NAME

xdr - library routines for external data representation

SYNOPSIS AND DESCRIPTION

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

The prototypes below are declared in<*rpc/xdr.h*> and make use of the following types:

typedef int *bool_t***;**

typedef bool_t (***xdrproc_t***) (XDR *, void *,...);**

For the declaration of the *XDR* type, see <*rpc/xdr.h*>.

bool_t xdr_array(XDR ****xdrs*, char *****arrp***, unsigned int ****sizep***,**
unsigned int *maxsize***, unsigned int** *elsize***,**
xdrproc_t *elproc***);**

A filter primitive that translates between variable-length arrays and their corresponding external representations. The argument *arrp* is the address of the pointer to the array, while *sizep* is the address of the element count of the array; this element count cannot exceed *maxsize*. The argument *elsize* is the *sizeof* each of the array's elements, and *elproc* is an XDR filter that translates between the array elements' C form, and their external representation. This routine returns one if it succeeds, zero otherwise.

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

A filter primitive that translates between booleans (C integers) and their external representations. When encoding data, this filter produces values of either one or zero. This routine returns one if it succeeds, zero otherwise.

bool_t xdr_bytes(XDR ****xdrs*, char *****sp***, unsigned int ****sizep***,**
unsigned int *maxsize***);**

A filter primitive that translates between counted byte strings and their external representations. The argument *sp* is the address of the string pointer. The length of the string is located at address *sizep*; strings cannot be longer than *maxsize*. This routine returns one if it succeeds, zero otherwise.

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

A filter primitive that translates between C characters and their external representations. This routine returns one if it succeeds, zero otherwise. Note: encoded characters are not packed, and occupy 4 bytes each. For arrays of characters, it is worthwhile to consider **xdr_bytes**(), **xdr_opaque**() or **xdr_string**().

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.

bool_t xdr_double(XDR ****xdrs*, double ****dp***);**

A filter primitive that translates between C *double* precision numbers and their external representations. This routine returns one if it succeeds, zero otherwise.

bool_t xdr_enum(XDR ****xdrs*, enum_t ****ep***);**

A filter primitive that translates between C *enum*s (actually integers) and their external representations. This routine returns one if it succeeds, zero otherwise.

bool_t xdr_float(XDR ****xdrs*, float ****fp***);**

A filter primitive that translates between C *float*s and their external representations. This routine returns one if it succeeds, zero otherwise.

void xdr_free(xdrproc_t *proc***, char ****objp***);**

Generic freeing routine. The first argument is the XDR routine for the object being freed. The second argument is a pointer to the object itself. Note: the pointer passed to this routine is *not* freed, but what it points to *is* freed (recursively).

unsigned int xdr_getpos(XDR ****xdrs*);**

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

long *xdr_inline(XDR ****xdrs*, int** *len***);**

A macro that invokes the inline routine associated with the XDR stream, *xdrs*. The routine returns a pointer to a contiguous piece of the stream's buffer; *len* is the byte length of the desired buffer. Note: pointer is cast to *long **.

Warning: **xdr_inline**() may return NULL (0) if it cannot allocate a contiguous piece of a buffer. Therefore the behavior may vary among stream instances; it exists for the sake of efficiency.

bool_t xdr_int(XDR ****xdrs*, int ****ip***);**

A filter primitive that translates between C integers and their external representations. This routine returns one if it succeeds, zero otherwise.

bool_t xdr_long(XDR ****xdrs*, long ****lp***);**

A filter primitive that translates between C *long* integers and their external representations. This routine returns one if it succeeds, zero otherwise.

void xdrmem_create(XDR ****xdrs*, char ****addr***, unsigned int** *size***,**
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 more than *size* bytes long. The *op* determines the direction of the XDR stream (either **XDR_ENCODE**, **XDR_DECODE**, or **XDR_FREE**).

bool_t xdr_opaque(XDR ****xdrs*, char ****cp***, unsigned int** *cnt***);**

A filter primitive that translates between fixed size opaque data and its external representation. The argument *cp* is the address of the opaque object, and *cnt* is its size in bytes. This routine returns one if it succeeds, zero otherwise.

bool_t xdr_pointer(XDR ****xdrs*, char *****objpp***,**
unsigned int *objsize***, xdrproc_t** *xdrobj***);**

Like **xdr_reference**() except that it serializes null pointers, whereas **xdr_reference**() does not. Thus, **xdr_pointer**() can represent recursive data structures, such as binary trees or linked lists.

void xdrrec_create(XDR ****xdrs*, unsigned int** *sendsize***,**
unsigned int *recvsize***, char ****handle***,**
int (**readit***) (char *, char *, int),**
int (**writeit***) (char *, char *, int));**

This routine initializes the XDR stream object pointed to by *xdrs*. The stream's data is written to a buffer of size *sendsize*; a value of zero indicates the system should use a suitable default. The stream's data is read from a buffer of size *recvsize*; it too can be set to a suitable default by passing a zero 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(2) and write(2), except that *handle* is passed to the former routines as the first argument. Note: the XDR stream's *op* field must be set by the caller.

Warning: to read from an XDR stream created by this API, you'll need to call **xdrrec_skiprecord**() first before calling any other XDR APIs. This inserts additional bytes in the stream to provide record boundary information. Also, XDR streams created with different **xdr*_create** APIs are not compatible for the same reason.

bool_t xdrrec_endofrecord(XDR *_xdrs_, int _sendnow_);

This routine can be invoked only on streams created by **xdrrec_create**(). The data in the output buffer is marked as a completed record, and the output buffer is optionally written out if *sendnow* is nonzero. This routine returns one if it succeeds, zero otherwise.

bool_t xdrrec_eof(XDR *_xdrs_);

This routine can be invoked only on streams created by **xdrrec_create**(). After consuming the rest of the current record in the stream, this routine returns one if the stream has no more input, zero otherwise.

bool_t xdrrec_skiprecord(XDR *_xdrs_);

This routine can be invoked only on streams created by **xdrrec_create**(). It tells the XDR implementation that the rest of the current record in the stream's input buffer should be discarded. This routine returns one if it succeeds, zero otherwise.

bool_t xdr_reference(XDR *_xdrs_, char **_pp_, unsigned int _size_,
xdrproc_t _proc_**);**

A primitive that provides pointer chasing within structures. The argument *pp* is the address of the pointer; *size* is the *sizeof* the structure that **pp* points to; and *proc* is an XDR procedure that filters the structure between its C form and its external representation. This routine returns one if it succeeds, zero otherwise.

Warning: this routine does not understand null pointers. Use **xdr_pointer**() instead.

xdr_setpos(XDR *_xdrs_, unsigned int _pos_);

A macro that invokes the set position routine associated with the XDR stream *xdrs*. The argument *pos* is a position value obtained from **xdr_getpos**(). This routine returns one if the XDR stream could be repositioned, and zero otherwise.

Warning: it is difficult to reposition some types of XDR streams, so this routine may fail with one type of stream and succeed with another.

bool_t xdr_short(XDR *_xdrs_, short *_sp_);

A filter primitive that translates between C *short* integers and their external representations. This routine returns one if it succeeds, zero otherwise.

void xdrstdio_create(XDR *_xdrs_, FILE *_file_, 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 *stdio* stream *file*. The argument *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(3) on the *file* stream, but never fclose(3).

bool_t xdr_string(XDR *_xdrs_, char **_sp_, unsigned int _maxsize_**);**

A filter primitive that translates between C strings and their corresponding external representations. Strings cannot be longer than*maxsize*. Note:*sp* is the address of the string' s pointer. This routine returns one if it succeeds, zero otherwise.

bool_t xdr_u_char(XDR *_xdrs_, unsigned char *_ucp_);

A filter primitive that translates between *unsigned* C characters and their external representations. This routine returns one if it succeeds, zero otherwise.

bool_t xdr_u_int(XDR *_xdrs_, unsigned *_up_);

A filter primitive that translates between C *unsigned* integers and their external representations. This routine returns one if it succeeds, zero otherwise.

bool_t xdr_u_long(XDR ****xdrs*, unsigned long ****ulp***);**

A filter primitive that translates between C *unsigned long* integers and their external representations. This routine returns one if it succeeds, zero otherwise.

bool_t xdr_u_short(XDR ****xdrs*, unsigned short ****usp***);**

A filter primitive that translates between C *unsigned short* integers and their external representations. This routine returns one if it succeeds, zero otherwise.

bool_t xdr_union(XDR ****xdrs*, int ****dscmp***, char ****unp***,**
struct xdr_discrim ****choices*,**
xdrproc_t *defaultarm***); /* may equal NULL */**

A filter primitive that translates between a discriminated C *union* and its corresponding external representation. It first translates the discriminant of the union located at*dscmp*. This discriminant is always an *enum_t*. Next the union located at *unp* is translated. The argument *choices* is a pointer to an array of **xdr_discrim**() structures. Each structure contains an ordered pair of [*value*,*proc*]. If the union's discriminant is equal to the associated *value*, then the *proc* is called to translate the union. The end of the **xdr_discrim**() structure array is denoted by a routine of value NULL. If the discriminant is not found in the*c hoices* array, then the *defaultarm* procedure is called (if it is not NULL). Returns one if it succeeds, zero otherwise.

bool_t xdr_vector(XDR ****xdrs*, char ****arrp***, unsigned int** *size***,**
unsigned int *elsize***, xdrproc_t** *elproc***);**

A filter primitive that translates between fixed-length arrays and their corresponding external representations. The argument *arrp* is the address of the pointer to the array, while *size* is the element count of the array. The argument *elsize* is the *sizeof* each of the array's elements, and *elproc* is an XDR filter that translates between the array elements' C form, and their external representation. This routine returns one if it succeeds, zero otherwise.

bool_t xdr_void(void);

This routine always returns one. It may be passed to RPC routines that require a function argument, where nothing is to be done.

bool_t xdr_wrapstring(XDR ****xdrs*, char *****sp***);**

A primitive that calls **xdr_string(xdrs, sp,MAXUN.UNSIGNED);** where **MAXUN.UNSIGNED** is the maximum value of an unsigned integer. **xdr_wrapstring**() is handy because the RPC package passes a maximum of two XDR routines as arguments, and **xdr_string**(), one of the most frequently used primitives, requires three. Returns one if it succeeds, zero otherwise.

ATTRIBUTES

For an explanation of the terms used in this section, see attributes(7).

Interface	Attribute	Value
xdr_array(), **xdr_bool**(), **xdr_bytes**(), **xdr_char**(), **xdr_destroy**(), **xdr_double**(), **xdr_enum**(), **xdr_float**(), **xdr_free**(), **xdr_getpos**(), **xdr_inline**(), **xdr_int**(), **xdr_long**(), **xdrmem_create**(), **xdr_opaque**(), **xdr_pointer**(), **xdrrec_create**(), **xdrrec_eof**(), **xdrrec_endofrecord**(), **xdrrec_skiprecord**(), **xdr_reference**(), **xdr_setpos**(), **xdr_short**(), **xdrstdio_create**(), **xdr_string**(), **xdr_u_char**(), **xdr_u_int**(), **xdr_u_long**(), **xdr_u_short**(), **xdr_union**(), **xdr_vector**(), **xdr_void**(), **xdr_wrapstring**()	Thread safety	MT-Safe

SEE ALSO

rpc(3)

The following manuals:

eXternal Data Representation Standard: Protocol Specification
eXternal Data Representation: Sun Technical Notes
XDR: External Data Representation Standard, RFC 1014, Sun Microsystems, Inc., USC-ISI.

COLOPHON

This page is part of release 4.10 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.