NAME

xdr_complex, xdr_array, xdr_bytes, xdr_opaque, xdr_pointer, xdr_reference, xdr_string, xdr_union, xdr_vector, xdr_wrapstring – library routines for external data representation

MT-LEVEL

Safe

DESCRIPTION

XDR library routines allow C programmers to describe complex 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 are the XDR library routines for complex data structures. They require the creation of XDR stream (see **xdr_create**(3N)).

Routines

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

#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***);**

xdr_array( ) translates between variable-length arrays and their corresponding external representations. The parameter *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 parameter *elsize* is the size of each of the array's elements, and *elproc* is an XDR routine that translates between the array elements' C form and their external representation. If **aarp* is null when decoding, **xdr_array( )** allocates memory and **aarp* points to it. This routine returns **TRUE** if it succeeds, **FALSE** otherwise.

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

xdr_bytes( ) translates between counted byte strings and their external representations. The parameter *sp* is the address of the string pointer. The length of the string is located at address *sizep*; strings cannot be longer than *maxsize*. If **sp* is null when decoding, **xdr_bytes( )** allocates memory and **sp* points to it. This routine returns **TRUE** if it succeeds, **FALSE** otherwise.

bool_t xdr_opaque(XDR **xdrs*, **caddr_t** *cp*, **const u_int** *cnt***);**

xdr_opaque( ) translates between fixed size opaque data and its external representation. The parameter *cp* is the address of the opaque object, and *cnt* is its size in bytes. This routine returns **TRUE** if it succeeds, **FALSE** otherwise.

bool_t xdr_pointer(XDR **xdrs*, **char** ***objpp*, **u_int** *objsize*, **const 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. If **objpp* is null when decoding, **xdr_pointer( )** allocates memory and **objpp* points to it.

bool_t xdr_reference(XDR **xdrs*, **caddr_t** **pp*, **u_int** *size*, **const xdrproc_t** *proc*);

xdr_reference( ) provides pointer chasing within structures. The parameter *pp* is the address of the pointer; *size* is the **sizeof** the structure that **pp* points to; and *proc* is an XDR procedure that translates the structure between its C form and its external representation. If **pp* is null when decoding, **xdr_reference( )** allocates memory and **pp* points to it. This routine returns **1** if it succeeds, **0** otherwise.

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

bool_t xdr_string(XDR **xdrs*, **char** ***sp*, **const u_int** *maxsize*);

xdr_string( ) 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. If **sp* is null when decoding, **xdr_string( )** allocates memory and **sp* points to it. This routine returns **TRUE** if it succeeds, **FALSE** otherwise. Note: **xdr_string( )** can be used to send an empty string (" "), but not a NULL string.

bool_t xdr_union(XDR **xdrs*, **enum_t** **dscmp*, **char** **unp*,
const struct xdr_discrim **choices*, **const xdrproc_t (****defaultarm*);

xdr_union( ) 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 parameter *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 *choices* array, then the *defaultarm* procedure is called (if it is not **NULL**). Returns **TRUE** if it succeeds, **FALSE** otherwise.

bool_t xdr_vector(XDR **xdrs*, **char** **arrp*, **const u_int** *size*, **const u_int** *elsize*,
const xdrproc_t *elproc*);

xdr_vector( ) translates between fixed-length arrays and their corresponding external representations. The parameter *arrp* is the address of the pointer to the array, while *size* is the element count of the array. The parameter *elsize* is the **sizeof** each of the array's elements, and *elproc* is an XDR routine that translates between the array elements' C form and their external representation. This routine returns **TRUE** if it succeeds, **FALSE** otherwise.

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

A routine that calls **xdr_string(***xdrs*, *sp*, *maxuint***);** where *maxuint* is the maximum value of an unsigned integer.

Many routines, such as **xdr_array( )**, **xdr_pointer( )**, and **xdr_vector( )** take a function pointer of type **xdrproc_t( )**, which takes two arguments. **xdr_string( )**, one of the most frequently used routines, requires three arguments, while **xdr_wrapstring( )** only requires two. For these routines, **xdr_wrapstring( )** is desirable. This routine returns **TRUE** if it succeeds, **FALSE** otherwise.

SEE ALSO

rpc(3N), **xdr_admin**(3N), **xdr_create**(3N), **xdr_simple**(3N)