR Stream
xdrrec_eof()	Check EOF on XDR Stream
xdrrec_skiprecord()	Skip Current Record in XDR Stream
xprt_register()	Register Transport Handle
xprt_unregister()	Unregister Service Transport Handle
xdrtdio_create()	Initialize XDR Stream Object

xdr_accepted_reply()

Encode RPC Reply Messages

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_accepted_reply(
   XDR *xdrs,
   struct accepted_reply *ar)
```

Description

`xdr_accepted_reply()` encodes the status of the RPC call. It is used to generate RPC-style messages without using the RPC package.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Service transport handle.
`ar`	Message accepted reply.

xdr_array()

Translate Arrays to/from XDR

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_array(
   XDR *xdrs,
   caddr_t *addrp,
   u_int *sizep,
   u_int maxsize,
   u_int elsize,
   xdrproc_t elproc)
```

Description

The `xdr_array()` filter primitive translates between variable-length arrays and their corresponding external representations. `elproc` translates between the array elements' C form and their external representation.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`addrp`	Address of the pointer to the array.
`sizep`	Address of the element count of the array.
`maxsize`	Specifies the maximum size of `sizep`.
`elsize`	Size of each of the array's elements.
`elproc`	Filter.

xdr_authunix_parms()

Generate UNIX Credentials

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_authunix_parms(
   XDR *xdrs,
   struct authunix_parms *p)
```

Description

`xdr_authunix_parms()` externally describes credentials. This routine generates these credentials without using the authentication package.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Service transport handle.
`p`	Authentication parameters.

xdr_bool()

Translate Booleans to/from XDR

Syntax

```
#include <RPC/rpc.h>
boot_l xdr_bool(
   XDR *xdrs,
   bool_t *bp)
```

Description

The `xdr_bool()` filter primitive translates between booleans (C integers) and their external representations. When encoding data, this filter produces values of either 1 or 0.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
bp	Target boolean variable or constant.

xdr_bytes()

Translate Counted Bytes to/from XDR

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_bytes(
   XDR *xdrs,
   char *cpp,
   u_int *sizep,
   u_int maxsize)
```

Description

The `xdr_bytes()` filter primitive translates between counted byte strings and their external representations.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`cpp`	Address of the string pointer.
`sizep`	Length of the string.
`maxsize`	Specifies maximum size of `sizep`.

xdr_callhdr()

Encode RPC Call Header

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_callhdr(
   XDR *xdrs,
   struct rpc_msg *cmsg)
```

Description

`xdr_callhdr()` encodes the static part of the call message header. It is used to generate RPC-style messages without using the RPC package.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Service transport handle.
`cmsg`	Header portion of RPC message.

xdr_callmsg()

Encode RPC Call Message

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_callmsg(
   XDR *xdrs,
   struct rpc_msg *cmsg)
```

Description

`xdr_callmsg()` encodes an RPC call message. It is used to generate RPC-style messages without using the RPC package.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Service transport handle.
`cmsg`	RPC call message.

xdr_char()

Translate Characters to/from XDR

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_char(
   XDR *xdrs,
   char *cp)
```

Description

The `xdr_char()` filter primitive translates between C characters and their external representations.

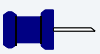
If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`cp`	Target character.



Note

Encoded characters are not packed. They occupy four bytes each.

xdr_destroy()

Destroy XDR Stream

Syntax

```
#include <RPC/rpc.h>
void xdr_destroy(XDR *xdrs)
```

Description

The `xdr_destroy()` macro invokes the destroy routine associated with `xdrs`. Destruction usually involves freeing private data structures associated with the stream. Using `xdrs` after invoking `xdr_destroy()` is undefined.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.

xdr_double()

Translate Double Precision Numbers to/from XDR

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_double(
   XDR *xdrs,
   double *dp)
```

Description

The `xdr_double()` filter primitive translates between C double precision numbers and their external representations.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`dp`	Target double precision variable.

xdr_enum()

Translate Enumerated Types to/from XDR

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_enum(
   XDR *xdrs,
   enum_t *ep)
```

Description

The `xdr_enum()` filter primitive translates between C enumerations (actually integers) and their external representations.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`ep`	Target enumeration variable.

xdr_float()

Translate Floating Point Numbers to/from XDR

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_float(
   XDR *xdrs,
   float *fp)
```

Description

The `xdr_float()` filter primitive translates between C floating point numbers and their external representations.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`fp`	Target float variable.

xdr_free()

Free XDR Structure

Syntax

```
#include <RPC/rpc.h>
void xdr_free(
   xdrproc_t proc,
   char *objp)
```

Description

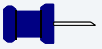
`xdr_free()` is a generic freeing routine.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`proc`	Routine for the object being freed.
`objp`	Pointer to the object itself.



Note

The pointer passed to this routine is not freed, but what it points to is recursively freed.

xdr_getpos()

Return Position in an XDR Stream

Syntax

```
#include <RPC/rpc.h>
u_int xdr_getpos(XDR *xdrs)
```

Description

The `xdr_getpos()` macro invokes the get-position routine associated with `xdrs`. The routine returns an unsigned integer indicating the position of the byte stream. A desirable feature of streams is that simple arithmetic works with this number, although the stream instances need not guarantee this.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.

xdr_inline()

Invoke Inline XDR Function

Syntax

```
#include <RPC/rpc.h>
long * xdr_inline(
   XDR *xdrs,
   int len)
```

Description

The `xdr_inline` macro invokes the inline routine associated with `xdrs`. The routine returns a pointer to a contiguous piece of the stream's buffer.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`len`	Byte length of the desired buffer.

WARNING

`xdr_inline()` may return 0 if it cannot allocate a contiguous piece of a buffer. Therefore, the behavior may vary among stream instances. It exists for efficiency.

xdr_int()

Translate Integers to/from XDR

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_int(
   XDR *xdrs,
   int *ip)
```

Description

The `xdr_int()` filter primitive translates between C integers and their external representations.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`ip`	Target integer variable.

xdr_long()

Translate Long Integers to/from XDR

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_long(
   XDR *xdrs,
   long *lp)
```

Description

The `xdr_long()` filter primitive translates between C long integers and their external representations.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`lp`	Target long variable.

xdr_opaque()

Translate Opaque Data to/from XDR

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_opaque(
   XDR *xdrs,
   caddr_t cp,
   u_int cnt)
```

Description

The `xdr_opaque()` filter primitive translates between fixed size opaque data and its external representation.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`cp`	Address of opaque object.
`cnt`	Size of the opaque object in bytes.

xdr_opaque_auth()

Describe RPC Authentication Information

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_opaque_auth(
   XDR *xdrs,
   struct opaque_auth *ap)
```

Description

`xdr_opaque_auth()` describes RPC authentication information messages. It is used to generate RPC-style messages without using the RPC package.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Service transport handle.
`ap`	Opaque authentication structure.

xdr_pmap()

Describe Procedure Parameters and Port Maps

Syntax

```
#include <RPC/pmap_prot.h)
bool_t xdr_pmap(
   XDRS *xdrs,
   struct pmap *regs)
```

Description

`xdr_pmap()` externally describes parameters to various procedures. This routine generates these parameters without using the pmap interface.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Service transport handle.
`regs`	Portmap parameters.

xdr_pmaplist()

Describe Procedure Parameters and Port Maps

Syntax

```
#include <RPC/pmap_prot.h)
bool_t xdr_pmaplist(
   XDR *xdrs,
   struct pmaplist **rp)
```

Description

`xdr_pmaplist()` externally describes a list of port mappings. This routine generates these parameters without using the pmap interface.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Service transport handle.
`rp`	Portmap list.

xdr_pointer()

Translate Pointer to/from XDR

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_pointer(
   XDR *xdrs,
   char *objpp,
   u_int obj_size,
   xdrproc_t xdr_obj)
```

Description

`xdr_pointer()` serializes pointers. This routine can handle recursive data structures, such as binary trees or linked lists.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`objpp`	Pointer to object.
`obj_size`	Size of object.
`xdr_obj`	XDR procedure to process target object.

xdr_reference()

Translate Pointers to/from XDR

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_reference(
   XDR *xdrs,
   caddr_t *pp,
   u_int size,
   xdrproc_t proc)
```

Description

The `xdr_reference()` primitive provides pointer chasing within structures. `proc` filters the structure between its C form and its external representation.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`pp`	Address of the pointer.
`size`	Size of structure pointed to by `pp`.
`proc`	Filter.

xdr_rejected_reply()

Encode Rejected RPC Message

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_rejected_reply(
   XDR *xdrs,
   struct rejected_reply *rr)
```

Description

`xdr_rejected_reply()` encodes the rejecting RPC message. It is used to generate RPC-style messages without using the RPC package.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Service transport handle.
`rr`	Message rejected reply.

xdr_replymsg()

Encode RPC Reply Message

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_replymsg(
   XDR *xdrs,
   struct rpc_msg *rmsg)
```

Description

`xdr_replymsg` encodes an RPC reply message. It is used to generate RPC-style messages without using the RPC package.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Service transport handle.
`rmsg`	RPC message.

xdr_setpos()

Set Position within XDR Stream

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_setpos(
   XDR *xdrs,
   u_int pos)
```

Description

The `xdr_setpos()` macro invokes the set position routine associated with `xdrs`. This routine returns a value of 1 if the stream could be repositioned. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`pos`	Position value.

WARNING

It is difficult to reposition some stream types. This routine may fail with one type of stream and succeed with another.

xdr_short()

Translate Short Integers to/from XDR

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_short(
   XDR *xdrs,
   short *sp)
```

Description

The `xdr_short()` filter primitive translates between C short integers and their external representations.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`sp`	Target short variable.

xdr_string()

Translate Strings to/from XDR

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_string(
   XDR *xdrs,
   char **cpp,
   u_int maxsize)
```

Description

The `xdr_string()` filter primitive translates between C strings and their corresponding external representations.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`cpp`	Address of the pointer to the string.
`maxsize`	Maximum size of the string.

xdr_u_char()

Translate Unsigned Characters to/from XDR

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_u_char(
   XDR *xdrs,
   u_char *cp)
```

Description

The `xdr_u_char()` filter primitive translates between C unsigned characters and their external representations.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`cp`	Target unsigned `char` variable.

xdr_u_int()

Translate Unsigned Integers to/from XDR

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_u_int(
   XDR *xdrs,
   u_int *up)
```

Description

The `xdr_u_int()` filter primitive translates between C integers and their external representations.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`up`	Target unsigned integer variable.

xdr_u_long()

Translate Unsigned Long Integers to/from XDR

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_u_long(
   XDR *xdrs,
   u_long *ulp)
```

Description

The `xdr_u_long()` filter primitive translates between C unsigned long integers and their external representations.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`ulp`	Target unsigned long variable.

xdr_u_short()

Translate Unsigned Short Integers to/from XDR

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_u_short(
   XDR *xdrs,
   u_short *usp)
```

Description

The `xdr_u_short()` filter primitive translates between C unsigned short integers and their external representations.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`usp`	Target unsigned short variable.

xdr_union()

Translate Discriminated Union to/from XDR

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_union(
   XDR *xdrs,
   enum_t *dscmp,
   char *unp,
   struct xdr_discrim *choices,
   xdrproc_t dfault)
```

Description

The `xdr_union()` filter primitive translates between a discriminated C union and its corresponding external representation. It translates the discriminant of the union located at `dscmp`. This discriminant is always an integer. Then, the union located at `unp` is translated. Each structure contains an ordered pair of values. If the union's discriminant is equal to the associated value, the routine translates the union. The end of the structure array is denoted by a routine of value 0. If the discriminant is not found in the array, the `dfault` procedure is called.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`dscmp`	Location of a union.
`unp`	Location of a union.
`choices`	Pointer to an array of structures.
`dfault`	Function to call if discriminant is not found (may be `NULL`).

xdr_vector()

Translate Fixed-Length Arrays to/from XDR

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_vector(
   XDR *xdrs,
   char *basep,
   u_int nelem,
   u_int elemsize,
   xdrproc_t xdr_elem)
```

Description

The `xdr_vector()` filter primitive translates between fixed-length arrays and their corresponding external representations. `xdr_elem` translates between the array elements' C form and their external representation.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`basep`	Address of the pointer to the array.
`nelem`	Element count of array.
`elemsize`	Size of each of the array's elements.
`xdr_elem`	A filter.

xdr_void()

Translate Nothing to/from XDR

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_void(void)
```

Description

`xdr_void()` always returns 1. It may be passed to routines that require a function parameter, where nothing is to be done.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

none

xdr_wrapstring()

Package RPC Message

Syntax

```
#include <RPC/rpc.h>
bool_t xdr_wrapstring(
   XDR *xdrs,
   char **cpp)
```

Description

The `xdr_wrapstring()` primitive calls `xdr_string()`. It is useful because the package passes a maximum of two routines as parameters, and one of the most frequently used primitives, `xdr_string()` requires three.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`cpp`	Address of pointer to the string to convert.

xdrmem_create()

Initialize XDR Stream Object

Syntax

```
#include <RPC/rpc.h>
void xdrmem_create(
   XDR *xdrs,
   caddr_t addr,
   u_int size,
   enum xdr_op op)
```

Description

`xdrmem_create()` initializes the stream object pointed to by `xdrs`. The stream's data is written to memory or read from memory at location `addr`. `size` specifies the stream size.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`addr`	Address of a memory location.
`size`	Maximum length of memory location.
`op`	Determines the direction of the stream.

xdrrec_create()

Initialize XDR Stream

Syntax

```
#include <RPC/rpc.h>
void xdrrec_create(
   XDR *xdrs,
   u_int sendsize,
   u_int recvsize,
   caddr_t handle,
   int (*readit)(),
   int (*writeit)())
```

Description

`xdrrec_create()` initializes the stream object pointed to by `xdrs`. Specifying values of 0 for `sendsize` or `recvsize` causes the system to choose suitable defaults. When a stream's output buffer is full, `writeit` is called. Similarly, when a stream's input buffer is empty, `readit` is called.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`sendsize`	Size of stream's outgoing data buffer.
`recvsize`	Size of stream's incoming data buffer.
`handle`	Client handle.
`readit`	Procedure for input empty condition.
`writeit`	Procedure for output full condition.

Note

This stream implements an intermediate record stream. Therefore, additional bytes in the stream provide record boundary information.

xdrrec_endofrecord()

Mark End of Record in XDR Stream

Syntax

```
#include <RPC/rpc.h>
bool_t xdrrec_endofrecord(
   XDR *xdrs,
   bool_t sendnow)
```

Description

`xdrrec_endofrecord()` marks the end of record in an XDR stream. This routine can be invoked only on record streams. The data in the output buffer is marked as a completed record. The output buffer is optionally written out if `sendnow` equals TRUE.

If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`sendnow`	Send flag.

xdrrec_eof()

Check EOF on XDR Stream

Syntax

```
#include <RPC/rpc.h>
bool_t xdrrec_eof(XDR *xdrs)
```

Description

`xdrrec_eof()` checks for an end-of-file condition on an XDR stream. After using the rest of the current record in the stream, this routine returns 1 if the stream has no more input. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Pointer to a stream.

xdrrec_skiprecord()

Skip Current Record in XDR Stream

Syntax

```
#include <RPC/rpc.h>
bool_t xdrrec_skiprecord(XDR *xdrs)
```

Description

`xdrrec_skiprecord()` skips the rest of the current record in the stream's input buffer. If successful, this routine returns a value of 1. Otherwise, it returns a value of 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Pointer to a stream.

xprt_register()

Register Transport Handle

Syntax

```
#include <RPC/rpc.h>
void xprt_register(SVCXPRT *xprt)
```

Description

`xprt_register()` registers the transport handle `xprt`. After service transport handles are created, they should register themselves with the service package. Service implementors usually do not need to use this routine.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xprt`	Service transport handle.

xprt_unregister()

Unregister Service Transport Handle

Syntax

```
#include <RPC/rpc.h>
void xprt_unregister(SVCXPRT *xprt)
```

Description

`xprt_unregister()` unregisters a service transport handle. Before a service transport handle is destroyed, it should unregister itself with the service package. Service implementors usually do not need to use this routine.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xprt`	Service transport handle.

xdrtdio_create()

Initialize XDR Stream Object

Syntax

```
#include <RPC/rpc.h>
void xdrstdio_create(
   XDR *xdrs,
   FILE *file,
   enum xdr_opop)
```

Description

`xdrstdio_create()` initializes the stream object pointed to by `xdrs`. The stream data is written to, or read from, `file`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`xdrs`	Stream.
`file`	Name of file containing stream data.
`op`	Determines the direction of the stream.

Index

Symbols

A

B

C

D

E

F

G

I

L

M

O

P

R

S

T

U

X

Product Discrepancy Report

To: Microware Customer Support

FAX: 515-224-1352

From:_____

Company:_____

Phone:_____

Fax:_____Email:_____

Product Name:

Description of Problem:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Host Platform_____

Target Platform_____

RadiSys.

MICROWARE SOFTWARE