IBM

eserver

iSeries

Remote Procedure Call (RPC) APIs

Version 5 Release 3

iSeries

Remote Procedure Call (RPC) APIs

Version 5 Release 3

Note

Before using this information and the product it supports, be sure to read the information in "Notices," on page 129.

Sixth Edition (August 2005)

This edition applies to version 5, release 3, modification 0 of Operating System/400 (product number 5722-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

Contents

Remote Procedure Call (RPC) APIs

The Remote Procedure Call (RPC) APIs include:

- "Authentication APIs"
- "Name-to-Address Translation APIs" on page 7
- "Network Selection APIs" on page 22
- "Transport-Independent Remote Procedure Call APIs" on page 31
- "External Data Representation APIs" on page 88

These APIs are intended for programmers who develop distributed applications. They enable distributed applications to communicate with each other. Open Networking Computers (ONC) RPC was developed by Sun Microsystems and is used to easily separate and distribute a client application from a server by using the SUN RPC protocol. RPC includes a method of abstracting data, called eXternal Data Representation, or XDR, to allow communications to be abstracted at the API level.

Transport-Independent RPC (TI-RPC), or ONC+ RPC, is the latest incantation of RPC. It provides a method of abstracting the underlying protocol used at the network layer, providing a more seamless transition from one protocol to another.

Note: These functions use header (include) files from the library QSYSINC, which is optionally installable. Make sure QSYSINC is installed on your system before using any of the functions. See "Header Files for Remote Procedure Call APIs" on page 127 for the file and member name of each header file.

The following terms relate to the RPC applications:

RPCBind	A daemon program that allows client programs to obtain the aress of a service that is registered with the RPCBind daemon.
RPCGen	A compiler that accepts a remote-program interface definition written in the RPC language (RPCL), which is similar to the C programming language. From this definition, RPCGen produces C-language output for client stub functions, a server skeleton, XDR filter routines, and a header file.

For more information on RPCBind and RPCGen, see the Control Language topic.

For more information about these APIs, see Sun TI-RPC distributed applications in the Information Center.

Top | APIs by category

APIs

These are the APIs for this category.

Authentication APIs

The authentication APIs are used to provide authentication to the Transport-Independent Remote Procedure Call (TI-RPC) applications. These APIs enable a client to pass appropriate information as required by a remote service.

The authentication APIs are:

- "authnone_create()—Create Null Authentication" (Create null authentication) creates and returns a default RPC authentication handle that passes null authentication information with each remote procedure call.
- "authsys_create()—Create Authentication with OS Permission" on page 4 (Create authentication with OS permission) creates and returns an RPC authentication handle that contains authentication information.
- "auth_destroy()—Destroy Authentication Information" on page 6 (Destroy authentication information) destroys the authentication information structure that is pointed to by the auth parameter.

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

authnone_create()—Create Null Authentication

Syntax

```
#include <rpc/rpc.h>

AUTH *authnone_create();
```

Default Public Authority: *USE

Service Program Name: QZNFTRPC

Threadsafe: No

The **authnone_create()** function creates and returns a default RPC authentication handle that passes null authentication information with each remote procedure call.

Parameters

None.

Authorities

No authorization is required.

Return Value

auth	Upon successful completion, this API returns a pointer to an RPC authentication handle.
NULL	**authnone_create()** was not successful. The *errno* variable is set to indicate the reason.

Error Conditions

[ENOMEM]	Storage allocation failed.
[EUNKNOWN]	Unknown System State.

Error Messages

Message ID	Error Message Text
CPIA1B0 I	An authentication problem was encountered by one of the TI-RPC APIs.
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- "authsys_create()—Create Authentication with OS Permission" on page 4

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **authnone_create()** is used:

```
#include <stdio.h>
#include <rpc/rpc.h>
/* Define remote program number and version */
#define RMTPROGNUM (u_long)0x3fffffffL
#define RMTPROGVER (u_long)0x1

main()
{
  CLIENT *client;  /* client handle */

  ...

  /* Create a null authentication */
  client->cl_auth = authnone_create();
  if (client->cl_auth == (AUTH *)NULL){
    fprintf(stderr, "authnone_create failed!!\n");
    exit(1);
  }

  ...

}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

authsys_create()—Create Authentication with OS Permission

Syntax

```
#include <rpc/rpc.h>

AUTH *authsys_create(const char *host,
                     const uid_t uid,
                     const gid_t gid,
                     const int len,
                     const gid_t *aup_gids);
```

Default Public Authority: *USE

Service Program Name: QZNFTRPC

Threadsafe: No

The **authsys_create()** function creates and returns an RPC authentication handle that contains authentication information.

Parameters

host (Input)
A pointer to the name of the machine on which the permission was created.

uid (Input)
The caller's effective user ID (UID).

gid (Input)
The caller's effective group ID (GID).

len (Input)
The length of the group's array.

aup_gids (Input)
A pointer to the counted array of groups to which the user belongs.

Authorities

No authorization is required.

Return Value

auth	Upon successful completion, this API returns an RPC authentication handle.
NULL	**authsys_create()** was not successful. The *errno* variable is set to indicate the reason.

Error Conditions

[EINVAL]	An invalid *len* parameter was passed.
[ENOMEM]	Storage allocation failed.
[EUNKNOWN]	Unknown system state.

Error Messages

Message ID	Error Message Text
CPIA1B0 I	An authentication problem was encountered by one of the TI-RPC APIs.
CPDA1C1 D	An authentication problem has occurred.
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- "authnone_create()—Create Null Authentication" on page 2

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **authsys_create()** is used:

```
#include <stdio.h>
#include <rpc/rpc.h>

/* Define remote program number and version */
#define RMTPROGNUM (u_long)0x3fffffffL
#define RMTPROGVER (u_long)0x1

main()
{
  CLIENT *client;  /* The client handle */
  char *host;
  uid_t uid;
  gid_t gid, *aup_gids;
  int len;

  /* Service request to host RPCSERVER_HOST */
  client = clnt_create("RPCSERVER_HOST", RMTPROGNUM, RMTPROGVER,
                       "tcp");
 if (client == (CLIENT *)NULL) {
   printf("Could not create client\n");
   exit(1);
  }

  ...

  uid = geteuid();
  gid = getegid();
  len = getgroups(NGRPS, aup_gids));
  /* Initialized the authsys_create()'s arguments before use */
  client->cl_auth = authsys_create(host, uid, gid,
                                             len, aup_gids);
  if (client->cl_auth == (AUTH *)NULL) {
    fprintf(stderr, "authsys_create failed!!\n");
    exit(1);
  }

  ...

}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

auth_destroy()—Destroy Authentication Information

Syntax

```
#include <rpc/rpc.h>

void auth_destroy(AUTH *auth);
```

Default Public Authority: *USE

Service Program Name: QZNFTRPC

Threadsafe: No

The **auth_destroy()** function destroys the authentication information structure that is pointed to by the *auth* parameter.

Parameters

auth

(Input)
A pointer to the authentication information structure to be destroyed. By destroying the *auth* structure, you deallocate private data structures.

Authorities

No authorization is required.

Return Value

None.

Error Conditions

None.

Error Messages

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- "authsys_create()—Create Authentication with OS Permission" on page 4
- "authnone_create()—Create Null Authentication" on page 2

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **auth_destroy()** is used:

```
#include <stdio.h>
#include <rpc/rpc.h>

/* Define remote program number and version */
#define RMTPROGNUM (u_long)0x3fffffffL
#define RMTPROGVER (u_long)0x1

main()
{
  CLIENT *clnt; /* The client handle */

  /*
    Create the client handle, and initialize the authentication in
    the clnt->cl_auth struct
  */
  clnt = clnt_create("RPCSERVER_HOST", RMTPROGNUM, RMTPROGVER,
                      "tcp");
  if (clnt == (CLIENT *)NULL) {
    printf("Could not create client\n");
    exit(1);
  }

  ...

  /*
    Destroy the authentication information associated with
    clnt->cl_auth
  */
  auth_destroy(clnt->cl_auth);

  ...

}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

Name-to-Address Translation APIs

The name-to-address translation APIs allow an application to obtain the address of a service on a specified host in a transport-independent manner. These APIs are typically used by the applications that use the expert level TI-RPC APIs.

The name-to-address translation APIs are:

- "netdir_free()—Free Netdir Structures" on page 8 (Free netdir structures) frees structures that are allocated by name-to-address translation APIs.
- "netdir_getbyaddr()—Translate a Netbuf Address to a Host" on page 9 (Translate a netbuf address to a host) maps addresses into host names and service names.
- "netdir_getbyname()—Translate a Given Host-Service Pair" on page 11 (Translate a given host-service pair) maps the host name and service name that are specified in the service parameter to a set of addresses that are consistent with the transport identified in the netconfig structure.
- "netdir_options()—Access Transport-Specific Capabilities" on page 14 (Access transport-specific capabilities) provides interfaces to transport-specific capabilities such as the broadcast address and reserved port facilities of TCP and UDP.
- "netdir_sperror()—Indicate an Error in an NTA Routine" on page 16 (Indicate an error in an NTA Routine) issues an informational message that states why one of the name-to-address translation APIs may have failed.

- “taddr2uaddr()—Translate a Local Address” on page 18 (Translate a local address) translates a transport-specific (local) address to a transport-independent (universal) address.
- “uaddr2taddr()—Translate a Universal Address” on page 20 (Translate a universal address) translates a transport-independent (universal) address to a transport-specific (local) address (netbuf structure).

Top | “Remote Procedure Call (RPC) APIs,” on page 1 | APIs by category

netdir_free()—Free Netdir Structures

Syntax

```
#include <netdir.h>

void netdir_free(void *ptr,
                 int struct_type);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **netdir_free()** function frees structures that are allocated by name-to-address translation APIs.

Parameters

ptr (Input)
A pointer to a structure that is to be freed.

struct_type (Input)
The integer value that indicates to **netdir_free()** which type of structure to be freed.

The following combination is supported:

ND_HOSTSERV	A pointer to an nd_hostserv structure.
ND_HOSTSERVLIST	A pointer to an nd_hostservlist structure.
ND_ADDR	A pointer to a netbuf structure.
ND_ADDRLIST	A pointer to an nd_addrlist structure.

Authorities

No authorization is required.

Error Conditions

If **netdir_free()** takes an exception, *nd_errno* is set to the following error:

[ND_SYSTEM]	A damaged object was encountered. The damaged object cannot be used.

Error Messages

Message ID	Error Message Text
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Usage Notes

netdir_free() frees the structure allocated by the netdir APIs. The type of structure to be freed is indicated by the struct_type.

Refer to other name-to-address translation functions to see how **netdir_free()** function is used.

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

netdir_getbyaddr()—Translate a Netbuf Address to a Host

Syntax

```
#include <netdir.h>

int netdir_getbyaddr(struct netconfig *nconf,
                     struct nd_hostservlist
                          **service,
                     struct netbuf *netaddr);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **netdir_getbyaddr()** function maps addresses into host names and service names.

Parameters

nconf (Input)
A pointer to a netconfig structure that is returned by either **getnetconfig()** or **getnetconfigent()**.

service (Output)
A pointer to a list of service names.

netaddr (Input)
A pointer to the address.

Authorities

No authorization is required.

Return Value

0	**netdir_getbyaddr()** was successful. A list of host names and service name pairs is returned in *service*.
-1	**netdir_getbyaddr()** was not successful. The *nd_errno* (defined in **<netdir.h>**) is set to indicate the error.

Error Conditions

If **netdir_getbyaddr()** is not successful, *nd_errno* usually indicates one of the following errors:

[ND_BADARG]	Bad argument passed.
[ND_NO_DATA]	The host name is a valid name but there is no corresponding IP address.
[ND_NOHOST]	The host name that the user specified by the host address was not found.
[ND_NOMEM]	Not enough memory left.
[ND_NO_RECOVERY]	An unrecoverable error has occurred.
[ND_SYSTEM]	A damaged object was encountered. The damaged object cannot be used.
[ND_TRY_AGAIN]	The local server did not receive a response from an authoritative server. An attempt at a later time may succeed.

Error Messages

Message ID	**Error Message Text**
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Usage Notes

netdir_getbyaddr() is called with an address in the *netaddr* structure.

The caller is responsible to free the storage allocated by **netdir_getbyaddr()** by using the function **netdir_free()**.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **netdir_getbyaddr()** is used:

```
#include <netdir.h>

void findhost(void)
{

   void *handlep;
   struct netconfig *nconf;
   struct nd_hostservlist *nd_hostserv;
   struct netbuf nbuf;
   char uaddr[16];

   /* Initialize the network selection mechanism */
   if (handlep = setnetconfig()) == (void *)NULL)
   {
      exit(1);
   }

   /* Get the netconfig handle */
```

```
    if ((nconf = getnetconfig(handlep)) == (struct netconf *)NULL)
    {
        printf("Error getting the netconfig handle\n");
        exit(1);
    }
    memset(uaddr, NULL, 16);
    printf("Enter the host IP address appended by low and high order
            port numbers:\n");
    scanf("%s", uaddr);

    /* Convert universal address notation into transport-specific
     * address format.
     */
    nbuf = uaddr2taddr(nconf, uaddr);

    /* Get the hostname from the address over the transport */
    /* provider specified in the netconfig structure       */
    if (netdir_getbyaddr(nconf, &nd_hostserv, &nbuf)
        != ND_OK)
    {
        printf("Cannot determine the host\n");
        exit(1);
    }
    printf("The host name is: %s\n",
    nd_hostserv->h_hostservs->h_host);
    printf("The Service is: %s\n", nd_hostserv->h_hostservs->h_serv);

    /* Free the netdir structure allocated by netdir_getbyname() */
    netdir_free(nd_hostserv, ND_HOSTSERVLIST);

   /* Release the netconfig handle allocated by set setnetconfig() */
   endnetconfig(handlep);
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

netdir_getbyname()—Translate a Given Host-Service Pair

Syntax

```
#include <netdir.h>

int netdir_getbyname(struct netconfig *nconf,
                     struct nd_hostserv *service,
                     struct nd_addrlist **addrs);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **netdir_getbyname()** function maps the host name and service name that are specified in the *service* parameter to a set of addresses that are consistent with the transport identified in the *netconfig* structure.

Parameters

nconf (Input)
A pointer to a netconfig structure that is returned by either **getnetconfig()** or **getnetconfigent()**.

service (Input)
A pointer to a service name.

addrs (Output)
A pointer to the addresses being returned.

Authorities

No authorization is required.

Return Value

0	**netdir_getbyname()** was successful.
-1	**netdir_getbyname()** was not successful. The *nd_errno* (defined in **<netdir.h>**) is set to indicate the error.

Error Conditions

If **netdir_getbyname()** is not successful, *nd_errno* usually indicates one of the following errors:

[ND_BADARG]	Bad argument passed.
[ND_NOHOST]	The host that was specified by the host name was not found.
[ND_NOMEM]	Not enough memory left.
[ND_NO_RECOVERY]	An unrecoverable error has occurred.
[ND_NOSERV]	Service name is unknown.
[ND_SYSTEM]	A damaged object was encountered. The damaged object cannot be used.
[ND_TRY_AGAIN]	The local server did not receive a response from an authoritative server. An attempt at a later time may succeed.

Error Messages

Message ID	**Error Message Text**
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Usage Notes

netdir_getbyname() maps the host and service name to a set of addresses consistent with the transport specified in *netconfig*.

The caller is responsible to free the storage allocated by **netdir_getbyname()** by using the function **netdir_free()**.

netdir_getbyname() does not support HOST_ANY or HOST_BROADCAST for host names specified in the *nd_hostserv* structure.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **netdir_getbyname()** is used:

```
#include <netdir.h>

main()
{
  void *handlep;                  /* A handle into network selection */
  struct netconfig *nconf;                  /* transport information */
  struct nd_hostserv nd_hostserv;    /* host and service information */
  struct nd_addrlist *nd_addrlistp;     /* addresses for the service */

  /* Initialize the network selection mechanism */
  if (handlep = setnetconfig()) == (void *)NULL)
  {
     exit(1);
  }

  /* Get the netconfig handle */
  if ((nconf = getnetconfig(handlep)) == (struct netconf *)NULL)
  {
     printf("Error in getting the netconfig handle.\n");
     exit(1);
  }

  /* Allocate memory for host and service names */
  nd_hostserv.h_host = (char *)malloc(24);
  nd_hostserv.h_serv = (char *)malloc(24);
  if ((nd_hostserv.h_host == (char *)NULL)
     || (nd_hostserv.h_serv == (char *)NULL))
   {
     printf("No memory available.  netdir_getbyname()
     failed.\n");
     exit(1);
   }

   printf("Enter the hostname:\n");
   scanf("%s", nd_hostserv.h_host);
   printf("Enter the service name:\n");
   scanf("%s", nd_hostserv.h_serv);

   /* Get the address for the service on the host on the
    * transport provider specified in the netconfig structure
    */
   if (netdir_getbyname(nconf, &nd_hostserv, &nd_addrlistp)
      != ND_OK)
   {
      printf("Cannot determine address for service\n");
      exit(1);
   }
   printf("The address of the <%s> service on host
          <%s> was found.\n", nd_hostserv.h_serv,
          nd_hostserv.h_host);

   /* Free the netdir structure allocated by netdir_getbyname() */
   netdir_free(nd_addrlistp, ND_ADDRLIST);

  /* Release the netconfig handle allocated by set setnetconfig() */
  endnetconfig(handlep);
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

netdir_options()—Access Transport-Specific Capabilities

Syntax

```
#include <netdir.h>

int netdir_options(struct netconfig *nconf,
                   int option,
                   int fd,
                   char *point_to_args);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **netdir_options()** function provides interfaces to transport-specific capabilities such as the broadcast address and reserved port facilities of TCP and UDP.

Parameters

nconf (Input)
A pointer to a netconfig structure that specifies a transport.

option (Input)
Specifies the transport-specific action to take. The following values may be used for *option*:

ND_SET_BROADCAST	Set the transport for broadcast if supported.
ND_SET_RESERVEDPORT	Let the application bind to a reserved port if allowed by the transport.
ND_CHECK_RESERVEDPORT	Verify that an address corresponds to a reserved port if the transport supports reserved ports.
ND_MERGEADDR	Transform a locally meaningful address into an address that the client host can connect to.

fd (Input)
The file descriptor that may or may not be used based on the option. The only value supported for this field is **RPC_ANYFD**. The file descriptor value is used only if the specified option is ND_SET_BROADCAST or ND_SET_RESERVEDPORT.

point_to_args (Input)
A pointer to the operation-specific data.

Authorities

The caller must have the *IOSYSCFG special authority to bind to a reserved port.

Return Value

0	netdir_options() was successful.
-1	**netdir_options()** was not successful. The *nd_errno* global variable (defined in **<netdir.h>**) is set to indicate the error.

Error Conditions

If **netdir_options()** is not successful, *nd_errno* indicates one of the following errors:

[ND_ACCESS]	The user does not have permission to use the specified address.
[ND_BADARG]	Bad argument passed. A file descriptor that was not valid was passed to the API.
[ND_FAILCTRL]	Control operation failed.
[ND_NO_ADDRESS]	Bad address.
[ND_NOCONVERT]	Conversion error. One or more characters could not be converted from the source CCSID to the target CCSID.
[ND_NOCTRL]	The function was used in the wrong sequence. An incorrect option was specified.
[ND_NO_DATA]	Incorret amount of data.
[ND_NOHOST]	The host that was specified by the host name was not found.
[ND_NOMEM]	Not enough memory left.
[ND_NO_RECOVERY]	An unrecoverable error has occurred.
[ND_OPEN]	File could not be opened.
[ND_SYSTEM]	A damaged object was encountered. The damaged object cannot be used. The system detected an address that was not valid.
[ND_TRY_AGAIN]	The local server did not receive a response from an authoritative server. An attempt at a later time may succeed.

Error Messages

Message ID	**Error Message Text**
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **netdir_options()** is used:

```
#include <netdir.h>
#include <rpc/rpc_com.h>   /* for RPC_ANYFD definition */

main()
{
  void *handlep;
  struct netconfig *nconf;

  /* Initialize the network selection mechanism */
  if (handlep = setnetconfig()) == (void *)NULL)
  {
     exit(1);
  }

  /* Get a netconfig structure from the netconfig file */
  if ((nconf = getnetconfig(handlep)) == (struct netconf *)NULL)
  {
     printf("Unable to obtain a netconfig structure\n");
  }
  /* Set the protocol specific negotiation for broadcast */
  if (netdir_options(nconf, ND_SET_BROADCAST, RPC_ANYFD, NULL))
  {
     printf("Error setting the broadcasting option\n");
  }

  /* Release the netconfig handle allocated by setnetconfig() */
  endnetconfig(handlep);
}
```

API introduced: V4R2

Top | “Remote Procedure Call (RPC) APIs,” on page 1 | APIs by category

netdir_sperror()—Indicate an Error in an NTA Routine

Syntax

```
#include <netdir.h>

void  netdir_sperror();
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **netdir_sperror()** function issues an informational message that states why one of the name-to-address translation APIs may have failed.

Parameters

None.

Authorities

No authorization is required.

Return Value

None

netdir_sperror() issues an informational message that indicates the error in one of the name-to-address translation APIs.

Error Messages

Message ID	**Error Message Text**
CPIA1B7	The previous name-to-address translation has completed.

Usage Notes

The **netdir_sperror()** function issues CPIA1B7 message that indicates why one of the name-to-address translation mapping APIs failed. This function should be used after a failed call to a name-to-address translation function prior to calling another name-to-address translation function.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **netdir_sperror()** is used:

```
#include <netdir.h>
#include <rpc/rpc_com.h>

main()
{
  void *handlep;
  struct netconfig *nconf;

  /* Initialize the network selection mechanism */
  if (handlep = setnetconfig()) == (void *)NULL)
  {
     exit(1);
  }

  /* Get a netconfig structure from the netconfig file */
  if ((nconf = getnetconfig(handlep)) == (struct netconf *)NULL)
  {
     printf("Unable to obtain a netconfig structure\n");
  }

  /* Set the protocol specific negotiation for broadcast */
  if (netdir_options(nconf, ND_SET_BROADCAST, RPC_ANYSOCK, NULL))
  {
     printf("Error setting the broadcasting option\n");
     printf("See the job log for error message\n");
     netdir_sperror();
  }

  /* Release the netconfig handle allocated by setnetconfig() */
  endnetconfig(handlep);
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

taddr2uaddr()—Translate a Local Address

Syntax

```
#include <netdir.h>

char *taddr2uaddr(struct netconfig *nconf,
                  struct netbuf *addr);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **taddr2uaddr()** function translates a transport-specific (local) address to a transport-independent (universal) address.

Parameters

nconf (Input)
The transport for which the address is valid.

addr (Input)
The address to be translated to the universal representation.

Authorities

No authorization is required.

Return Value

universal address	A string that contains the universal address is returned if the function **taddr2uaddr()** was successful.
NULL	A NULL pointer is returned if the function **taddr2uaddr()** was not successful. The *nd_errno* global variable (defined in **<netdir.h>**) is set to indicate the error.

Error Conditions

If the function **taddr2uaddr()** is not successful, *nd_errno* usually indicates the following error:

[ND_BADARG]	Bad argument passed.
[ND_SYSTEM]	A damaged object was encountered. The damaged object cannot be used.

Error Messages

Message ID	Error Message Text
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Usage Notes

taddr2uaddr() translates the address pointed to by *addr* and returns a transport independent character representation of the address (universal address).

The caller is responsible to free the returned universal address when done.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **taddr2uaddr()** is used:

```
#include <netconfig.h>
#include <netdir.h>

main()
{
  void *handlep;                  /* A handle into network selection */
  struct netconfig *nconf;                  /* Transport information */
  struct nd_hostserv nd_hostserv;    /* Host and service information */
  struct nd_addrlist *nd_addrlistp;     /* Addresses for the service */
  struct netbuf *netbufp;              /* The address of the service */
  int i;                                  /* The number of addresses */
  char *uaddr;                          /* Service universal address */

  /* Initialize the network selection mechanism */
  if (handlep = setnetconfig()) == (void *)NULL)
  {
     exit(1);
  }

  /* Get the netconfig handle */
  if ((nconf = getnetconfig(handlep)) == (struct netconf *)NULL)
  {
     printf("Error in getting the netconfig handle.\n");
     exit(1);
  }

   /* Get the address for service specified in nd_hostserv.h_serv
    * on the host specified in nd_hostserv.h_host over the
    * transport provider specified in the netconfig structure
    * Note: nd_hostserv.h_host and nd_hostserv.h_serv need to be
    * set up prior to the call to netdir_getbyname().
    */
   if (netdir_getbyname(nconf, &nd_hostserv, &nd_addrlistp)
        != ND_OK)
   {
      printf("Cannot determine address for service\n");
      /* Release the netconfig handle allocated by setnetconfig() */
      endnetconfig(handlep);
      exit(1);
   }

   /* Convert the transport-specific address into universal address
    * notation and print it.
    */
   netbufp = nd_addrlistp->n_addrs;
   uaddr = taddr2uaddr(nconf, netbufp);
   if (uaddr != NULL)
   {
      printf("The address of the service %s on host %s is %s\n",
              nd_hostserv.h_serv, nd_hostserv.h_host, uaddr);
      free(uaddr);
   }

   /* Free the netdir structure allocated by netdir_getbyname() */
   netdir_free(nd_addrlistp, ND_HOSTSERVLIST);

  /* Release the netconfig handle allocated by setnetconfig() */
  endnetconfig(handlep);
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

uaddr2taddr()—Translate a Universal Address

Syntax

```
#include <netdir.h>

struct netbuf *uaddr2taddr(struct netconfig *nconf,
                           char *uaddr);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **uaddr2taddr()** function translates a transport-independent (universal) address to a transport-specific (local) address (netbuf structure).

Parameters

nconf (Input)
The transport for which the address is valid.

uaddr (Input)
The address to be translated to the netbuf structure.

Authorities

No authorization is required.

Return Value

netbuf structure	uaddr2taddr() was successful.
NULL	**uaddr2taddr()** was not successful. The *nd_errno* (defined in **<netdir.h>**) is set to indicate the error.

Error Conditions

If **uaddr2taddr()** is not successful, *nd_errno* usually indicates one of the following errors:

[ND_BADARG]	Bad argument passed.
[ND_NOMEM]	Not enough memory left.
[ND_NO_RECOVERY]	An unrecoverable error has occurred.
[ND_SYSTEM]	A damaged object was encountered. The damaged object cannot be used.

Error Messages

Message ID	Error Message Text
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Usage Notes

uaddr2taddr() translates the universal address pointed to by *addr* and returns a pointer to a netbuf structure.

It is the caller's responsibility to free the returned netbuf structure when done using the **netdir_free()** function.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **uaddr2taddr()** is used:

```
#include <netconfig.h>
#include <netdir.h>

Void sample (void)
{

   void *handlep;
   struct netconfig *nconf;
   struct netbuf *netbufp;
   char universal_addr[24];
   int i;

   /* Initialize the network selection mechanism */
   if (handlep = setnetconfig()) == (void *)NULL)
   {
      exit(1);
   }

   /* Get the transport information */
   if ((nconf = getnetconfig(handlep)) == (struct netconf *)NULL)
   {
      printf("Error in getting the transport information\n"E);
      exit(1);
   }

   memset(universal_addr,24,NULL);
   printf("EEnter the IP address appended by low and high order
           port numbers:\n"E);
   scanf(%s, universal_addr);

   /* Convert the input universal address to its local representation */
   if ((netbufp = uaddr2taddr(nconf, universal_addr)) ==
                  (struct netbuf *) NULL)
   {
       printf("Euaddr2taddr() failed\n"E);
   }

   /*Free the netbuf structure returned from uaddr2taddr() */
   netdir_free((char *)netbufp, ND_ADDR);

   /* Release the netconfig handle allocated by setnetconfig() */
   endnetconfig(handlep);

   return;
}
```

API introduced: V4R2

Top | “Remote Procedure Call (RPC) APIs,” on page 1 | APIs by category

Network Selection APIs

The network selection APIs provide the means to choose the transport on which an application should run. These APIs are typically used by the applications that use the intermediate-level and expert-level TI-RPC APIs.

The network selection APIs are:

- “endnetconfig()—Release the Pointer in the Netconfig File” (Release the pointer in the netconfig file) releases the pointer to the records stored in the netconfig file.
- “freenetconfigent()—Free the Netconfig Structure” on page 24 (Free the netconfig structure) frees the netconfig structure that is returned from the call to the getnetconfigent() function.
- “getnetconfig()—Return Current Record from the Netconfig File” on page 26 (Return current record from the netconfig file) returns the pointer to the current record in the netconfig file and increments its pointer to the next record.
- “getnetconfigent()—Return a Pointer to a Netconfig Structure” on page 27 (Return a pointer to a netconfig structure) returns the pointer to the netconfig structure that corresponds to the input netid.
- “setnetconfig()—Initialize the Pointer in the Netconfig File” on page 29 (Initialize the pointer in the netconfig file) initializes the record pointer to the first entry in the netconfig file.

Top | “Remote Procedure Call (RPC) APIs,” on page 1 | APIs by category

endnetconfig()—Release the Pointer in the Netconfig File

Syntax

```
#include <netconfig.h>

int endnetconfig (void *);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **endnetconfig()** function releases the pointer to the records stored in the **netconfig** file.

Parameters

void pointer (Input)
A void pointer that is set by a call to the **setnetconfig()** function.

Authorities

No authorization is required.

Return Value

0	**endnetconfig()** was successful. The pointer to the netconfig structure in the netconfig file is released. This function is always successful.

Error Conditions

When an exception occurs, **endnetconfig()** is trying to free the handle to the /etc/netconfig file. If **endnetconfig()** is not successful, *errno* indicates the following error.

[EUNKNOWN] Unknown system state.

The operation failed because of an unknown system state. See any messages in the job log and correct any errors that are indicated. Then retry the operation.

Error Messages

Message ID	**Error Message Text**
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Usage Notes

endnetconfig() API must be used to release the pointer to the netconfig structure obtained by a call to the **setnetconfig()** API.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **endnetconfig()** is used:

```
#include <netconfig.h>

main()
{
 void *handlep;
 struct netconfig *nconf;

 /* Initialize the network selection mechanism */
 if ((handlep = setnetconfig()) == (void *)NULL)
 {
   exit(1);
 }

 /* Loop through the transport providers */
 while ((nconf = getnetconfig(handlep)) != (struct netconfig *) NULL)
 {
    /* Print out the information associated with the */
    /* transport providers described in the          */
    /* "netconfig" structure.                        */
    printf("Transport provider name: %s\n", nconf->nc_netid);
    switch(nconf->nc_semantics)
    {
       case NC_TPI_CLTS:
          printf("Transport type name: TPI_CLTS\n");
          break;
       case NC_TPI_COTS:
          printf("Transport type name: TPI_COTS\n");
          break;
       case NC_TPI_COTS_ORD:
          printf("Transport type name: TPI_COTS_ORD\n");
          break;
       default:
          break;
    }
    switch(nconf->nc_flag)
    {
       case 0:
          printf("Transport flag name: N\n");
          break;
       case 1:
          printf("Transport flag name: V\n");
          break;
       default:
          break;
    }
    printf("Transport family name: %s\n", nconf->nc_protofmly);
    printf("Transport protocol name: %s\n", nconf->nc_proto);
 }

 /*Release the netconfig handle allocated by setnetconfig() */
 endnetconfig(handlep);
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

freenetconfigent()—Free the Netconfig Structure

Syntax

```
#include <netconfig.h>

void freenetconfigent(struct netconfig *);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **freenetconfigent()** function frees the netconfig structure that is returned from the call to the **getnetconfigent()** function.

Parameters

netconfig (Input)
A pointer to a netconfig structure that is set by a call to the **setnetconfig()** function.

Authorities

No authorization is required.

Return Value

None.

Error Conditions

If an exception occurs, **freenetconfigent()** fails to free the netconfig structure. If **freenetconfigent()** is not successful, *errno* indicates the following error.

[EUNKNOWN]	Unknown system state. The operation failed because of an unknown system state. See any messages in the job log and correct any errors that are indicated. Then retry the operation.

Error Messages

Message ID	Error Message Text
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **freenetconfigent()** is used:

```
#include <netconfig.h>

main()
{
 struct netconfig *nconf;

 /* Assuming UDP is a netid on the system, get the netconfig structure *
 if ((nconf = getnetconfigent("UDP")) == (struct netconfig *)NULL)
 {
   printf("There is no information about UDP\n");
   exit(1);
 }

 /* Print out the information associated with the transport */
 /* identified with the netid of UDP                         */
 printf("Transport provider name: %s\n", nconf->nc_netid);
    switch(nconf->nc_semantics)
    {
       case NC_TPI_CLTS:
          printf("Transport type name: TPI_CLTS\n");
          break;
       case NC_TPI_COTS:
          printf("Transport type name: TPI_COTS\n");
          break;
       case NC_TPI_COTS_ORD:
          printf("Transport type name: TPI_COTS_ORD\n");
          break;
       default:
          break;
    }
    switch(nconf->nc_flag)
    {
       case 0:
          printf("Transport flag name: N\n");
          break;
       case 1:
          printf("Transport flag name: V\n");
          break;
       default:
          break;
    }
 printf("Transport family name: %s\n", nconf->nc_protofmly);
 printf("Transport protocol name: %s\n", nconf->nc_proto);

 /* Free the netconfig structure returned by getnetconfigent() */
 freenetconfigent(nconf);
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

getnetconfig()—Return Current Record from the Netconfig File

Syntax

```
#include <netconfig.h>

struct netconfig *getnetconfig(void *);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **getnetconfig()** function returns the pointer to the current record in the netconfig file and increments its pointer to the next record.

Parameters

void pointer (Input)
A void pointer that is set by a call to the **setnetconfig()** function.

Authorities

No authorization is required.

Return Value

netconfig	**getnetconfig()** was successful. A pointer to the current netconfig structure in the **netconfig** file is returned.
NULL	**getnetconfig()** was not successful. A NULL pointer is returned. The *errno* global variable is set to indicate the error.

Error Conditions

If **getnetconfig()** is not successful, *errno* usually indicates one of the following errors. Under some conditions, *errno* could indicate an error other than those listed here.

[EUNKNOWN]	Unknown system state. The operation failed because of an unknown system state. See any messages in the job log and correct any errors that are indicated. Then retry the operation.

Error Messages

Message ID	**Error Message Text**
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

For more information, see the example for "endnetconfig()—Release the Pointer in the Netconfig File" on page 22.

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

getnetconfigent()—Return a Pointer to a Netconfig Structure

Syntax

```
#include <netconfig.h>

struct netconfig *getnetconfigent(char *);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **getnetconfigent()** function returns the pointer to the netconfig structure that corresponds to the input netid.

Parameters

netid (Input)
A character pointer to a netid such as "tcp" or "udp".

Authorities

The caller of **getnetconfigent()** function must have execute (*X) authority to the **/etc** directory and must have read (*R) authority to the **netconfig** file.

Return Value

netconfig	**getnetconfigent()** was successful. A pointer to a netconfig structure is returned.
NULL	**getnetconfigent()** was not successful. The *errno* global variable is set to indicate the error.

Error Conditions

If **getnetconfigent()** is not successful, *errno* usually indicates one of the following errors. Under some conditions, *errno* could indicate an error other than those listed here.

[EACCES] Permission denied.

- An attempt was made to access an object in a way forbidden by its object access permissions.
- The job does not have access to the specified file, directory, component, or path.

[EAGAIN] Operation would have caused the process to be suspended.

[EBADNAME] The object name specified is not correct.

[EBUSY] Resource busy.

[ECONVERT] Conversion error.

- One or more characters could not be converted from the source CCSID to the target CCSID.

[EDAMAGE] A damaged object was encountered.

- A referenced object is damaged. The object cannot be used.

[EIO] Input/output error.

- A physical I/O error occurred. A reference object may be damaged.

[EMFILE] Too many open files for this process.

- An attempt was made to open more files than allowed by the value OPEN_MAX. The value of OPEN_MAX can be retrieved using the **sysconf()** function.

[ENFILE] Too many open files in the system.

- A system limit has been reached for the number of files that are allowed to be concurrently open in the system.

[ENOENT] No such path or directory.

- The directory or a component of the path name specified does not exist.
- A named file or directory does not exist or is an empty string.

[ENOMEM] Storage allocation request failed.

- The function needed to allocate storage, but no storage is available.
- There is not enough memory to perform the requested function.

[ENOSPC] No space available.

- The requested operations required additional space on the device and there is no space left. This could also be caused by exceeding the user profile storage limit when creating or transferring ownership of an object.
  - Insufficient space remains to hold the intended file.

[ENOSYSRSC] System resources not available to complete the request.

[EPERM] Operation not permitted.

- You must have appropriate privileges or other resources to do the requested operation.

[EUNKNOWN] Unknown system state.

- The operation failed because of an unknown system state. See any messages in the job log and correct any errors that are indicated. Then retry the operation.

Error Messages

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPFA0D4 E	File system error occurred.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPIA1C0 I	The file /etc/netconfig cannot be opened by readers because another job has it open with write authority.

Usage Notes

getnetconfigent() returns a pointer to a netconfig structure in the netconfig file for the corresponding netid. The netid is expected in the job CCSID. It returns NULL if it is unsuccessful.

The callers of the **getnetconfigent()** function do not need to call the **setnetconfig()** function prior to calling the **getnetconfigent()** function but must call the **freenetconfigent()** function to free the storage allocated by the **getnetconfigent()** function.

The **getnetconfigent()** function will return [ENOENT] if the /etc/netconfig file does not exist. The **getnetconfigent()** function will fail with [ECONVERT] if the data conversion required to convert the data stored in the /etc/netconfig file cannot be converted to the job CCSID.

Example

For more information, see the example for "freenetconfigent()—Free the Netconfig Structure" on page 24.

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

setnetconfig()—Initialize the Pointer in the Netconfig File

Syntax

```
#include <netconfig.h>

void *setnetconfig(void);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **setnetconfig()** function initializes the record pointer to the first entry in the netconfig file. The **setnetconfig()** function must be used before the first use of **getnetconfig()** function. The **setnetconfig()** function returns a unique handle (a pointer to the records stored in the netconfig file) to be used by the **getnetconfig()** function.

Parameters

None.

Authorities

The caller of **setnetconfig()** function must have execute (*X) authority to the **/etc** directory and must have read (*R) authority to the **netconfig** file.

Return Value

void pointer	**setnetconfig()** was successful. A void pointer to the records stored in the **netconfig** file is returned.
NULL	**setnetconfig()** was not successful. The *errno* global variable is set to indicate the error.

Error Conditions

If **setnetconfig()** is not successful, *errno* usually indicates one of the following errors. Under some conditions, *errno* could indicate an error other than those listed here.

[EACCES] Permission denied.

- An attempt was made to access an object in a way forbidden by its object access permissions.
- The job does not have access to the specified file, directory, component, or path.

[EAGAIN] Operation would have caused the process to be suspended.

[EBADNAME] The object name specified is not correct.

[EBUSY] Resource busy.

[ECONVERT] Conversion error.

- One or more characters could not be converted from the source CCSID to the target CCSID.

[EDAMAGE] A damaged object was encountered.

- A referenced object is damaged. The object cannot be used.

[EIO] Input/output error.

- A physical input/output error occurred. A reference object may be damaged.

[EMFILE] Too many open files for this process.

- An attempt was made to open more files than allowed by the value OPEN_MAX. The value of OPEN_MAX can be retrieved by using the **sysconf()** function.

[ENFILE] Too many open files in the system.

- A system limit has been reached for the number of files that are allowed to be concurrently open in the system.

[ENOENT] No such path or directory.

- The directory or a component of the path name specified does not exist.
- A named file or directory does not exist or is an empty string.

[ENOMEM] Storage allocation request failed.

- The function needed to allocate storage, but no storage is available.
- There is not enough memory to perform the requested function.

[ENOSPC] No space available.

- The requested operations required additional space on the device and there is no space left. This could also be caused by exceeding the user profile storage limit when creating or transferring ownership of an object.
- Insufficient space remains to hold the intended file.

[ENOSYSRSC] System resources not available to complete the request.

[EPERM] Operation not permitted.

- You must have appropriate privileges or other resources to do the requested operation.

[EUNKNOWN]	Unknown system state. • The operation failed because of an unknown system state. See any messages in the job log and correct any errors that are indicated. Then retry the operation.

Error Messages

Message ID	**Error Message Text**
CPE3418 E	Possible APAR condition or hardware failure.
CPFA0D4 E	File system error occurred.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPIA1C0 I	The file /etc/netconfig cannot be opened by readers because another job has it open with write authority.

Usage Notes

The **setnetconfig()** function is used prior to using the **getnetconfig()** function to initialize the record pointer to the data stored in the netconfig file.

The **setnetconfig()** function will fail with [ENOENT] if the /etc/netconfig file does not exist. The **setnetconfig()** function will fail with [ECONVERT] if the data conversion required to convert the data stored in the /etc/netconfig file cannot be converted to the job CCSID.

Example

For more information, see the example for "endnetconfig()—Release the Pointer in the Netconfig File" on page 22.

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

Transport-Independent Remote Procedure Call APIs

The Transport-Independent Remote Procedure Call (TI-RPC) functions allow distributed applications to communicate with each other in a transport independent fashion. These APIs are provided to perform Transport-Independent Remote Procedure Calls.

The TI-RPC APIs are divided into five separate sections:
- "Simplified APIs" on page 32
- "Top-level APIs" on page 37
- "Intermediate-level APIs" on page 50
- "Expert-level APIs" on page 56
- "Other APIs" on page 73 (These APIs work with the other four sections.)

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

Simplified APIs

The simplified interfaces specify the type of transport to use. Applications using this level do not have to explicitly create handles. These APIs combine all the API calls into one procedure and can be used to quickly develop an RPC service and corresponding client application.

The simplified APIs are:

- "rpc_call()—Call a Remote Procedure on the Specified System" (Call a remote procedure on the specified system) calls the remote procedure that is associated with prognum, versnum, and procnum on the machine, host.
- "rpc_reg()—Register a Procedure with RPC Service Package" on page 35 (Register a procedure with RPC service package) registers a procedure with the RPC service package (RPCBind).

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

rpc_call()—Call a Remote Procedure on the Specified System

Syntax

```
#include <rpc/rpc.h>

enum clnt_stat rpc_call(const char *host,
                        const u_long prognum,
                        const u_long versnum,
                        const u_log procnum,
                        const xdrproc_t inproc,
                        const char *in,
                        const xdrproc_t outproc,
                        char *out,
                        const char *nettype);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **rpc_call()** API calls the remote procedure that is associated with *prognum*, *versnum*, and *procnum* on the machine, *host*. **rpc_call()** tries all the transports of the nettype class available from the netconfig database file, and chooses the first successful one. Transports are tried in top-to-bottom order in the netconfig database file. A default time-out is set and can be modified using **clnt_control()**.

Parameters

host (Input)
A pointer to the program name of the remote machine.

prognum (Input)
The program number of the remote program.

vernum (Input)
The version number of the remote program.

procnum (Input)
The number of the procedure that is associated with the remote program being called.

inproc (Input)
The name of the XDR procedure that encodes the procedure parameters.

in (Input)
The address of the procedure arguments.

outproc (Input)
The name of the XDR procedure that decodes the procedure results.

out (Output)
The address where results are placed.

nettype (Input)
The following classes of transport protocol are valid and are represented as a string either in lowercase or in uppercase: NETPATH, VISIBLE, CIRCUIT_V, DATAGRAM_V, CIRCUIT_N, DATAGRAM_N, TCP, and UDP. When this parameter is NULL, NETPATH is assumed.

Authorities

The caller of the **rpc_call()** API must have execute (*X) authority to the /etc directory and must have read (*R) authority to the netconfig file.

Return Value

RPC_SUCCESS (0)	Successful
Non-zero value	rpc_call() was not successful. The *rpc_createerr* global structure is set to indicate the error.

Error Conditions

Upon failure, **rpc_call()** sets the global structure *rpc_createerr*. The *rpc_createerr.cf_stat* variable has a status value that indicates the error reason. The *rpc_createerr.cf_error.re_errno* variable is meaningful when some status values are set.

The *rpc_createerr.cf_stat* variable can be set to one of the following values:

This API calls **clnt_create()** and **clnt_call()** APIs in order to perform its task. All error conditions from those APIs are inherited except RPC_FAILED from **clnt_call()**.

[RPC_SYSTEMERROR] RPC error returned from system call. The *rpc_createerr.cf_error.re_errno* variable can be set to one of the following values:

[ENOMEM]
Out of memory.

[RPC_UNKNOWNHOST]
Unknown host.

Error Messages

Message ID	Error Message Text
CPIA1B1 I	A problem was encountered in the RPC client.
CPIA1B2 I	TI-RPC encountered a problem in the transport protocol.
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- "clnt_call()—Call a Remote Procedure Associated with the Client" on page 37

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **rpc_call()** is used:

```
/* Define remote program number and version */

#define RMTPROGNUM (u_long)0x3fffffffL
#define RMTPROGVER (u_long)0x1
#define RMTPROCNUM (u_long)0x1

#include <stdio.h>
#include <rpc/rpc.h>

main()
{
  int inproc=100, outproc;
  enum clnt_stat rstat;

  ...

  /* Service request to host RPCSERVER_HOST */
  if (rstat = rpc_call("as400.somewhere.ibm.com", RMTPROGNUM,
             RMTPROGVER, RMTPROCNUM, xdr_int, (char *)&inproc,
             xdr_int, (char *)&outproc, "VISIBLE")
             != RPC_SUCCESS){
    fprintf(stderr,"rpc_call() failed\n");
    exit(1);
  }
  ...
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

rpc_reg()—Register a Procedure with RPC Service Package

Syntax

```
=
#include <rpc/rpc.h>

bool_t rpc_reg(const u_long prognum,
               const u_long versnum,
               const u_long procnum,
               char *(*procname)(char *),
               const xdrproc_t inproc,
               const xdrproc_t outproc,
               const char *nettype);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **rpc_reg()** function registers a procedure with the RPC service package (**RPCBind**). If a request arrives that matches the values of the *prognum* parameter, the *versnum* parameter, and the *procnum* parameter, then the *procname* parameter is called with a pointer to its parameters. The *procname* returns a pointer to its static results.

The procedure is registered for each transport of the specified type (the *nettype* parameter). If the *nettype* parameter is (char *)NULL, the procedure is registered for all transports that are specified in the /etc/netconfig file with a corresponding flag value visible. After registering the local procedure, the server program's main procedure calls **svc_run()**, the RPC library's remote procedure dispatcher.

Parameters

prognum (Input)
: The program number of the remote program.

versnum (Input)
: The version number of the remote program.

procnum (Input)
: The procedure number to be called.

procname (Input)
: The procedure name.

inproc (Input)
: The eXternal Data Representation (XDR) subroutine that decodes the procedure parameters.

outproc (Input)
: The XDR subroutine that encodes the procedure results.

nettype (Input)
: The following classes of transport protocol are valid and are represented as a string either in lowercase or in uppercase: NETPATH, VISIBLE, CIRCUIT_V, DATAGRAM_V, CIRCUIT_N, DATAGRAM_N, TCP, AND UDP. When this parameter is NULL, NETPATH is assumed.

Authorities

The caller of the **rpc_reg()** API must have execute (*X) authority to the /etc directory and must have read (*R) authority to the netconfig file.

Return Value

TRUE (1)	**rpc_reg()** was successful.
FALSE (0)	**rpc_reg()** was not successful. The *errno* variable is set to indicate the reason.

Error Conditions

This API inherits all error conditions from the **setnetconfig()** and **getnetconfig()** APIs. It also inherits all error conditions from the **svc_tli_create()** and **svc_reg()** APIs.

Error Messages

Message ID	**Error Message Text**
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPIA1B2 I	TI-RPC encountered a problem in the transport protocol.
CPIA1B3 I	TI-RPC encountered a problem in the server.
CPIA1B5 I	An incorrect nettype was given.

Related Information

- "svc_reg()—Associate Program and Version with Dispatch" on page 67

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **rpc_reg()** is used:

```
/* Define remote program number and version */
#define RMTPROGNUM (u_long)0x3fffffffL
#define RMTPROGVER (u_long)0x1
#define RMTPROCNUM (u_long)0x1

#include <stdio.h>
#include <rpc/rpc.h>

int *rmtproc(int *param) /* remote procedure */
{
  static int result;
  result = *param + *param;
  return(&result);
}

main()
{
  int *rmtprog();

  /* Register remote program with RPCBind */
  if (rpc_reg(RMTPROGNUM, RMTPROGVER, RMTPROCNUM, rmtprog,
                        xdr_int, xdr_int, "VISIBLE") == -1) {
    fprintf(stderr, "Could not Register\n");
    exit(1);
  }
  svc_run();
  exit(1);
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

Top-level APIs

The top-level APIs allow more customization to both the client and the service while still maintaining an ease of development and use.

The top-level APIs are:

- "clnt_call()—Call a Remote Procedure Associated with the Client" (Call a remote procedure associated with the client) calls the remote procedure that is associated with the client handle pointed to by the clnt parameter.
- "clnt_control()—Change Information about a Client Object" on page 40 (Change information about a client object) is used to change or retrieve information about a client object.
- "clnt_create()—Create a Generic Client Handle" on page 43 (Create a generic client handle) creates and returns a generic client handle for program prognum and version versnum on a remote host where the server is located.
- "clnt_destroy()—Destroy the RPC Client's Handle" on page 45 (Destroy the RPC Client's Handle) destroys the RPC client's handle.
- "svc_create()—Create a Server Handle" on page 47 (Create a server handle) creates server handles for all the transports belonging to the class nettype.
- "svc_destroy()—Destroy an RPC Service Transport Handle" on page 49 (Destroy an RPC service transport handle) destroys an RPC service transport handle.

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

clnt_call()—Call a Remote Procedure Associated with the Client

Syntax

```
#include <rpc/rpc.h>

enum clnt_stat clnt_call(CLIENT *clnt,
                         const u_long procnum,
                         const xdrproc_t inproc,
                         const caddr_t in,
                         const xdrproc_t outproc,
                         caddr_t out,
                         const struct timeval tout);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **clnt_call()** API calls the remote procedure that is associated with the client handle pointed to by the *clnt* parameter.

The caller of the **clnt_call()** API must pass a valid client handle obtained from a successful call to the **clnt_create()** API.

Parameters

clnt (Input)
A pointer to the client handle structure that results from calling a client creation function that uses a Remote Procedure Call (RPC) such as the **clnt_create()** API.

procnum (Input)
The procedure on the host machine.

inproc (Input)
The name of the XDR procedure that encodes the procedure parameters.

in (Input)
The address of the procedure arguments.

outproc (Input)
The name of the XDR procedure that decodes the procedure results.

out (Output)
The address where results are placed.

tout (Input)
The time allowed for the server to respond.

Authorities

None

Return Value

RPC_SUCCESS (0)	Successful
Non-zero value	**clnt_call()** was not successful.

Error Conditions

Upon failure, **clnt_call()** sets a private field in the client handle. This field has a type 'struct rpc_err', and can be accessed by the **clnt_geterr()** function.

The *re_status* field can be set to one of the following values:

[RPC_AUTHERROR]	Authentication error. Server's response did not pass authentication validation.
[RPC_CANTDECODERES]	The *outproc* XDR function has failed.
[RPC_CANTENCODEARGS]	The *inproc* XDR function has failed.

[RPC_CANTRECV]	Failure in receiving result. RPC is unable to receive server's response. The *re_errno* field is set to the value returned from the failed call. *[EBADF]* Bad file descriptor. This value is set when the *client* parameter is not valid or the file descriptor associated with it is already closed or damaged. *[EIO]* Input/output error. This value is set as a result of network transport failure. It indicates that RPC cannot handle an error that occurred in the lower transport levels. *[ENOMEM]* Out of memory. *[EOPNOTSUPP]* Operation is not supported. This value is set when *client* is not valid or the file descriptor associated with it has a limited capabilities. *[EUNKNOWN]* Unknown system state.
[RPC_CANTSEND]	Failure in sending call. RPC is unable to send a request. The *re_errno* field is set to the value returned from the failed call. *[EBADF]* Bad file descriptor. This value is set when the *client* parameter is not valid or the file descriptor associated with it is already closed or damaged. *[EIO]* Input/output error. This value is set as a result of network transport failure. It indicates that RPC cannot handle an error that occurred in the lower transport levels. *[ENOMEM]* Out of memory. *[EOPNOTSUPP]* Operation is not supported. This value is set when *client* is not valid or the file descriptor associated with it has a limited capabilities. *[EUNKNOWN]* Unknown system state.
[RPC_FAILED]	The *tout* parameter is not set properly.
[RPC_INTR]	Interrupted RPC call. An exception has occurred in the RPC API. The *re_errno* field is set to EUNKNOWN.
[RPC_TIMEDOUT]	RPC call is timed out. The client cannot receive a response in the specified timeout period.
[RPC_PROGVERSMISNATCH]	There are no registered versions for the program.
[RPC_PROGNOTREGISTERED]	The program is not registered with the server.
[RPC_PROGUNAVAIL]	The program is not registered with the server.

Error Messages

Message ID	**Error Message Text**
CPIA1B1 I	A problem was encountered in the RPC client.
CPIA1B2 I	TI-RPC encountered a problem in the transport protocol.
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- "rpc_call()—Call a Remote Procedure on the Specified System" on page 32

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **clnt_call()** is used:

```
#include <stdio.h>
#include <rpc/rpc.h>
#include <sys/time.h>

main()
{
  u_long procnum;
  CLIENT *clnt;
  enum clnt_stat cs;
  struct rpc_err client_error;
  struct timeval total_timeout;
  int intsend, intrecv;

  ...

  /* Call the remote procedure that is associated with client */
  cs = clnt_call(clnt, procnum, xdr_int,
                            (caddr_t)&intsend, xdr_int,
                            (caddr_t)&intrecv, total_timeout);

  if (cs != RPC_SUCCESS){
    clnt_geterr(client,&client_error);
    ...
    exit(1);
  }
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

clnt_control()—Change Information about a Client Object

Syntax

```
#include <rpc/rpc.h>

bool_t clnt_control(CLIENT *clnt,
                    const u_int req,
                    char *info);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **clnt_control()** function is used to change or retrieve information about a client object. For both connectionless and connection-oriented transports, the supported values for *req*, their argument types, and what they do follow:

Values for the req Parameter	Argument Type	Function
CLSET_TIMEOUT	(struct timeval *)	Set total time out
CLGET_TIMEOUT	(struct timeval *)	Get total time out
CLGET_SERVER_ADDR	(struct netbuf *)	Get server's address
CLGET_SVC_ADDR	(struct netbuf *)	Get server's address
CLSET_SVC_ADDR	(struct netbuf *)	Set to new address
CLGET_FD	(int *)	Get the associated file descriptor
CLSET_FD_CLOSE	(void)	Close the file descriptor when the API destroys the client handle
CLSET_FD_NCLOSE	(void)	Do not close the file descriptor when the API destroys the client handle
CLGET_VERS	(unsigned long *)	Get the RPC program's version number that is associated with the client handle
CLSET_VERS	(unsigned long *)	Set the RPC program's version number that is associated with the client handle
CLGET_PROG	(unsigned long *)	Get the program number
CLSET_PROG	(unsigned long *)	Set the program number
CLGET_XID	(unsigned long *)	Get the XID of the previous RPC
CLSET_XID	(unsigned long *)	Set the XID of the next RPC
CLSET_RETRY_TIMEOUT[1]	(struct timeval *)	Set the retry time-out
CLGET_RETRY_TIMEOUT[1]	(struct timeval *)	Get the retry time-out
Note: *1* Valid only for connectionless transports.		

Parameters

clnt (Input)
A pointer to the client handle structure.

req (Input)
The type of operation.

info (Input/Output)
A pointer to the information for request type. The *info* parameter is expected to be a pointer to an appropriate structure. The nature of the structure depends on the *req* parameter.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

Failure is returned only when a bad format of parameters is detected. For example, the *info* parameter is NULL, when a pointer to a timeval structure is expected.

Error Messages

Message ID	Error Message Text
CPIA1B1 I	A problem was encountered in the RPC client.
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **clnt_control()** is used:

```
#include <rpc/rpc.h>

main()
{
  CLIENT *clnt;
  int fd;

  ...

  /* Get the associated file descriptor */
  clnt_control(clnt, CLGET_FD, (int *)&fd);
  ...

}
```

Warning: Temporary Level 4 Header

Notes:

1. If the time-out is set using the **clnt_control()** API, the *timeout* parameter passed to the **clnt_call()** API will be ignored in all future calls.
2. The retry time-out is the time that the connectionless RPC client waits for the server to reply before retransmitting the request.

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

clnt_create()—Create a Generic Client Handle

Syntax

```
#include <rpc/rpc.h>

CLIENT *clnt_create(const char *host,
                    const u_long prognum,
                    const u_long versnum,
                    const char *nettype);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **clnt_create()** API creates and returns a generic client handle for program *prognum* and version *versnum* on a remote host where the server is located. This is done using an available transport of the *nettype* class. The **clnt_create()** API tries all the transports of the *nettype* class available from the /etc/netconfig file, and chooses the first successful one. Transports are tried in top-to-bottom order in the netconfig database. A default time-out is set and can be modified using **clnt_control()**.

Parameters

host (Input)
The name of the remote host where the server is located.

prognum (Input)
The program number of the remote program.

vernum (Input)
The version number of the remote program.

nettype (Input)
The following classes of transport protocol are valid and are represented as a string either in lowercase or in uppercase: NETPATH, VISIBLE, CIRCUIT_V, DATAGRAM_V, CIRCUIT_N, DATAGRAM_N, TCP, and UDP. When this parameter is NULL, NETPATH is assumed.

Authorities

The caller of the **clnt_create()** API must have execute (*X) authority to the /etc directory and must have read (*R) authority to the netconfig file.

Return Value

clnt	Upon successful completion, this API returns a client handle.
NULL	**clnt_create()** was not successful. The *rpc_createerr* variable is set to indicate the reason.

Error Conditions

Upon failure, **clnt_create()** sets the global structure *rpc_createerr*. The *rpc_createerr.cf_stat* variable contains a status value that indicates the error reason. The *rpc_createerr.cf_error.re_errno* variable is meaningful when some status values are set.

The *rpc_createerr.cf_stat* variable can be set to one of the following values:

[RPC_INTR] Interrupted RPC call. An exception has occurred in the RPC API. The *rpc_createerr.cf_error.re_errno* variable is set to EUNKNOWN.

[RPC_N2AXLATEFAILURE] Name-to-address translation failed. Cannot resolve the hostname given in *host*.

[RPC_SYSTEMERROR] An RPC error was returned from the system call. The *rpc_createerr.cf_error.re_errno* variable is set to the value returned from the failed call.

[EACCES]
Permission denied.

[EADDRINUSE]
Local address is in use. This value is set when *host* is not valid or the file descriptor associated with it cannot be bound to any local address.

[EADDRNOTAVAIL]
Address not available. This value is set when the address obtained by the **rpcb_getaddr()** is rejected by the transport layer.

[EAGAIN]
Operation would have caused the process to be blocked.

[EBADF]
Bad file descriptor. This value is set when *host* is not valid or the file descriptor associated with it is already closed or damaged.

[ECONNREFUSED]
TI-RPC encountered a problem in the transport. The client cannot connect to the server.

[EFAULT]
The address created by the **rpcb_getaddr()** was not available.

[EIO] Input/output error. This value is set as a result of network transport failure. It indicates that RPC cannot handle an error that occurred in lower transport levels.

[ENOBUFS]
There is not enough buffer space available for the API.

[ENOMEM]
Out of memory.

[EOPNOTSUPP]
Operation is not supported. This value is set when *host* is not valid or the file descriptor associated with it has limited capabilities.

[EUNKNOWN]
Unknown system state.

[RPC_UNKNOWNHOST] Unknown *host*.

[RPC_UNKNOWNPROTO] Unknown client/server protocol. The *rpc_createerr.cf_error.re_errno* is set with the errno value returned by **setnetconfig()** or **getnetconfig()** call. This error is set when the *netconf* pointer is NULL.

Error Messages

Message ID	Error Message Text
CPIA1B1 I	A problem was encountered in the RPC client.
CPIA1B2 I	TI-RPC encountered a problem in the transport protocol.
CPIA1B5 I	An incorrect nettype was given.
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- "clnt_tp_create()—Create a Client Handle" on page 50
- "clnt_tli_create()—Create a Client Handle" on page 57

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **clnt_create()** is used:

```
/* Define remote program number and version */
#define RMTPROGNUM (u_long)0x3fffffffL
#define RMTPROGVER (u_long)0x1

#include <stdio.h>
#include <rpc/rpc.h>

main()
{
  CLIENT *client;

  /* Service request to host RPCSERVER_HOST */
  client = clnt_create("as400.somewhere.ibm.com", RMTPROGNUM,
                                RMTPROGVER, "TCP");

  if (client == (CLIENT *)NULL) {
    fprintf(stderr,"Couldn't create client\n");
    exit(1);
  }
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

clnt_destroy()—Destroy the RPC Client's Handle

Syntax

```
#include <rpc/rpc.h>

void clnt_destroy(CLIENT *clnt);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **clnt_destroy()** API destroys the RPC client's handle. This function deallocates private data structures, including the *clnt* parameter itself. The use of the *clnt* parameter becomes undefined upon calling the **clnt_destroy()** API. If the RPC library opened the associated file descriptor, or was set using **clnt_control()**, the associated file descriptor will be closed.

The caller should call **auth_destroy** (before calling **clnt_destroy**) to destroy the associated AUTH structure.

Parameters

clnt (Input)
A pointer to the client handle structure.

Authorities

No authorization is required.

Return Value

None.

Error Conditions

None.

Error Messages

Message ID	**Error Message Text**
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

"svc_destroy()—Destroy an RPC Service Transport Handle" on page 49

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **clnt_destroy()** is used:

```
#include <rpc/rpc.h>

main()
{
  CLIENT *clnt;

  /* Create client handle */
  clnt = clnt_create(..);

  ...

  /* Destroy the client handle */
  clnt_destroy(clnt);
  exit(0);
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

svc_create()—Create a Server Handle

Syntax

```
#include <rpc/rpc.h>

int svc_create(const void
              (*dispatch)(const svc_req *,
                   const  SVCXPRT *),
              const u_long prognum,
              const u_long versnum,
              const char *nettype);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **svc_create()** function creates server handles for all the transports belonging to the class *nettype*.

svc_create() tries all the transports of the *nettype* class that are available from the /etc/netconfig file in top-to-bottom order. **svc_create()** registers itself with the RPCBind service.

Parameters

dispatch (Input)
The server dispatch function. *dispatch* is called when there is a remote procedure call for the given *prognum* and *versnum*.

prognum (Input)
The program number of the remote program.

vernum (Input)
The version number of the remote program

nettype (Input)
The following classes of transport protocol are valid: NETPATH, VISIBLE, CIRCUIT_V, DATAGRAM_V, CIRCUIT_N, DATAGRAM_N, TCP, and UDP.

Authorities

The caller of the **svc_create()** API must have execute (*X) authority to the /etc directory and must have read (*R) authority to the netconfig file.

Return Value

num	Upon successful completion, **svc_create()** returns the number of server handles it creates.
0	**svc_create()** was not successful. The *errno* variable is set to indicate the reason.

Error Conditions

This API calls **setnetconfig()** and **getnetconfig()** APIs in order to perform its task. The API inherits all error conditions from those APIs. It also inherits all error conditions from **svc_tp_create()** API except EINVAL.

Error Messages

Message ID	Error Message Text
CPIA1B1 I	A problem was encountered in the RPC client.
CPIA1B2 I	TI-RPC encountered a problem in the transport protocol.
CPIA1B3 I	TI-RPC encountered a problem in the server.
CPIA1B5 I	An incorrect nettype was given.
CPIA1B8 I	A problem occurred while trying to contact the RPCBind daemon.
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- "svc_tp_create()—Create a Server Handle" on page 54
- "svc_tli_create()—Create a Server Handle" on page 69

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **svc_create()** is used:

```
/* Define remote program number and version */
#define RMTPROGNUM (u_long)0x3fffffffL
#define RMTPROGVER (u_long)0x1

#include <stdio.h>
#include <rpc/rpc.h>

static void exm_proc();

main()
{
  int transpnum;

  ...
  transpnum = svc_create(exm_proc, RMTPROGNUM, RMTPROGVER,
                                  "VISIBLE");
  if (transpnum == 0){
    fprintf(stderr, "Cannot create a service.\n");
    exit(1);
  }
  svc_run();        /* No return */
}

/* The server dispatch function */
static void exm_proc(struct svc_req *rqstp, SVCXPRT *transp)
{

  ...

}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

svc_destroy()—Destroy an RPC Service Transport Handle

Syntax

```
#include <rpc/rpc.h>

void svc_destroy(SVCXPRT *xprt);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **svc_destroy()** function destroys an RPC service transport handle. This function deallocates the private data structures, including the handle itself. After the **svc_destroy()** API is used, the handle pointed to by the *xprt* parameter is no longer defined.

Parameters

xprt (Input)
A pointer to the RPC service transport handle.

Authorities

No authorization is required.

Return Value

None.

Error Conditions

None.

Error Messages

None.

Related Information

- "clnt_destroy()—Destroy the RPC Client's Handle" on page 45
- "svc_create()—Create a Server Handle" on page 47

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **svc_destroy()** is used:

```
#include <rpc/rpc.h>

main()
{
  SVCXRPT *transp;

  ...
  /* Destroy the service handle */
  svc_destroy(transp);
  ...
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

Intermediate-level APIs

The intermediate-level APIs are similar to the top-level APIs, but the user applications select the transport-specific information by using network selection APIs. These APIs allow more customization and greater control over the transport that is used.

The intermediate-level APIs are:

- "clnt_tp_create()—Create a Client Handle" (Create a client handle) creates a client handle for the program prognum, the version versnum, and for the transport specified by netconf.
- "svc_tp_create()—Create a Server Handle" on page 54 (Create a server handle) creates a server handle for the network specified by netconf, and registers itself with the RPC service package (RPCBind).

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

clnt_tp_create()—Create a Client Handle

Syntax

```
#include <rpc/rpc.h>
#include <netconfig.h>

CLIENT *clnt_tp_create(const char *host,
                       const u_long prognum,
                       const u_long versnum,
                       const struct netconfig
                           *netconf);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **clnt_tp_create()** API creates a client handle for the program *prognum*, the version *versnum*, and for the transport specified by *netconf*. The remote **RPCBind** service on the host machine *host* is consulted for the address of the remote service.

Parameters

host (Input)
: The name of the remote host where the server is located.

prognum (Input)
: The program number of the remote program.

vernum (Input)
The version number of the remote program.

netconf (Input)
The transport protocol to use.

Authorities

The caller of the **clnt_tp_create()** API must have execute (*X) authority to the /etc directory and must have read (*R) authority to the netconfig file.

Return Value

clnt	Upon successful completion, this function returns a client handle.
NULL	**clnt_tp_create()** was not successful. The *rpc_createerr* variable is set to indicate the reason.

Error Conditions

Upon failure, **clnt_tp_create()** sets the global structure *rpc_createerr*. The *rpc_createerr.cf_stat* variable contains a status value that indicates the error reason. The *rpc_createerr.cf_error.re_errno* variable is meaningful when some status values are set.

The *rpc_createerr.cf_stat* variable can be set to one of the following values:

[RPC_INTR]	Interrupted RPC call. An exception has occurred in the RPC API. The *rpc_createerr.cf_error.re_errno* is set to EUNKNOWN.
[RPC_N2AXLATEFAILURE]	Name-to-address translation failed. The API cannot resolve the hostname given in *host*.
[RPC_PROGNOTREGISTERED]	Remote program is not registered.
[RPC_RPCBFAILURE]	A failure occurred in the RPCBind daemon.

[RPC_SYSTEMERROR] RPC error returned from system call. The *rpc_createerr.cf_error.re_errno* variable is set to the value returned from the failed call.

[EACCES]
Permission denied.

[EADDRINUSE]
Local address is in use. This value is set when the transport endpoint cannot be bound to any local address. This API calls **rpcb_getaddr()** API in order to perform the API's task. It inherits all error conditions from **clnt_tli_create()** and **rpcb_getaddr()** APIs, except RPC_FAILED.

[EADDRNOTAVAIL]
Address not available. This value is set when the address obtained by the **rpcb_getaddr()** is rejected by transport layer.

[EAGAIN]
Operation would have caused the process to be blocked.

[EBADF]
Bad file descriptor. This value is set when the transport endpoint created is not valid.

[EFAULT]
The address created by the **rpcb_getaddr()** was not available.

[EIO] Input/output error. This value is set as a result of network transport failure. It indicates that RPC cannot handle an error that occurred in lower transport levels.

[ENOBUFS]
There is not enough buffer space available for the API.

[ENOMEM]
Out of memory.

[EOPNOTSUPP]
Operation is not supported. This value is set when the transport endpoint was opened with limited capabilities.

[EUNKNOWN]
Unknown system state.

[RPC_UNKNOWNHOST]
Unknown *host*.

[RPC_UNKNOWNPROTO]
Unknown client/server protocol. The *rpc_createerr.cf_error.re_errno* variable is not applicable. This error is set when the *netconf* pointer is NULL.

Error Messages

Message ID	**Error Message Text**
CPIA1B1 I	A problem was encountered in the RPC client.
CPIA1B2 I	TI-RPC encountered a problem in the transport protocol.
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- "clnt_create()—Create a Generic Client Handle" on page 43
- "clnt_tli_create()—Create a Client Handle" on page 57

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **clnt_tp_create()** is used:

```
/* Define remote program number and version */
#define RMTPROGNUM ((u_long)0x3fffffff)
#define RMTPROGVER ((u_long)0x1)

#include <stdio.h>
#include <rpc/rpc.h>
#include <netconfig.h>
#include <netdir.h>

main()
{
  CLIENT *client;
  struct netconfig *nconf;

  /* Returns a pointer to nconf corresponding to NETCONF */
  if ((nconf = getnetconfigent("UDP")) ==
             (struct netconfig *)NULL) {
    fprintf(stderr, "Cannot get netconfig entry for UDP\n");
    exit(1);
  }

  client = clnt_tp_create("as400.somewhere.ibm.com", RMTPROGNUM,
                                    RMTPROGVER, nconf);
  if (client == (CLIENT *)NULL) {
    fprintf(stderr, "Cannot create an RPC client\n");
    exit(1);
  }

  fprintf(stderr, "Successfully created a client handle\n");

  clnt_destroy(client);
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

svc_tp_create()—Create a Server Handle

Syntax

```
#include <rpc/rpc.h>

SVCXPRT svc_tp_create(const void
                (*dispatch)(const svc_req *,
                     const SVCXPRT *),
                 const u_long prognum,
                 const u_long versnum,
                 const struct netconfig *netconf);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **svc_tp_create()** function creates a server handle for the network specified by *netconf*, and registers itself with the RPC service package (RPCBind).

Parameters

dispatch() (Input)
The server dispatch function. *dispatch()* is called when there is a remote procedure call for the given *prognum* and *versnum*. The call to *dispatch* requires calling **svc_run()** on the server side.

prognum (Input)
The program number of the remote program.

vernum (Input)
The version number of the remote program.

netconf (Input)
The transport protocol to use.

Authorities

No authorization is needed.

Return Value

xprt	Upon successful completion, this function returns the service handle.
NULL	**svc_tp_create()** was not successful. The *errno* variable is set to indicate the reason.

Error Conditions

This API calls **svc_tli_create()** and **svc_reg()** functions in order to perform its task. It inherits all error conditions from those functions, except **setnetconfig()** and **getnetconfig()** errors and RPC_UNKNOWNADDR from **svc_reg()**.

Error Messages

Message ID	Error Message Text
CPIA1B1 I	A problem was encountered in the RPC client.
CPIA1B2 I	TI-RPC encountered a problem in the transport protocol.
CPIA1B3 I	TI-RPC encountered a problem in the server.
CPIA1B8 I	A problem occurred while trying to contact the RPCBind daemon.
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- "svc_create()—Create a Server Handle" on page 47
- "svc_tli_create()—Create a Server Handle" on page 69

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **svc_tp_create()** is used:

```
/* Define remote program number and version */
#define RMTPROGNUM (u_long)0x3fffffffL
#define RMTPROGVER (u_long)0x1

#include <stdio.h>
#include <rpc/rpc.h>
#include <netconfig.h>

static void exm_proc();
/* Dispatcher routine, defined later in program */

main()
{
  SVCXPRT *transp;
  struct netconfig *nconf;

  /* Returns a pointer to nconf corresponding to UDP */
  if ((nconf = getnetconfigent("UDP")) ==
             (struct netconfig *)NULL) {
    fprintf(stderr, "Cannot get netconfig entry for UDP\n");
    exit(1);
  }

  transp = svc_tp_create(exm_proc, RMTPROGNUM, RMTPROGVER,
                                   nconf);
  if (transp == (SVCXPRT *)NULL) {
    fprintf(stderr, "Cannot create service.\n");
    exit(1);
  }

  ...
  svc_run();
}

/* The server dispatch function */
static void exm_proc(struct svc_req *rqstp, SVCXPRT *transp)
{
   ...
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

Expert-level APIs

The expert-level APIs are the lowest layer of TI-RPC APIs available on the server. The application directly chooses the transport to use, and has an increased level of control over the details of the client-side and the server-side transport handles. These APIs are similar to the intermediate-level APIs with an additional control provided by using the name-to-address translation APIs.

The expert-level APIs are:

- "clnt_tli_create()—Create a Client Handle" on page 57 (Create a client handle) creates an RPC client handle for the remote program prognum and version versnum.
- "rpcb_getaddr()—Find the Universal Address of a Service" on page 60 (Find the universal address of a service) is an interface to the RPC service package (RPCBind).
- "rpcb_set()—Register the Server Address with the RPCBind" on page 62 (Register the server address with the RPCBind) is an interface to the RPC service package (RPCBind) daemon.
- "rpcb_unset()—Unregister Their Addresses" on page 65 (Unregister Their Addresses) is an interface to the RPC service package (RPCBind), which destroys the mapping between the triple (prognum, versnum, netconf->nc_netid) and the address on the host machine's RPCBind service.
- "svc_reg()—Associate Program and Version with Dispatch" on page 67 (Associate program and version with dispatch) associates prognum and versnum with the service dispatch procedure dispatch.
- "svc_tli_create()—Create a Server Handle" on page 69 (Create a server handle) creates an RPC server handle.
- "svc_unreg()—Delete an Association Set by svc_reg()" on page 71 (Delete an association set by svc_reg()) removes mappings between dispatch functions and the service procedure that is identified by the prognum and versnum parameters.

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

clnt_tli_create()—Create a Client Handle

Syntax

```
#include <rpc/rpc.h>
#include <netconfig.h>

CLIENT *clnt_tli_create(const int fildes,
                        const struct netconfig
                           *netconf,
                        const struct netbuf *svcaddr,
                        const u_long prognum,
                        const u_long versnum,
                        const u_int sendsz,
                        const u_int recvsz);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **clnt_tli_create()** API creates an RPC client handle for the remote program *prognum* and version *versnum*. The remote program is located at address *svcaddr*. The client uses the transport that is specified by *netconf*. Depending upon the type of the transport (connection-oriented or connectionless), **clnt_tli_create()** calls the appropriate client-creation functions.

Parameters

fildes (Input)
A file descriptor. The only permitted value is RPC_ANYFD. The API opens an internal file descriptor which is not accessible by the user applications.

netconf (Input)
The transport protocol.

svcaddr (Input)
A pointer to the address where the remote program is located.

prognum (Input)
The program number of the remote program.

vernum (Input)
The version number of the remote program.

sendsz (Input)
The size of the send buffer. When a value of zero is specified, a suitable default will be chosen by the system.

recvsz (Input)
The size of the receive buffer. When a value of zero is specified, a suitable default will be chosen by the system.

Authorities

No authorization is required.

Return Value

clnt	Upon successful completion, this function returns a client handle.
NULL	**clnt_tli_create()** was not successful. The *rpc_createerr* variable is set to indicate the reason.

Error Conditions

Upon failure, **clnt_tli_create()** sets the global structure *rpc_createerr*. The *rpc_createerr.cf_stat* variable contains a status value that indicates the error reason. The *rpc_createerr.cf_error.re_errno* variable is meaningful when some status values are set.

The *rpc_createerr.cf_stat* variable can be set to one of the following values:

[RPC_INTR]
Interrupted RPC call. An exception has occurred in the RPC API. The *rpc_createerr.cf_error.re_errno* is set to EUNKNOWN.

[RPC_SYSTEMERROR]
RPC error returned from system call. The *rpc_createerr.cf_error.re_errno* variable is set to the value returned from the failed call.

[EACCES]
Permission denied.

[EADDRINUSE]
Local address is in use. This value is set when *fildes* cannot be bound to any local address.

[EADDRNOTAVAIL]
Address not available. This value is set when *svcaddr* is rejected by the transport layer.

[EAGAIN]
Operation would have caused the process to be blocked.

[EBADF]
Bad file descriptor. This value is set when the *fildes* parameter is *not valid* or cannot be used as a transport endpoint.

[ECONNREFUSED]
TI-RPC encountered a problem in the transport. The client cannot connect to the server.

[EFAULT]
The address used for an *svcaddr* was not available.

[EIO]
Input/output error. This value is set as a result of network transport failure. It indicates that RPC cannot handle an error that occurred in lower transport levels.

[ENOBUFS]
There is not enough buffer space available for the API.

[ENOMEM]
Out of memory.

[EOPNOTSUPP]
Operation is not supported. This value is set when *fildes* represents a transport endpoint with limited capabilities.

[EUNKNOWN]
Unknown system state.

[RPC_UNKNOWNADDR]
Unknown remote address. The *rpc_createerr.cf_error.re_errno* variable is not applicable. This error is set when the *svcaddr* pointer is NULL.

[RPC_UNKNOWNPROTO]
Unknown client/server protocol. The *rpc_createerr.cf_error.re_errno* variable is not applicable. This error is set when the *netconf* pointer is NULL.

Error Messages

Message ID	Error Message Text
CPIA1B1 I	A problem was encountered in the RPC client.
CPIA1B2 I	TI-RPC encountered a problem in the transport protocol.
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- "clnt_create()—Create a Generic Client Handle" on page 43
- "clnt_tp_create()—Create a Client Handle" on page 50

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **clnt_tli_create()** is used:

```
/* Define remote program number and version */
#define RMTPROGNUM ((u_long)0x3fffffff)
#define RMTPROGVER ((u_long)0x1)

#include <stdio.h>
#include <rpc/rpc.h>
#include <netconfig.h>
#include <netdir.h>

main()
{
  CLIENT *client;
  struct netconfig *nconf;
  struct netbuf *service_address;
  struct nd_addrlist *nas;
  struct nd_hostserv hs;

  /* Returns a pointer to nconf corresponding to NETCONF */
  if ((nconf = getnetconfigent("UDP")) ==
              (struct netconfig *)NULL) {
    fprintf(stderr, "Cannot get netconfig entry for UDP\n");
    exit(1);
  }

  hs.h_host = "as400.somewhere.ibm.com";
  hs.h_serv = "RPCBIN";
  if(netdir_getbyname(nconf,&hs,&nas) < 0
  || nas->n_cnt == 0) {
     fprintf(stderr, "Cannot translate host name or service name\n");
  service_address = nas->n_addrs;

  client = clnt_tli_create(RPC_ANYFD, nconf, service_address,
                      RMTPROGNUM, RMTPROGVER, 0, 0);
  if (client == (CLIENT *)NULL) {
    fprintf(stderr, "Cannot create an RPC client\n");
    exit(1);
  }
  fprintf(stderr, "Successfully created a client handle\n");

  clnt_destroy(client);
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

rpcb_getaddr()—Find the Universal Address of a Service

Syntax

```
#include <rpc/rpc.h>
#include <netconfig.h>

bool_t rpcb_getaddr(const u_long prognum,
                    const u_long versnum,
                    const struct netconfig *netconf,
                    struct netbuf *svcaddr,
                    const char *host);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **rpcb_getaddr()** function is an interface to the RPC service package (RPCBind). The function finds the address of the service on the *host* that is registered with program number *prognum* and version *versnum,* and uses the transport protocol that is associated with *netconf*.

Parameters

prognum (Input)
The program number of the remote program.

vernum (Input)
The version number of the remote program.

netconf (Input)
The transport protocol.

svcaddr (Output)
A pointer to the address of the requested service on the remote *host* machine.

host (Input)
The name of the remote host where the server is located.

Authorities

The caller of the **rpcb_getaddr()** API must have execute (*X) authority to the /etc directory and must have read (*R) authority to the netconfig file.

Return Value

TRUE (1)	**rpcb_getaddr()** was successful. The address of the remote service in the *svcaddr* parameter was returned.
FALSE (0)	**rpcb_getaddr()** was unsuccessful.

Error Conditions

Upon failure, **rpcb_getaddr()** sets the global structure *rpc_createerr*. The *rpc_createerr.cf_stat* variable contains a status value, which indicates the error reason. The *rpc_createerr.cf_error.re_errno* variable is meaningful when some status values are set.

The *rpc_createerr.cf_stat* variable can be set to one of the following values:

[RPC_FAILED]	The buffer referenced by the *svcaddr* parameter does not have enough space. re_errno field is set to ENOBUFS.
[RPC_INTR]	Interrupted RPC call. An exception has occurred in the RPC API. The *rpc_createerr.cf_error.re_errno* is set to EUNKNOWN.
[RPC_N2AXLATEFAILURE]	Name-to-address translation failed.
[RPC_PROGNOTREGISTERED]	Remote program is not registered.
[RPC_RPCBFAILURE]	Unable to contact the RPCBind daemon.
[RPC_UNKNOWNADDR]	Unknown address. The *svcaddr* is invalid.
[RPC_UNKNOWNHOST]	Unknown host. The *rpc_createerr.cf_error.re_errno* variable is not applicable.
[RPC_UNKNOWNPROTO]	Unknown client/server protocol. The *rpc_createerr.cf_error.re_errno* is set with errno value returned from the **setnetconfig()** or **getnetconfig()** call.

This API calls **clnt_tli_create()** and **clnt_call()** APIs. It inherits RPC_SYSTEMERROR from **clnt_tli_create()** API and it inherits all error conditions from **clnt_call()** API except RPC_TIMEDOUT, RPC_PROGNOTREGISTERED, RPC_PROGVERSMISMATCH, and RPC_FAILED.

Error Messages

Message ID	Error Message Text
CPIA1B1 I	A problem was encountered in the RPC client.
CPIA1B2 I	TI-RPC encountered a problem in the transport protocol.
CPIA1B8 I	A problem occurred while trying to contact the RPCBind daemon.
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **rpcb_getaddr** is used:

```
/* Define remote program number and version */
#define RMTPROGNUM (u_long)0x3fffffffL
#define RMTPROGVER (u_long)0x1
#define ADDBUFSIZE 100

#include <stdio.h>
#include <rpc/rpc.h>
#include <netconfig.h>
```

```
main()
{
  struct netconfig *nconf;
  struct netbuf *svcaddr;
  char addrbuf[ADDRBUFSIZE];
  ...

  svcaddr.len = 0;
  svcaddr.maxlen = ADDRBUFSIZE;
  svcaddr.buf = addrbuf;

  /* Returns a pointer to nconf corresponding to NETCONF */
  if ((nconf = getnetconfigent("UDP")) ==
             (struct netconfig *)NULL) {
    fprintf(stderr, "Cannot get netconfig entry for UDP\n");
    exit(1);
  }

  ...

  if (!rpcb_getaddr(RMTPROGNUM, RMTPROGVER, nconf,
                             svcaddr, "as400.somewhere.ibm.com")){
    fprintf(stderr, "rpcb_getaddr failed!!\n");
    exit(1);
  }

  ...

}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

rpcb_set()—Register the Server Address with the RPCBind

Syntax

```
#include <rpc/rpc.h>
#include <netconfig.h>

bool_t rpcb_set(const u_long prognum,
                const u_long versnum,
                const struct netconfig *netconf,
                const struct netbuf *svcaddr);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **rpcb_set()** function is an interface to the RPC service package (RPCBind) daemon. The function establishes a mapping between the triple (*prognum, versnum, netconf->nc_netid*) and *svcaddr* on the machine's *RPCBind* service. The value of *netconf->nc_netid* must correspond to a network identifier that is defined by the netconfig database.

Parameters

prognum (Input)
The program number of the remote program.

vernum (Input)
The version number of the remote program.

netconf (Input)
The transport protocol.

svcaddr (Input)
A pointer to the local address of the service.

Authorities

No authorization is required.

Return Value

TRUE (1)	**rpcb_set** was successful.
FALSE (0)	**rpcb_set** was unsuccessful.

Error Conditions

Upon failure, **rpcb_set()** sets the global structure *rpc_createerr*. The *rpc_createerr.cf_stat* variable contains a status value, which indicates the error reason. The *rpc_createerr.cf_error.re_errno* variable is meaningful when some status values are set.

The *rpc_createerr.cf_stat* variable can be set to one of the following values:

[RPC_INTR]	Interrupted RPC call. An exception has occurred in the RPC API. The *rpc_createerr.cf_error.re_errno* is set to EUNKNOWN.
[RPC_N2AXLATEFAILURE]	Name to address translation failed.
[RPC_RPCBFAILURE]	Unable to contact the RPCBind daemon.
[RPC_UNKNOWNADDR]	Unknown address. The *svcaddr* is invalid.
[RPC_UNKNOWNADDR]	Unknown remote address. The *rpc_createerr.cf_error.re_errno* variable is not applicable.
[RPC_UNKNOWNPROTO]	Unknown client/server protocol. The *rpc_createerr.cf_error.re_errno* variable is not applicable.

This API calls **clnt_tli_create()** and **clnt_call()** APIs in order to perform its task. It inherits RPC_SYSTEMERROR from **clnt_tli_create()** API and it inherits all error conditions from **clnt_call()** API except RPC_TIMEDOUT and RPC_FAILED.

Error Messages

Message ID	Error Message Text
CPIA1B1 I	A problem was encountered in the RPC client.
CPIA1B2 I	TI-RPC encountered a problem in the transport protocol.
CPIA1B8 I	A problem occurred while trying to contact the RPCBind daemon.
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- "rpcb_unset()—Unregister Their Addresses" on page 65

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **rpcb_set()** is used:

```
/* Define remote program number and version */
#define RMTPROGNUM (u_long)0x3fffffffL
#define RMTPROGVER (u_long)0x1

#include <stdio.h>
#include <rpc/rpc.h>
#include <netconfig.h>

main()
{
  struct netconfig *nconf;
  struct netbuf *svcaddr;

  ...

  /* Returns a pointer to nconf corresponding to NETCONF */
  if ((nconf = getnetconfigent("UDP")) ==
             (struct netconfig *)NULL) {
    fprintf(stderr, "Cannot get netconfig entry for UDP\n");
    exit(1);
  }

  ...

  /* Register to the RPCBind */
  if (!rpcb_set(RMTPROGNUM, RMTPROGVER, nconf, svcaddr)){
    fprintf(stderr, "rpcb_set failed!!\n");
    exit(1);
  }

  ...

}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

rpcb_unset()—Unregister Their Addresses

Syntax

```
#include <rpc/rpc.h>
#include <netconfig.h>

bool_t rpcb_unset(const u_long prognum,
                  const u_long versnum,
                  const struct netconfig *netconf);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **rpcb_unset()** function is an interface to the RPC service package (RPCBind), which destroys the mapping between the triple (*prognum, versnum, netconf*->*nc_netid*) and the address on the host machine's RPCBind service. If *netconf* is NULL, **rpcb_unset()** destroys all mapping between the above triple and the addresses on the machine's RPCBind service.

Parameters

prognum (Input)
The program number of the remote program.

vernum (Input)
The version number of the remote program.

netconf (Input)
The transport protocol.

Authorities

The caller of the **rpcb_unset()** API must have execute (*X) authority to the /etc directory and must have read (*R) authority to the netconfig file.

Return Value

TRUE (1)	**rpcb_unset** was successful.
FALSE (0)	**rpcb_unset** was unsuccessful.

Error Conditions

Upon failure, **rpcb_unset()** sets the global structure *rpc_createerr*. The *rpc_createerr.cf_stat* variable contains a status value, which indicates the error reason. The *rpc_createerr.cf_error.re_errno* variable is meaningful when some status values are set.

The *rpc_createerr.cf_stat* variable can be set to one of the following values:

[RPC_INTR]	Interrupted RPC call. An exception has occurred in the RPC API. The *rpc_createerr.cf_error.re_errno* is set to EUNKNOWN.
[RPC_RPCBFAILURE]	Unable to contact the RPCBind daemon.

This API calls **clnt_tli_create()** and **clnt_call()** APIs in order to perform its task. It inherits RPC_SYSTEMERROR from **clnt_tli_create()** API and it inherits all error conditions from **clnt_call()** API except RPC_TIMEDOUT and RPC_FAILED.

Error Messages

Message ID	Error Message Text
CPIA1B1 I	A problem was encountered in the RPC client.
CPIA1B2 I	TI-RPC encountered a problem in the transport protocol.
CPIA1B8 I	A problem occurred while trying to contact the RPCBind daemon.
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- "rpcb_set()—Register the Server Address with the RPCBind" on page 62

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **rpcb_unset()** is used:

```
/* Define remote program number and version */
#define RMTPROGNUM (u_long)0x3fffffffL
#define RMTPROGVER (u_long)0x1

#include <stdio.h>
#include <rpc/rpc.h>
#include <netconfig.h>

main()
{
  struct netconfig *nconf;

  ...

  /* Returns a pointer to nconf corresponding to NETCONF */
  if ((nconf = getnetconfigent("UDP")) ==
              (struct netconfig *)NULL) {
    fprintf(stderr, "Cannot get netconfig entry for UDP\n");
    exit(1);
  }
  ...

  /* Destroy the connect with the RPCBind daemon */
  if (!rpcb_unset(RMTPROGNUM, RMTPROGVER, nconf)){
    fprintf(stderr, "rpcb_unset failed!!\n");
    exit(1);
  }

  ...
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

svc_reg()—Associate Program and Version with Dispatch

Syntax

```
#include <rpc/rpc.h>
#include <netconfig.h>

bool_t svc_reg(const SVCXPRT *xprt,
            const u_long prognum,
            const u_long versnum,
            const void (*dispatch)(const svc_req *,
                const SVCXPRT *),
            const struct netconfig *netconf);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **svc_reg()** API associates *prognum* and *versnum* with the service dispatch procedure *dispatch*. If *netconf* is NULL, the service is not registered with the RPC service package (RPCBind). If *netconf* is non-null, then a mapping of the triple (*prognum, versnum, netconf->nc_netid*) to *xprt->xp_ltaddr* is established with the local RPCBind service.

Parameters

xprt (I/O)
A pointer to a Remote Procedure Call (RPC) service transport handle.

prognum (Input)
The program number of the remote program.

versnum (Input)
The version number of the remote program.

dispatch (Input)
The server dispatch function.

netconf (Input)
The transport protocol.

Authorities

The caller of the **svc_reg()** API must have execute (*X) authority to the /etc directory and must have read (*R) authority to the netconfig file.

Return Value

TRUE (1)	**svc_reg()** was successful.
FALSE (0)	**svc_reg()** was not successful. The *errno* variable is set to indicate the reason.

Error Conditions

This API calls the **setnetconfig()** and **getnetconfig()** functions in order to perform its task. The API inherits all error conditions from those functions. It also calls **rpcb_set()** for registering in RPCBind inheriting all error conditions from the API, except RPC_UNKNOWNPROTO.

[EINVAL]	Attempt to register a dispatcher with prognum and versnum, which are already used by another dispatcher.
[EALREADY]	Attempting to register a service which is already registered.

Error Messages

Message ID	**Error Message Text**
CPIA1B1 I	A problem was encountered in the RPC client.
CPIA1B2 I	TI-RPC encountered a problem in the transport protocol.
CPIA1B8 I	A problem occurred while trying to contact the RPCBind daemon.
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPIA1B2 I	TI-RPC encountered a problem with the transport protocol.
CPIA1B8 I	A problem occurred while trying to contact the RPCBind daemon.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **svc_reg()** is used:

```
/* Define remote program number and version */
#define RMTPROGNUM (u_long)0x3fffffffL
#define RMTPROGVER (u_long)0x1

#include <stdio.h>
#include <rpc/rpc.h>
#include <netconfig.h>

static void exm_proc();

main()
{
  SVCXPRT *xprt;
  struct netconfig *nconf;
  int result;

  ...
  /* Returns a pointer to nconf corresponding to NETCONF */
  if ((nconf = getnetconfigent("UDP")) ==
              (struct netconfig *)NULL) {
    fprintf(stderr, "Cannot get netconfig entry for UDP\n");
    exit(1);
  }

  ...

  result = svc_reg(xprt, RMTPROGNUM, RMTPROGVER,
                                exm_proc, nconf);
  if ( !result){
    fprintf(stderr, "svc_reg failed!!\n");
    exit(1);
  }
  ...
}

/* The server dispatch function */
static void exm_proc(struct svc_req *rqstp, SVCXPRT *transp)
{

  ...

}
```

API introduced: V4R2

Top | “Remote Procedure Call (RPC) APIs,” on page 1 | APIs by category

svc_tli_create()—Create a Server Handle

Syntax

```
#include <rpc/rpc.h>
#include <netconfig.h>

SVCXPRT svc_tli_create(const int fildes,
                       const struct netconfig
                           *netconf,
                       const struct t_bind
                           *bindaddr,
                       const u_int sendsz,
                       const u_int recvsz);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **svc_tli_create()** function creates an RPC server handle.

Parameters

fildes (Input)
The file descriptor on which the service is listening. The only permitted value for a user application is RPC_ANYFD. If the file descriptor *fildes* is RPC_ANYFD, it opens a file descriptor on the transport specified by *netconf*.

netconf (Input)
The transport protocol.

bindaddr (Input)
The address where *fildes* is bound if it is unbound.

sendsz (Input)
The size of the send buffer. When a value of zero is specified, a suitable default value will be chosen by the system.

recvsz (Input)
The size of the receive buffer. When a value of zero is specified, a suitable default value will be chosen by the system.

Authorities

No authorization is required.

Return Value

xprt	Upon successful completion, this function returns a pointer to the created RPC server handle.
NULL	**svc_tli_create()** was not successful. The *errno* variable is set to indicate the reason.

Error Conditions

[ENOMEM]	Out of memory.
[EUNKNOWN]	Unknown system state.
[EADDRNOTAVAIL]	Address not available. This value is set when *bindaddr* is rejected by the transport layer.
[EIO]	Input/output error. This value is set as a result of network transport failure. It indicates that RPC cannot handle an error that occurred in lower transport levels.
[EACCES]	Permission denied.
[EBADF]	Bad file descriptor. This value is set when the *fildes* parameter is *not valid* or cannot be used as a transport endpoint.
[EFAULT]	The address used for a *bindaddr* was not available.
[ENOBUFS]	There is not enough buffer space available for the API.
[EINVAL]	An invalid value was supplied for the input parameter *nconf*.
[EADDRINUSE]	Local address is in use. This value is set when *fildes* cannot be bound to any local address.

Error Messages

Message ID	**Error Message Text**
CPIA1B2 I	TI-RPC encountered a problem in the transport protocol.
CPIA1B3 I	TI-RPC encountered a problem in the server.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- "svc_create()—Create a Server Handle" on page 47
- "svc_tp_create()—Create a Server Handle" on page 54

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **svc_tli_create** is used:

```
/* Define remote program number and version */
#define RMTPROGNUM (u_long)0x3fffffffL
#define RMTPROGVER (u_long)0x1
```

```
#include <stdio.h>
#include <rpc/rpc.h>
#include <netconfig.h>

main()
{
  SVCXPRT *svc;
  struct netconfig *nconf;
  int fd;

  ...
  /* Returns a pointer to nconf corresponding to UDP */
  if ((nconf = getnetconfigent("UDP")) ==
               (struct netconfig *)NULL) {
    fprintf(stderr, "Cannot get netconfig entry for UDP\n");
    exit(1);
  }

  ...

  svc = svc_tli_create(RPC_ANYFD,nconf,
                              (struct t_bind *)NULL,
                              0, 0);
  if (svc == (SVCXPRT *)NULL){
    fprintf(stderr, "svc_tli_create failed!!\n");
    exit(1);
  }

  ...

}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

svc_unreg()—Delete an Association Set by svc_reg()

Syntax

```
#include <rpc/rpc.h>

void svc_unreg(const u_long prognum,
               const u_long versnum);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **svc_unreg()** function removes mappings between dispatch functions and the service procedure that is identified by the *prognum* and *versnum* parameters. It also removes the mapping between the port number and the service procedure, which is identified by the *prognum* and *versnum* parameters.

Parameters

prognum (Input)
The program number of the remote program.

vernum (Input)
The version number of the remote program.

Authorities

No authorization is required.

Return Value

None.

Error Conditions

None.

Error Messages

Message ID	Error Message Text
CPIA1B1 I	A problem was encountered in the RPC client.
CPIA1B2 I	TI-RPC encountered a problem in the transport protocol.
CPIA1B8 I	A problem occurred while trying to contact the RPCBind daemon.
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Related Information

- "svc_reg()—Associate Program and Version with Dispatch" on page 67

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **svc_unreg** is used:

```
/* Define remote program number and version */
#define RMTPROGNUM (u_long)0x3fffffffL
#define RMTPROGVER (u_long)0x1

#include <stdio.h>
#include <rpc/rpc.h>
#include <netconfig.h>

static void exm_proc();

main()
{
  SVCXPRT *xprt;
  struct netconfig *nconf;

  ...

  result = svc_reg(xprt, RMTPROGNUM, RMTPROGVER,
                             exm_proc, nconf);
  if ( !result){
    fprintf(stderr, "svc_reg failed!!\n");
    exit(1);
  }

  ...

  /* Removes mapping between procedures and objects */
  svc_unreg(RMTPROGNUM, RMTPROGVER);

}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

Other APIs

These APIs are used primarily in conjunction with all the layers except the simplified-level APIs. These APIs provide methods for sending back errors from the service to the client, for freeing space allocated to the clients and services, and for enhancing error detection and reporting.

The system functions that work with applications from the previous four categories are:

- "clnt_freeres()—Free Data Allocated by the RPC or XDR System" on page 74 (Free data allocated by the RPC or XDR system) frees any data allocated by the RPC or XDR system when it decoded the results of an RPC call.
- "clnt_geterr()—Get the Error Structure from the Client Handle" on page 76 (Get the error structure from the client handle) copies the error structure out of the client handle to the structure at address errp.
- "svcerr_decode()—Send Information to Client for Decode Error" on page 77 (Send information to client for decode error) sends information to the remote client that the service dispatch routine could not decode the remote parameters.
- "svcerr_noproc()—Send Information to Client for Procedure Number Error" on page 78 (Send information to client for procedure number error) sends information to the client that the service dispatch routine did not implement the procedure number that the caller requested.
- "svcerr_systemerr()—Send Information to Client for System Error" on page 79 (Send information to client for system error) sends information to the remote client that the service dispatch routine detected a system error not covered by any particular protocol.
- "svcerr_weakauth()—Send Authentication Error Indication to a Client" on page 82 (Send Authentication Error Indication to a Client) sends information to a remote client that the server dispatch function detected an authentication error.
- "svc_freeargs()—Free Data Allocated by the RPC or XDR System" on page 83 (Free data allocated by the RPC or XDR system) frees any data allocated by the RPC or XDR functions when those functions decode the arguments to a service procedure by using svc_getargs().
- "svc_getargs()—Decode the Arguments of an RPC Request" on page 84 (Decode the arguments of an RPC request) decodes the arguments of an RPC request associated with the RPC service transport handle xprt.
- "svc_getrpccaller()—Get the Network Address of the Caller" on page 85 (Get the network address of the caller) retrieves the network address of the remote client who is calling the procedure that is associated with the RPC service transport handle.
- "svc_run()—Wait for RPC Requests to Arrive" on page 86 (Wait for RPC requests to arrive) waits for RPC requests to arrive and calls the appropriate service procedure.
- "svc_sendreply()—Send the Results of a Procedure Call to a Remote Client" on page 87 (Send the results of a procedure call to a remote client) sends the results of a procedure call to a remote client.

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

clnt_freeres()—Free Data Allocated by the RPC or XDR System

Syntax

```
#include <rpc/rpc.h>

bool_t clnt_freeres(CLIENT *clnt,
                          const xdrproc_t inproc,
                          caddr_t in);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **clnt_freeres()** function frees any data allocated by the RPC or XDR system when it decoded the results of an RPC call.

Parameters

clnt (Input)
A pointer to the client handle.

inproc (Input)
XDR routine describing the results.

in (Input)
(Input) The address of the results.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

This function returns FALSE when the *in* parameter is NULL or an exception has occurred. In case of an exception, **clnt_freeres()** tries to set RPC_INTR in the client handle. This status can be retrieved by a call to **clnt_geterr()**.

Error Messages

Message ID	Error Message Text
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **clnt_freeres()** is used:

```
#include <stdio.h>
#include <rpc/rpc.h>

 ...

u_long procnum;
CLIENT *clnt;
enum clnt_stat stat;
struct rpc_err client_error;
struct timeval timeout;

struct array_args{
   unsigned int size;
   char *data;
};

struct array_args args;    /* Arg with buffer to send */
struct array_args result;  /* Arg with buffer to receive */

 ...

/* Call the remote procedure that is associated with client */

 ...

stat = clnt_call(clnt, procnum, (xdrproc_t)xdr_array,
                          (char *)&args, (xdrproc_t)xdr_array,
                          (char *)&result, timeout);
if (stat != RPC_SUCCESS){
  /* Failure on call */
  if (result.data != (char *) NULL){
    if(!clnt_freeres(clnt, (xdrproc_t)xdr_array,
                              (char *)&result))
      /* clnt_freeres() failed */

      ...

  }
  ...
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

clnt_geterr()—Get the Error Structure from the Client Handle

Syntax

```
#include <rpc/rpc.h>

void clnt_geterr(const CLIENT *clnt,
                         struct rpc_err *errp);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

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

Parameters

clnt (Input)
A pointer to the client handle.

errp (Output)
A pointer to the error structure.

Authorities

No authorization is required.

Return Value

None.

Error Conditions

When an exception occurs, **clnt_geterr()** tries to set RPC_INTR in the client handle. This status can be retrieved by another valid **clnt_geterr()** call. If the attempt was unsuccessful, no error indication is given.

Error Messages

Message ID	Error Message Text
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **clnt_geterr()** is used:

```
#include <stdio.h>
#include <rpc/rpc.h>
#include <sys/time.h>

main()
{
  u_long procnum;
  CLIENT *clnt;
  enum clnt_stat cs;
  struct rpc_err client_error;
  struct timeval total_timeout;
  int intsend, intrecv;

  ...

  /* Call the remote procedure that is associated with client */
  cs = clnt_call(clnt, procnum, xdr_int,
                         (caddr_t)&intsend, xdr_int,
                         (caddr_t)&intrecv, total_timeout);

  if (cs != RPC_SUCCESS){
    clnt_geterr(clnt,&client_error);
    ...
    exit(1);
  }
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

svcerr_decode()—Send Information to Client for Decode Error

Syntax

```
#include <rpc/rpc.h>

void svcerr_decode(const SVCXPRT *xprt);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **svcerr_decode()** function sends information to the remote client that the service dispatch routine could not decode the remote parameters.

Parameters

xprt (Input)
A pointer to the RPC service transport handle.

Authorities

No authorization is required.

Return Value

None.

Error Conditions

In case of an exception, the *errno* global variable is set to EUNKNOWN.

Error Messages

Message ID	Error Message Text
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

Refer to the example for "svcerr_systemerr()—Send Information to Client for System Error" on page 79

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

svcerr_noproc()—Send Information to Client for Procedure Number Error

Syntax

```
#include <rpc/rpc.h>

void svcerr_noproc(const SVCXPRT *xprt);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **svcerr_noproc()** function sends information to the client that the service dispatch routine did not implement the procedure number that the caller requested.

Parameters

xprt (Input)
A pointer to the RPC service transport handle.

Authorities

No authorization is required.

Return Value

None.

Error Conditions

In case of an exception, the *errno* global variable is set to EUNKNOWN.

Error Messages

Message ID	Error Message Text
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

Refer to the example for "svcerr_systemerr()—Send Information to Client for System Error"

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

svcerr_systemerr()—Send Information to Client for System Error

Syntax

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

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **svcerr_systemerr()** function sends information to the remote client that the service dispatch routine detected a system error not covered by any particular protocol.

Parameters

xprt (Input)
A pointer to the RPC service transport handle.

Authorities

No authorization is required.

Return Value

None.

Error Conditions

In case of an exception, the *errno* global variable is set to EUNKNOWN.

Error Messages

Message ID	Error Message Text
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **svcerr_systemerr()** is used:

```
#include <stdio.h>
#include <stdlib.h> /* getenv, exit */
#include <rpc/rpc.h>

#define MESSAGEPROG ((unsigned long)(0x20000001))
#define PRINTMESSAGEVERS ((unsigned long)(1))
#define PRINTMESSAGE ((unsigned long)(1))

/* This procedure is called by dispatcher routine */
int *printmessage_l(char **msg, struct svc_req *req)
{
  static int result;
  char stffl30";
  int fd;

  /* Do something with *msg contents */
  ...

  result = 1;
  return(&result);
}

/* This is the server dispatcher routine.
   It is called when a request arrives from client
   and it applies to MESSAGEPROG program number and PRINTMESSAGEVERS
   version number */

static void
messageprog_l(struct svc_req *rqstp, SVCXPRT *transp)
{
  union u_argument{
   char *printmessage_l_arg;
  }argument;
  char *result;
  bool_t (*_xdr_argument)(), (*_xdr_result)();
  char *(*local)(union u_argument *, struct svc_req *);

  _rpcsvccount++;
  switch(rqstp->rq_proc)
  {
    /* rqstp->rq_proc contains the procedure number
       of procedure that should be called */

    case NULLPROC: /* empty procedure, do nothing, just send the ack */
      svc_sendreply(transp, (xdrproc_t)xdr_void, (char *)NULL);
      return;
    case PRINTMESSAGE: /* printmessage_l() */
      if (rqstp->rq_cred.oa_flavor != AUTH_SYS) {
         /* AUTH_SYS is required by this procedure */
         svcerr_weakauth(transp);
         return;
       }
      _xdr_argument = (bool_t(*)())xdr_wrapstring;
      _xdr_result = (bool_t(*)())xdr_int;
      local = (char *(*)(u_argument *, struct svc_req *))
                         printmessage_l;
      break;
    default: /* no other procedures available */
```

```
      svcerr_noproc(transp);
      return;
  }
  memset((char *)&argument, 0, sizeof(argument));

  /* decode arguments for the procedure */
  if (!svc_getargs(transp, (xdrproc_t)_xdr_argument,
                   (char *)&argument)){
    svcerr_decode(transp);
    return;
  }

  /* Invoke the procedure */
  result = (*local)(&argument, rqstp);

  /* Send reply to the client containing results of the invocation */
  if (result != NULL && !svc_sendreply(transp,
                            (xdrproc_t)_xdr_result, result)){
    svcerr_systemerr(transp);
  }

  if (!svc_freeargs(transp, (xdrproc_t)_xdr_argument,
                            (char *)&argument)){
    printf("unable to free arguments");
    exit(1);
  }
  return;
}

main()
{
  pid_t pid;
  int i;

  printf("Start..");

  printf("Try to create..");
  /* Create a new RPC server instance which will use messageprog_1()
     as a dispatcher function associated with MESSAGEPROG program
     number and PRINTMESSAGEVERS version number.
     Since "VISIBLE" nettype is selected, a number of server instances
     will be actually created: one for each "VISIBLE" entry in
     /etc/netconfig */
  if(!svc_create(messageprog_1, MESSAGEPROG, PRINTMESSAGEVERS,
                 "VISIBLE")){
     printf("Unable to create service.");
     return 1;
  }

  /* Enter the main loop of RPC */
  svc_run();

  return 0;
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

svcerr_weakauth()—Send Authentication Error Indication to a Client

Syntax

```
#include <rpc/rpc.h>

void svcerr_weakauth(const SVCXPRT *xprt);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **svcerr_weakauth()** function sends information to a remote client that the server dispatch function detected an authentication error.

Parameters

xprt (Input)
A pointer to the RPC service transport handle.

Authorities

No authorization is required.

Return Value

None.

Error Conditions

In case of an exception, the *errno* global variable is set to EUNKNOWN.

Error Messages

Message ID	Error Message Text
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

Refer to the example for "svcerr_systemerr()—Send Information to Client for System Error" on page 79

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

svc_freeargs()—Free Data Allocated by the RPC or XDR System

Syntax

```
#include <rpc/rpc.h>

bool_t svc_freeargs(const SVCXPRT *xprt,
                          const xdrproc_t inproc,
                          caddr_t in);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **svc_freeargs()** function frees any data allocated by the RPC or XDR functions when those functions decode the arguments to a service procedure by using **svc_getargs()**.

Parameters

xprt (Input)
A pointer to the RPC service transport handle.

inproc (Input)
The XDR routine to free the arguments.

in (Input)
The address of the arguments.

Authorities

No authorization is required.

Return Value

TRUE (1)	**svc_freeargs** was successful.
FALSE (0)	**svc_freeargs** was unsuccessful.

Error Conditions

svc_freeargs() returns FALSE only when the in parameter is NULL or an exception has occurred. In case of the exception, the *errno* global variable is set to EUNKNOWN.

Error Messages

Message ID	**Error Message Text**
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library amp;2 ended. Reason code &3.

Example

Refer to the example for "svcerr_systemerr()—Send Information to Client for System Error" on page 79.

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

svc_getargs()—Decode the Arguments of an RPC Request

Syntax

```
#include <rpc/rpc.h>

bool_t svc_getargs(const SVCXPRT *xprt,
                         const xdrproc_t inproc,
                         caddr_t in);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **svc_getargs()** function decodes the arguments of an RPC request associated with the RPC service transport handle *xprt*.

Parameters

xprt (Input)
A pointer to the RPC service transport handle.

inproc (Input)
The XDR routine to decode the arguments.

in (Input)
The address of the arguments.

Authorities

No authorization is required.

Return Value

TRUE (1)	**svc_getargs** was successful.
FALSE (0)	**svc_getargs** was unsuccessful.

Error Conditions

svc_getargs() returns FALSE only when the in parameter is NULL or an exception has occurred. In case of the exception, the *errno* global variable is set to EUNKNOWN.

Error Messages

Message ID	Error Message Text
CPIA1B3 I	TI-RPC encountered a problem in the server.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

Refer to the example for “svcerr_systemerr()—Send Information to Client for System Error” on page 79

API introduced: V4R2

Top | “Remote Procedure Call (RPC) APIs,” on page 1 | APIs by category

svc_getrpccaller()—Get the Network Address of the Caller

Syntax

```
#include <rpc/rpc.h>

struct netbuf *svc_getrpccaller(SVCXPRT *xprt);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **svc_getrpccaller()** function macro retrieves the network address of the remote client who is calling the procedure that is associated with the RPC service transport handle.

Parameters

xprt (Input)
A pointer to the RPC service transport handle.

Authorities

No authorization is required.

Return Value

netbuf	Returns a pointer to a netbuf structure containing the address of the caller of a procedure that is associated with the RPC service *xprt*.

Error Conditions

None.

Error Messages

None.

Example

The following example shows how **svc_getrpccaller()** is used :

```
#include <rpc/rpc.h>

main()
{
  SVCXPRT *svc;
  struct netbuf *net_buf;

  ...
  /* Get the caller address */
  net_buf = svc_getrpccaller(svc);

  ...
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

svc_run()—Wait for RPC Requests to Arrive

Syntax

```
#include <rpc/rpc.h>

void svc_run(void);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **svc_run()** function waits for RPC requests to arrive and calls the appropriate service procedure.

Parameters

None.

Authorities

No authorization is required.

Return Value

None.

Error Conditions

The **svc_run()** function rarely exits. It exits only when an exception has occurred. In this case the *errno* global variable is set to EUNKNOWN.

Error Messages

Message ID	Error Message Text
CPIA1B2 I	TI-RPC encountered a problem in the transport protocol.
CPIA1B3 I	TI-RPC encountered a problem in the server.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

Refer to the example for "svcerr_systemerr()—Send Information to Client for System Error" on page 79.

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

svc_sendreply()—Send the Results of a Procedure Call to a Remote Client

Syntax

```
#include <rpc/rpc.h>

bool_t svc_sendreply(const SVCXPRT *xprt,
                     const xdrproc_t inproc,
                     const caddr_t in);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **svc_sendreply()** function sends the results of a procedure call to a remote client.

Parameters

xprt (Input)
A pointer to the RPC service transport handle.

inproc (Input)
XDR routine to encode the results.

in (Input)
The address of the results.

Authorities

No authorization is required.

Return Value

TRUE (1)	**svc_sendreply()** was successful.
FALSE (0)	**svc_sendreply()** was unsuccessful.

Error Conditions

The **svc_sendreply()** function returns FALSE when some transport error or some exception has occurred. The *errno* global variable can be set to the following values:

[EBADF]	Bad file descriptor.
[EINVAL]	General I/O error.
[EOPNOTSUPP]	Operation is not supported.
[EUNKNOWN]	Unknown system state or exception has occurred.

Error Messages

Message ID	**Error Message Text**
CPIA1B3 I	TI-RPC encountered a problem in the server.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

Refer to the example for “svcerr_systemerr()—Send Information to Client for System Error” on page 79

API introduced: V4R2

Top | “Remote Procedure Call (RPC) APIs,” on page 1 | APIs by category

External Data Representation APIs

The eXternal Data Representation (XDR) functions define a uniform way to represent data types and define a language that can describe data structures of arbitrary complexity in a standard way.

All XDR APIs can translate data in two directions:

Serializing	Translation from a local machine data representation to canonical XDR form.
Deserializing	Translation from canonical XDR form to a local machine representation.

The eXternal Data Representation APIs are:

- “xdr_array()—Translate between Arrays and Their XDR” on page 90 (Translate between arrays and their XDR) is a filter primitive that translates between variable-length arrays and their corresponding external representations.
- “xdr_bool()—Translate between Booleans and Their XDR” on page 92 (Translate between Booleans and their XDR) is a filter primitive that translates between Booleans (C integers) and their external representations.
- “xdr_bytes()—Translate between Counted Byte Arrays and Their XDR” on page 93 (Translate between counted byte arrays and their XDR) is a filter primitive that translates between counted byte arrays and their external representations.

- “xdr_char()—Translate between Characters and Their XDR” on page 95 (Translate between characters and their XDR) is a filter primitive that translates between C-language characters and their external representation.
- “xdr_double()—Translate between Double-Precision and XDR” on page 97 (Translate between double-precision and XDR) is a filter primitive that translates between C-language double-precision numbers and their external representations.
- “xdr_double_char()—Translate between Two-Byte Characters” on page 98 (Translate between two-byte characters) is a filter primitive that translates between C-language 2-byte characters and their external representation.
- “xdr_enum()—Translate between Enumeration and XDR” on page 100 (Translate between enumeration and XDR) is a filter primitive that translates between C-language enumeration (enum) and its external representation.
- “xdr_float()—Translate between Floats and Their XDR” on page 101 (Translate between floats and their XDR) is a filter primitive that translates between C-language floating-point numbers (normalized single floating-point numbers) and their external representations.
- “xdr_free()—Generic Freeing Function” on page 102 (Generic freeing function) recursively frees the object pointed to by the pointer passed in.
- “xdr_int()—Translate between Integers and Their XDR” on page 104 (Translate between integers and their XDR) is a filter primitive that translates between C-language integers and their external representation.
- “xdr_long()—Translate between Long Integers and Their XDR” on page 105 (Translate between long integers and their XDR) is a filter primitive that translates between C-language long integers and their external representations.
- “xdr_netobj()—Translate between Netobj Structures and Their XDR” on page 106 (Translate between netobj structures and their XDR) is a filter primitive that translates between variable-length opaque data and its external representation.
- “xdr_opaque()—Translate between Fixed-Size Data and Its XDR” on page 108 (Translate between fixed-size data and its XDR) is a filter primitive that translates between fixed-size opaque data and its external representation.
- “xdr_pointer()—Provide Pointer Chasing within Structures” on page 109 (Provide pointer chasing within structures) provides pointer chasing within structures and serializes null pointers.
- “xdr_reference()—Provide Pointer Chasing within Structures” on page 111 (Provide pointer chasing within structures) is a filter primitive that provides pointer chasing within structures.
- “xdr_short()—Translate between Short Integers and Their XDR” on page 112 (Translate between short integers and their XDR) is a filter primitive that translates between C-language short integers and their external representation.
- “xdr_string()—Translate between Strings and Their XDR” on page 114 (Translate between strings and their XDR) is a filter primitive that translates between C-language strings and their corresponding external representations.
- “xdr_union()—Translate between Unions and Their XDR” on page 115 (Translate between unions and their XDR) is a filter primitive that translates between discriminated C unions and their corresponding external representations.
- “xdr_u_char()—Translate between Unsigned Characters and Their XDR” on page 117 (Translate between unsigned characters and their XDR) is a filter primitive that translates between unsigned C-language characters and their external representations.
- “xdr_u_int()—Translate between an Unsigned Integer and Its XDR” on page 119 (Translate between an unsigned integer and its XDR) is a filter primitive that translates between C-language unsigned integers and their external representations.
- “xdr_u_long()—Translate between an Unsigned Long and Its XDR” on page 120 (Translate between an unsigned long and its XDR) is a filter primitive that translates between C-language unsigned long integers and their external representations.

- "xdr_u_short()—Translate between an Unsigned Short and Its XDR" on page 121 (Translate between an unsigned short and its XDR) is a filter primitive that translates between C-language unsigned short integers and their external representations.
- "xdr_vector()—Translate between Arrays and Their XDR" on page 123 (Translate between arrays and their XDR) is a filter primitive that translates between fixed-length arrays and their corresponding external representations.
- "xdr_void()—Supply an XDR Function to the RPC System" on page 124 (Supply an XDR function to the RPC system) has no parameters.
- "xdr_wrapstring()—Call the xdr_string() Function" on page 126 (Call the xdr_string() function) is a primitive that calls the xdr_string(xdr, sp, maxuint) API, where maxuint is the maximum value of an unsigned integer.

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

xdr_array()—Translate between Arrays and Their XDR

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_array(XDR *xdrs,
                 caddr_t *arrp,
                 u_int *sizep,
                 const u_int maxsize,
                 const u_int elsize,
                 const xdrproc_t elproc);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_array()** function is a filter primitive that translates between variable-length arrays and their corresponding external representations. This function is called to encode or decode each element of the array.

Parameters

xdrs (Input)
A pointer to the eXternal Data Representation (XDR) stream handle.

arrp (I/O)
The address of the pointer to the array. If *arrp==NULL and the array is being deserialized, XDR allocates an array of the appropriate size and sets this parameter to point to that array.

sizep (I/O)
The address of the element count of the array. The element count cannot exceed the value for the *maxsize* parameter.

maxsize (Input)
The maximum number of array elements.

elsize (Input)
The byte size of each of the array elements.

elproc (Input)
Translates between the C form of the array elements and their external representations. This parameter is an XDR filter.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	**Error Message Text**
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_array()** is used:

```
#include <stdio.h>
#include <values.h>
#include <xdr.h>

#define ARRAY_SIZE 256

typedef struct xarray
{
        int size;
        int *p_array;
} xarray ;

bool_t xdr_xarray(XDR *xdrs, xarray *p_xarray )
{
     /*
      * Force XDR to allocate memory while decoding
      */
     if((xdrs->x_op==XDR_DECODE)&&
     (p_xarray->p_array!=NULL))
     {
        free(p_xarray->p_array);
        p_xarray->p_array=NULL;
     }
     /*
      *  This code has a  dual job :
      *  A) While decoding, it allocated memory, stores the decoded
      *          xarray in it, and updates size field in xarray
      *          struct.
      *  B) While decoding it stores xarray's size and the data
      *         itself in XDR.
      */
    return xdr_array(
                 xdrs,
                 (char**)(&(p_xarray->p_array)),
                 &(p_xarray->size),
                 MAX_INT,
                 sizeof(int),
                 (xdrproc_t)xdr_int))

}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

xdr_bool()—Translate between Booleans and Their XDR

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_bool(XDR *xdrs,
                bool_t *bp);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_bool()** function is a filter primitive that translates between Booleans (C integers) and their external representations. When encoding data, this filter produces values of either 1 or 0.

Parameters

xdrs (Input)
A pointer to the eXternal Data Representation (XDR) stream handle.

bp (I/O)
The address of the Boolean data.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_bool()** is used:

```
#include <stdio.h>
#include <types.h>
#include <xdr.h>

typedef struct node
{
        bool_t connected;
        bool_t visited;
} node ;

bool xdr_node(XDR *xdrs, node *p_node)
{
        if(!xdr_bool(xdrs,&(p_node->connected)))
                return FALSE;
        return xdr_bool(xdrs,&(p_node->visited));
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

xdr_bytes()—Translate between Counted Byte Arrays and Their XDR

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_bytes(XDR *xdrs,
                 char **sp,
                 u_int *sizep,
                 const u_int maxsize);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_bytes()** function is a filter primitive that translates between counted byte arrays and their external representations. This function treats a subset of generic arrays in which the size of array elements is known to be 1 and the external description of each element is built-in. The length of the byte sequence is explicitly located in an unsigned integer. The byte sequence is not ended by a null character. The external representation of the bytes is the same as their internal representation.

Parameters

xdrs (Input)
A pointer to the eXternal Data Representation (XDR) stream handle.

sp (I/O)
The address of the pointer to the byte array. If *sp==NULL and the stream is being decoded, then XDR allocates the needed amount of memory.

sizep (I/O)
A pointer to the length of the byte area. The value of this parameter cannot exceed the value of the *maxsize* parameter.

maxsize (Input)
The maximum number of bytes allowed when XDR encodes or decodes messages.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	**Error Message Text**
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_bytes()** is used:

```
#include <stdio.h>
#include <values.h>
#include <xdr.h>

#define ARRAY_SIZE 256

typedef struct xarray
{
        int size;
        char *p_array;
} xarray ;

bool_t xdr_xarray(XDR *xdrs, xarray *p_xarray )
{
     /*
      * Force XDR to allocate memory while decoding
      */
     if((xdrs->x_op==XDR_DECODE)&&
     (p_xarray->p_array!=NULL))
     {
        free(p_xarray->p_array);
        p_xarray->p_array=NULL;
     }
     /*
      *  This code has a  dual job :
      *  A) While decoding, it allocated memory, stores the decoded
      *          xarray in it, and updates size field in xarray
      *          struct.
      *  B) While decoding it stores xarray's size and the data
      *          itself in XDR.
      */
    return xdr_bytes(
             xdrs,
             (&(p_xarray->p_array)),
             &(p_xarray->size),
             MAX_INT);
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

xdr_char()—Translate between Characters and Their XDR

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_char(XDR *xdrs,
                char *cp);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_char()** function is a filter primitive that translates between C-language characters and their external representation.

Note: Encoded characters are not packed and occupy 4 bytes each. For strings of characters, consider using the **xdr_string** function.

Parameters

xdrs (Input)
A pointer to the eXternal Data Representation (XDR) stream handle.

cp (I/O)
A pointer to the character.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	**Error Message Text**
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_char()** is used:

```
#include <stdio.h>
#include <xdr.h>

typedef struct grades
{
        char  math; /* Each grade is 'A'..'D' */
        char  literature;
         char  geography;
        char  computers;
} grades ;


bool xdr_grades(XDR *xdrs, grades *p_grades)
{
        if(!xdr_char(xdrs,&(p_grades->math)))
                return FALSE;
        if(!xdr_char(xdrs,&(p_grades->literature)))
                return FALSE;
        if(!xdr_char(xdrs,&(p_grades->geography)))
                return FALSE;
        return xdr_char(xdrs,&(p_grades->computers));
}
```

API introduced: V4R2

Top | “Remote Procedure Call (RPC) APIs,” on page 1 | APIs by category

xdr_double()—Translate between Double-Precision and XDR

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_double(XDR *xdrs,
                  double *dp);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_double()** function is a filter primitive that translates between C-language double-precision numbers and their external representations.

Parameters

xdrs (Input)
A pointer to the eXternal Data Representation (XDR) stream handle.

dp (I/O)
The address of the double-precision number.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_double()** is used:

```
#include <stdio.h>
#include <xdr.h>
```

```
typedef struct vector
{
        double x,y,z;
} vector ;

bool xdr_vector(XDR *xdrs, vector *p_vector)
{
        if(!xdr_double(xdrs,&(p_vector->x)))
                return FALSE;
        if(!xdr_double(xdrs,&(p_vector->y)))
                return FALSE;
        return xdr_double(xdrs,&(p_vector->z));
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

xdr_double_char()—Translate between Two-Byte Characters

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_double_char(XDR *xdrs,
                char_double_t *cp);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_double_char()** function is a filter primitive that translates between C-language 2-byte characters and their external representation.

Note: Encoded characters are not packed and occupy 2 bytes each. For strings of characters, consider using the **xdr_string()** API.

Parameters

xdrs (Input)
A pointer to the eXternal Data Representation (XDR) stream handle.

cp (I/O)
A pointer to the character.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_double_char()** is used:

```
#include <stdio.h>
#include <xdr.h>

typedef struct grades
{
        char_double_t  math; /* Each grade is 'A'..'D' */
        char_double_t  literature;
        char_double_t  geography;
        char_double_t  computers;
} grades ;


bool xdr_grades(XDR *xdrs, grades *p_grades)
{
        if(!xdr_double_char(xdrs,&(p_grades->math)))
                return FALSE;
        if(!xdr_double_char(xdrs,
                                  &(p_grades->literature)))
                return FALSE;
        if(!xdr_double_char(xdrs,&(p_grades->geography)))
                return FALSE;
        return xdr_double_char(xdrs,
                                  &(p_grades->computers));
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

xdr_enum()—Translate between Enumeration and XDR

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_enum(XDR *xdrs,
                enum_t *ep);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_enum** function is a filter primitive that translates between C-language enumeration (enum) and its external representation.

Parameters

xdrs (Input)
A pointer to the eXternal Data Representation (XDR) stream handle.

ep (I/O)
The address of the enumeration data.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	**Error Message Text**
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_enum()** is used:

```
#include <stdio.h>
#include <xdr.h>
```

```
typedef enum fruit_state { green, ripe } fruit_state;
typedef enum fruit_weight { small, sufficient } fruit_weight;

typedef struct fruit
{
        fruit_state state;
        fruit_weight weight;
} fruit;

bool xdr_fruit(XDR *xdrs, fruit *p_fruit)
{
        if(!xdr_enum(xdrs,(enum_t *)&(p_fruit->state)))
                return FALSE;
        return xdr_enum(xdrs,
                                 (enum_t *)&(p_fruit->weight));
}
```

API introduced: V4R2

Top | “Remote Procedure Call (RPC) APIs,” on page 1 | APIs by category

xdr_float()—Translate between Floats and Their XDR

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_float(XDR *xdrs,
                 float *fp);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_float()** function is a filter primitive that translates between C-language floating-point numbers (normalized single floating-point numbers) and their external representations.

Parameters

xdrs (Input)
A pointer to the eXternal Data Representation (XDR) stream handle.

fp (I/O)
The address of the floating-point number.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_float()** is used:

```
#include <stdio.h>
#include <xdr.h>

typedef struct vector
{
        float x,y,z;
} vector ;

bool xdr_vector(XDR *xdrs, vector *p_vector)
{
        if(!xdr_float(xdrs,&(p_vector->x)))
                return FALSE;
        if(!xdr_float(xdrs,&(p_vector->y)))
                return FALSE;
        return xdr_float(xdrs,&(p_vector->z));
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

xdr_free()—Generic Freeing Function

Syntax

```
#include <rpc/rpc.h>

void xdr_free(xdrproc_t proc,
              char *objp);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_free()** function recursively frees the object pointed to by the pointer passed in.

Parameters

proc (Input)
XDR routine for the object being freed.

objp (Input)
A pointer to the object to be freed.

Authorities

No authorization is required.

Return Value

None.

Error Conditions

In case of an exception, the *errno* global variable is set to EUNKNOWN.

Error Messages

Message ID	**Error Message Text**
CPF3CF2 E	Error(s) occurred during running of &1 API.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_free()** is used:

```
#include <rpc/rpc.h>

main()
{
  CLIENT *cl;
  char *outparam;
  int inparam;
  ...
  cl = clnt_create(...);
  ...
  outparam = NULL;
  clnt_call(cl, MYPROC, xdr_int, &inparam,
            xdr_wrapstring, &outparam, timeout);
  ...
  xdr_free(xdr_wrapstring, &outparam);
  ...
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

xdr_int()—Translate between Integers and Their XDR

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_int(XDR *xdrs,
               int *ip);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_int()** function is a filter primitive that translates between C-language integers and their external representation.

Parameters

xdrs (Input)
A pointer to the eXternal Data Representation (XDR) stream handle.

ip (I/O)
The address of the integer.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_int()** is used:

```
#include <stdio.h>
#include <xdr.h>
```

```
typedef struct vector
{
        int x,y,z;
} vector ;

bool xdr_vector(XDR *xdrs, vector *p_vector)
{
        if(!xdr_int(xdrs,&(p_vector->x)))
                return FALSE;
        if(!xdr_int(xdrs,&(p_vector->y)))
                return FALSE;
        return xdr_int(xdrs,&(p_vector->z));
}
```

API introduced: V4R2

Top | “Remote Procedure Call (RPC) APIs,” on page 1 | APIs by category

xdr_long()—Translate between Long Integers and Their XDR

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_long(XDR *xdrs,
                long *lp);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_long()** function is a filter primitive that translates between C-language long integers and their external representations.

Parameters

xdrs (Input)
A pointer to the XDR stream handle.

lp (I/O)
The address of the number.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_long()** is used:

```
#include <stdio.h>
#include <xdr.h>

typedef struct vector
{
        long x,y,z;
} vector ;

bool xdr_vector(XDR *xdrs, vector *p_vector)
{
        if(!xdr_long(xdrs,&(p_vector->x)))
                return FALSE;
        if(!xdr_long(xdrs,&(p_vector->y)))
                return FALSE;
        return xdr_long(xdrs,&(p_vector->z));
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

xdr_netobj()—Translate between Netobj Structures and Their XDR

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_netobj(XDR *xdrs,
                  struct netobj *np);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_netobj()** function is a filter primitive that translates between variable-length opaque data and its external representation.

Parameters

xdrs (Input)
A pointer to the eXternal Data Representation (XDR) stream handle.

np (I/O)
A pointer to the address of the netobj structure that contains both a length and a pointer to the opaque data.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	**Error Message Text**
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_netobj()** is used:

```
#include <stdio.h>
#include <xdr.h>

/*
 * Handle of an external client -
 * pid - process ID of the server process on our host
 * oid - object ID of the server assigned to that client
 * Typical case when the other side needs a handle, without
 * actually knowing what is it. We can use xdr_netobj() to send
 * the value
 * or xdr_opaque() to send a pointer.
 */
typedef struct handle
{
        int pid;
        int oid;
} handle ;

bool_t xdr_handle(XDR *xdrs, handle *p_handle )
{
     struct netobj obj;
     obj.n_len=sizeof(handle);
     obj.n_bytes=(char *)p_handle;
     return xdr_netobj(xdrs,&obj);
}
```

API introduced: V4R2

xdr_opaque()—Translate between Fixed-Size Data and Its XDR

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_opaque(XDR *xdrs,
                  caddr_t cp,
                  const u_int cnt);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_opaque()** function is a filter primitive that translates between fixed-size opaque data and its external representation.

Parameters

xdrs (Input)
A pointer to the eXternal Data Representation (XDR) stream handle.

cp (I/O)
The address of the opaque object.

cnt (Input)
The size, in bytes, of the object. By definition, the actual data that is contained in the opaque object will not be portable to another system.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	**Error Message Text**
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_opaque()** is used:

```
#include <stdio.h>
#include <xdr.h>

/*
 * Handle of an external client -
 * pid - process ID of the server process on our host
 * oid - object ID of the server assigned to that client
 * Typical case when the other side needs a handle, without
 * actually knowing what it is. We can use xdr_netobj()
 * or xdr_opaque().
 */
typedef struct handle
{
        int pid;
        int oid;
} handle ;

bool_t xdr_handle(XDR *xdrs, handle *p_handle )
{
     return xdr_opaque(xdrs,(caddr_t)p_handle,sizeof(handle));
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

xdr_pointer()—Provide Pointer Chasing within Structures

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_pointer(XDR *xdrs,
                   char **objpp,
                   u_int objsize,
                   const xdrproc_t xdrobj);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_pointer()** function provides pointer chasing within structures and serializes null pointers. This function can represent recursive data structures, such as binary trees or linked lists.

Pointer chasing is the substitution of the pointer itself with the actual structure it points to.

Parameters

xdrs (Input)
A pointer to the eXternal Data Representation (XDR) stream handle.

objjpp (I/O)
A pointer to the character pointer of the data structure. If decoding and *objpp==NULL, then the memory is allocated by XDR.

objsize (Input)
The size of the structure.

xdrobj (Input)
The XDR filter for the object.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_pointer()** is used:

```
#include <xdr.h>

typedef struct node
{
        int              value;
        struct node *p;
} node ;

bool_t xdr_list(XDR *xdrs, node **p_node)
{
        return xdr_pointer(xdrs,(caddr *)p_node,
        sizeof(node),(xdrproc_t)xdr_node)
}

bool_t xdr_node(XDR *xdrs, node *p_node)
{
        xdr_int(xdrs,&(p_node->value));
        return xdr_list(xdrs,&(p_node->p));
}
```

API introduced: V4R2

xdr_reference()—Provide Pointer Chasing within Structures

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_reference(XDR *xdrs,
                     caddr_t *pp,
                     u_int size,
                     const xdrproc_t proc);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_reference()** function is a filter primitive that provides pointer chasing within structures. This primitive allows the serializing, deserializing, and freeing of any pointers within one structure that are referenced by another structure.

The **xdr_reference()** function does not attach special meaning to a null pointer during serialization, and passing the address of a null pointer may cause a memory error. Therefore, the programmer must describe data with a two-sided discriminated union. One side is used when the pointer is valid; the other side, when the pointer is null.

Pointer chasing is the substitution of the pointer itself with the actual structure it points to.

Parameters

xdrs (Input)
A pointer to the eXternal Data Representation (XDR) stream handle.

pp (I/O)
The address of the structure. When you decode data, XDR allocates storage if the pointer is NULL.

size (Input)
The byte size of the structure pointed to by the *pp* parameter.

proc (Input)
A translation of the structure between its C form and its external representation. This parameter is the XDR procedure that describes the structure.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_reference()** is used:

```
#include <xdr.h>

typedef struct node
{
        int             value;
        struct node *p;
} node ;

/*
 * Do not call it with p_node==NULL, because it will fail.
 */
bool_t xdr_list(XDR *xdrs, node **p_node)
{
        return xdr_reference(xdrs,(caddr_t)p_node,
        sizeof(node),(xdrproc_t)xdr_node)
}

bool_t xdr_node(XDR *xdrs, node *p_node)
{
        xdr_int(xdrs,&(p_node->value));
        return xdr_list(xdrs,&(p_node->p));
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

xdr_short()—Translate between Short Integers and Their XDR

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_short(XDR *xdrs,
                 short *sp);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_short()** function is a filter primitive that translates between C-language short integers and their external representation.

Parameters

xdrs (Input)
A pointer to the eXternal Data Representation (XDR) stream handle.

sp (I/O)
The address of the short integer.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	**Error Message Text**
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_short()** is used:

```
#include <stdio.h>
#include <xdr.h>

typedef struct vector
{
        short x,y,z;
} vector ;

bool_t xdr_vector(XDR *xdrs, vector *p_vector)
{
        if(!xdr_short(xdrs,&(p_vector->x)))
                return FALSE;
        if(!xdr_short(xdrs,&(p_vector->y)))
                return FALSE;
        return xdr_short(xdrs,&(p_vector->z));
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

xdr_string()—Translate between Strings and Their XDR

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_string(XDR *xdrs,
                  char **sp,
                  u_int maxsize);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_string()** function is a filter primitive that translates between C-language strings and their corresponding external representations.

Parameters

xdrs (Input)
A pointer to the eXternal Data Representation (XDR) stream handle.

sp (I/O)
The address of the pointer to the string. If decoding and *sp==NULL, XDR allocated the storage needed for the decoded string.

maxsize (Input)
The maximum length of the string in bytes allowed during encoding or decoding.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	**Error Message Text**
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_string()** is used:

```
#include <stdio.h>
#include <xdr.h>

#define MAX_LENGTH 100

typedef struct adress
{
        char  street[MAX_LENGTH];
        int   number;
        int   apartment;
} address ;

bool_t xdr_address(XDR *xdrs, address *p_address)
{
        if!(xdr_string(xdrs,&(p_address->street),
        MAX_LENGTH))
                return FALSE;
        if(!xdr_int(xdrs,&(p_address->number)))
                return FALSE;
        return xdr_int(xdrs,&(p_address->apartment));
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

xdr_union()—Translate between Unions and Their XDR

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_union(XDR *xdrs,
                 enum_t *dscmp,
                 char *unp,
                 const struct xdr_discrim *choices,
                 const xdrproc_t (*defaultarm));
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_union()** function is a filter primitive that translates between discriminated C unions and their corresponding external representations.

Parameters

xdrs (Input)
A pointer to the eXternal Data Representation (XDR) stream handle.

dscmp (Input)
The address of the union's discriminant. The discriminant is an enumeration (**enum_t**) value.

unp (I/O)
The address of the union.

choices (Input)
A pointer to an array of *xdr_discrim* structures.

defaultarm (Input)
A structure provided in case no discriminants are found. This parameter can have a null value.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	**Error Message Text**
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Usage Notes

The size of any enum data types passed to the **xdr_union()** must be defined as 4 bytes.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_union()** is used:

```
#include <stdio.h>
#include <xdr.h>

#pragma enum size(4)   /* Set enum size to 4 bytes */

typedef enum time_type {END=0,DC,CT} time_type ;

#pragma enum size()   /* Reset enum size */

typedef union time_value
{
        int   discrete_time;
        float continuous_time;
} time_value ;

typedef struct time
{
        time_type  type;
        time_value value;
} time;

bool_t xdr_time(XDR *xdrs, time *p_time)
{
        struct xdr_discrim handlers[] =
        {
                {DT,(xdrproc_t)xdr_int},
                {CT,(xdrproc_t)xdr_float},
                {END,NULL}
        };
        return
xdr_union(xdrs,(enum_t *)(&(p_time->type)),
(caddr_t)&(p_time->value),handlers,NULL);
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

xdr_u_char()—Translate between Unsigned Characters and Their XDR

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_u_char(XDR *xdrs,
                  char *ucp);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_u_char()** function is a filter primitive that translates between unsigned C-language characters and their external representations.

Parameters

xdrs (Input)
A pointer to the eXternal Data Representation (XDR) stream handle.

ucp (I/O)
A pointer to an unsigned character.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_u_char()** is used:

```
#include <stdio.h>
#include <xdr.h>
typedef struct grades
{
        u_char  math; /* Each grade is 'A'..'D' */
        u_char  literature;
         u_char  geography;
        u_char  computers;
} grades ;


bool_t xdr_grades(XDR *xdrs, grades *p_grades)
{
        if(!xdr_u_char(xdrs,&(p_grades->math)))
                return FALSE;
        if(!xdr_u_char(xdrs,&(p_grades->literature)))
                return FALSE;
        if(!xdr_u_char(xdrs,&(p_grades->geography)))
                return FALSE;
        return xdr_u_char(xdrs,&(p_grades->computers));
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

xdr_u_int()—Translate between an Unsigned Integer and Its XDR

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_u_int(XDR *xdrs,
                 u_int *ulp);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_u_int()** function is a filter primitive that translates between C-language unsigned integers and their external representations.

Parameters

xdrs (Input)
A pointer to the eXternal Data Representation (XDR) stream handle.

ulp (I/O)
The address of the unsigned integer.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	**Error Message Text**
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_u_int()** is used:

```
#include <stdio.h>
#include <xdr.h>

typedef struct vector
{
        u_int x,y,z;
} vector ;

bool_t xdr_vector(XDR *xdrs, vector *p_vector)
{
        if(!xdr_u_int(xdrs,&(p_vector->x)))
                return FALSE;
        if(!xdr_u_int(xdrs,&(p_vector->y)))
                return FALSE;
        return xdr_u_int(xdrs,&(p_vector->z));
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

xdr_u_long()—Translate between an Unsigned Long and Its XDR

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_u_long(XDR *xdrs,
                  u_long *ulp);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_u_long()** function is a filter primitive that translates between C-language unsigned long integers and their external representations.

Parameters

xdrs (Input)
A pointer to the eXternal Data Representation (XDR) stream handle.

ulp (I/O)
The address of the unsigned long integer.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_u_long()** is used:

```
#include <stdio.h>
#include <xdr.h>

typedef struct vector
{
        u_long x,y,z;
} vector ;

bool_t xdr_vector(XDR *xdrs, vector *p_vector)
{
        if(!xdr_u_long&((xdrs,p_vector->x)))
                return FALSE;
        if(!xdr_u_long(xdrs,&(p_vector->y)))
                return FALSE;
        return xdr_u_long(xdrs,&(p_vector->z));
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

xdr_u_short()—Translate between an Unsigned Short and Its XDR

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_u_short(XDR *xdrs,
                   u_short *usp);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_u_short()** function is a filter primitive that translates between C-language unsigned short integers and their external representations.

Parameters

xdrs (Input)
A pointer to the eXternal Data Representation (XDR) stream handle.

usp (I/O)
The address of the unsigned short integer.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_u_short()** is used:

```
#include <stdio.h>
#include <xdr.h>

typedef struct vector
{
        u_short x,y,z;
} vector ;

bool_t xdr_vector(XDR *xdrs, vector *p_vector)
{
        if(!xdr_u_short(xdrs,&(p_vector->x)))
                return FALSE;
        if(!xdr_u_short(xdrs,&(p_vector->y)))
                return FALSE;
        return xdr_u_short(xdrs,&(p_vector->z));
}
```

API introduced: V4R2

Top | “Remote Procedure Call (RPC) APIs,” on page 1 | APIs by category

xdr_vector()—Translate between Arrays and Their XDR

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_vector(XDR *xdrs,
                  char *arrp,
                  const u_int size,
                  const u_int elsize,
                  const xdrproc_t elproc);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_vector()** function is a filter primitive that translates between fixed-length arrays and their corresponding external representations.

Parameters

xdrs (Input)
A pointer to the eXternal Data Representation (XDR) stream handle.

arrp (I/O)
The pointer to the array.

size (Input)
The element count of the array.

elsize (Input)
The byte size of each of the array elements.

elproc (Input)
Translates between the C form of the array elements and their external representations. This parameter is an XDR filter.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_vector()** is used:

```
#include <stdio.h>
#include <xdr.h>

#define MAX_VERTECIES 10
#define MAX_EDGES ((MAX_VERTECIES*(MAX_VERTECIES-1))/2)

typedef struct graph
{
        bool_t adjacent[MAX_VERTICIES,MAX_VERTICIES];
} graph ;

bool_t xdr_graph(XDR *xdrs, graph *p_graph)
{
        int i;
        for(i=0;i<MAX_VERTECIES;i++)
                if(!xdr_vector(xdrs,
                p_graph->adjacent[i]
                        AX_VERTECIES,sizeof(bool_t),xdr_bool))
                                return FALSE;
        return TRUE;
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

xdr_void()—Supply an XDR Function to the RPC System

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_void();
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_void()** function has no parameters. It is passed to other RPC functions that require a parameter, but does not transmit data.

Parameters

None

Authorities

No authorization is required.

Return Value

This function always returns a value of TRUE.

Error Conditions

None.

Error Messages

Message ID	Error Message Text
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_void()** is used:

```
#include <stdio.h>
#define RMTPROGNUM (u_long)0x3fffffffL
#define RMTPROGVER (u_long)0x1
#define RMTPROCNUM (u_long)0x1
main()
{
  int inproc=100;
  enum clnt_stat, rstat;
  ...

  /* Service request to host RPCSERVER_HOST */
  if ((rstat = rpc_call("RPCSERVER_HOST", RMTPROGNUM, RMTPROGVER,
                        RMTPROCNUM, xdr_int, (char *)&inproc,
                        xdr_void, (char *)0, "visible")) !=
              RPC_SUCCESS) {
     printf("Error in the rpc_call().\n");
     exit(1);
   }
   ...
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

xdr_wrapstring()—Call the xdr_string() Function

Syntax

```
#include <rpc/xdr.h>

bool_t xdr_wrapstring(XDR *xdrs,
                      char **sp);
```

Service Program Name: QZNFTRPC

Default Public Authority: *USE

Threadsafe: No

The **xdr_wrapstring()** function is a primitive that calls the **xdr_string**(*xdr, sp, maxuint*) API, where *maxuint* is the maximum value of an unsigned integer. The **xdr_wrapstring()** is useful where a translation of xdrproc_t is required. xdrproc_t has only two parameters while the **xdr_string()** function requires three.

Parameters

xdrs (Input)
A pointer to the eXternal Data Representation (XDR) stream handle.

sp (I/O)
The address of the pointer to the string. If decoding and *sp==NULL, XDR allocated memory for the decoded string.

Authorities

No authorization is required.

Return Value

TRUE (1)	Successful
FALSE (0)	Unsuccessful

Error Conditions

None.

Error Messages

Message ID	**Error Message Text**
CPE3418 E	Possible APAR condition or hardware failure.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.

Example

See Code disclaimer information for information pertaining to code examples.

The following example shows how **xdr_wrapstring()** is used:

```
#include <stdio.h>
#include <xdr.h>

#define MAX_LENGTH 100
typedef struct address
{
        char  street[MAX_LENGTH];
        int   number;
        int   apartment;
} address ;

bool_t xdr_address(XDR *xdrs, address *p_address)
{
        if!(xdr_wrapstring(xdrs,&(p_address->street)))
                return FALSE;
        if(!xdr_int(xdrs,&(p_address->number)))
                return FALSE;
        return xdr_int(xdrs,&(p_address->apartment));
}
```

API introduced: V4R2

Top | "Remote Procedure Call (RPC) APIs," on page 1 | APIs by category

Concepts

These are the concepts for this category.

Header Files for Remote Procedure Call APIs

Programs using the Remote Procedure Call (RPC) APIs must include <rpc/rpc.h> and one or more additional header files that contain information needed by the functions, such as:

- Macro definitions
- Data type definitions
- Structure definitions
- Function prototypes

The header files are provided in the QSYSINC library, which is optionally installable. Make sure QSYSINC is on your system before compiling programs that use these header files. For information on installing the QSYSINC library, see Include files and the QSYSINC Library.

The table below shows the file and member name in the QSYSINC library for each header file used by the TI-RPC APIs in the Information Center.

Name of Header File	*Name of File in QSYSINC*	*Name of Member*
netconfig.h [1]	H	NETCONFIG
netdir.h [2]	H	NETDIR
tirpccom.h	H	TIRPCCOM
rpc/auth.h	RPC	AUTH
rpc/auth_sys.h	RPC	AUTH_SYS
rpc/auth_unix.h	RPC	AUTH_UNIX
rpc/clnt.h	RPC	CLNT
rpc/rpc.h	RPC	RPC

Name of Header File	*Name of File in QSYSINC*	*Name of Member*
rpc/rpc_com.h	RPC	RPC_COM
rpc/rpc_msg.h	RPC	RPC_MSG
rpc/rpcb_clnt.h	RPC	RPCB_CLNT
rpc/rpcb_prot.h	RPC	RPCB_PROT
rpc/types.h	RPC	TYPES
rpc/svc.h	RPC	SVC
rpc/svc_auth.h	RPC	SVC_AUTH
rpc/xdr.h	RPC	XDR
Note: 1. The member netconfig.h in the H file in the QSYSINC library is used by the Network Selection functions. 2. The member netdir.h in the H file in the QSYSINC library is used by the Name-to-Address Translation functions.		

You can display a header file in QSYSINC by using one of the following methods:

- Using your editor. For example, to display the **netconfig.h** header file using the Source Entry Utility editor, enter the following command:

  ```
  STRSEU SRCFILE(QSYSINC/H) SRCMBR(NETCONFIG) OPTION(5)
  ```

- Using the Display Physical File Member command. For example, to display the **rpc/rpc.h** header file, enter the following command:

  ```
  DSPPFM FILE(QSYSINC/RPC) MBR(RPC)
  ```

You can print a header file in QSYSINC by using one of the following methods:

- Using your editor. For example, to print the **netdir.h** header file using the Source Entry Utility editor, enter the following command:

  ```
  STRSEU SRCFILE(QSYSINC/H) SRCMBR(netdir) OPTION(6)
  ```

- Using the Copy File command. For example, to print the **rpc/rpc.h** header file, enter the following command:

  ```
  CPYF FROMFILE(QSYSINC/RPC) TOFILE(*PRINT) FROMMBR(RPC)
  ```

Top | “Remote Procedure Call (RPC) APIs,” on page 1 | APIs by category

Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

```
IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY  10504-1785
U.S.A.
```

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

```
IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan
```

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION ″AS IS″ WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Trademarks

Terms and conditions for downloading and printing publications

Code disclaimer information

Printed in USA