NAME

xdr_create, xdr_destroy, xdrmem_create, xdrrec_create, xdrstdio_create – library routines for external data representation stream creation

MT-LEVEL

MT-Safe

DESCRIPTION

XDR library routines allow C programmers to describe arbitrary data structures in a machine-independent fashion. Protocols such as remote procedure calls (RPC) use these routines to describe the format of the data.

These routines deal with the creation of XDR streams. XDR streams have to be created before any data can be translated into XDR format.

Routines

See **rpc**(3N) for the definition of the **XDR**, **CLIENT**, and **SVCXPRT** data structures. Note that any buffers passed to the XDR routines must be properly aligned. It is suggested that **malloc**(3C) be used to allocate these buffers or that the programmer insure that the buffer address is divisible evenly by four.

#include <rpc/xdr.h>

void xdr_destroy(XDR **xdrs***);**

A macro that invokes the destroy routine associated with the XDR stream, *xdrs*. Destruction usually involves freeing private data structures associated with the stream. Using *xdrs* after invoking **xdr_destroy( )** is undefined.

void xdrmem_create(XDR **xdrs***, const caddr_t** *addr***, const u_int** *size***, const enum xdr_op** *op***);**

This routine initializes the XDR stream object pointed to by *xdrs*. The stream's data is written to, or read from, a chunk of memory at location *addr* whose length is no less than *size* bytes long. The *op* determines the direction of the XDR stream (either **XDR_ENCODE**, **XDR_DECODE**, or **XDR_FREE**).

void xdrrec_create(XDR **xdrs***, const u_int** *sendsz***, const u_int** *recvsz***, const caddr_t** *handle***, const int (****readit***)(const void** **read_handle***, char** **buf***, const int** *len***), const int (****writeit***)(const void** **write_handle***, const char** **buf***, const int** *len***));**

This routine initializes the read-oriented XDR stream object pointed to by *xdrs*. The stream's data is written to a buffer of size *sendsz*; a value of **0** indicates the system should use a suitable default. The stream's data is read from a buffer of size *recvsz*; it too can be set to a suitable default by passing a **0** value. When a stream's output buffer is full, *writeit* is called. Similarly, when a stream's input buffer is empty, *readit* is called. The behavior of these two routines is similar to the system calls **read( )** and **write( )** (see **read**(2) and **write**(2), respectively), except that an appropriate handle (*read_handle* or *write_handle*) is passed to the former routines as the first parameter instead of a file descriptor. Note: the XDR stream's *op* field must be set by the caller.

Warning: this XDR stream implements an intermediate record stream. Therefore there are additional bytes in the stream to provide record boundary information.

void xdrstdio_create(XDR *∗xdrs***, FILE** *∗file***, const enum xdr_op** *op***);**

This routine initializes the XDR stream object pointed to by *xdrs*. The XDR stream data is written to, or read from, the standard I/O stream *file*. The parameter *op* determines the direction of the XDR stream (either **XDR_ENCODE**, **XDR_DECODE**, or **XDR_FREE**).

Warning: the destroy routine associated with such XDR streams calls **fflush( )** on the *file* stream, but never **fclose( )** (see **fclose**(3S)).

Failure of any of these functions can be detected by first initializing the *x_ops* field in the **XDR** structure (*xdrs→x_ops*) to **NULL** before calling the xdr∗_create() function. After the return from the xdr∗_create() function, if the *x_ops* field is still **NULL ,** the call has failed. If the *x_ops* field contains some other value, the call can be assumed to have succeeded.

SEE ALSO

read(2), **write**(2), **malloc**(3C), **rpc**(3N), **xdr_admin**(3N), **xdr_complex**(3N), **xdr_simple**(3N), **fclose**(3S)