

# ASN1C

---

ASN.1 Compiler  
Version 5.8  
C/C++ XER/XML Runtime  
Reference Manual



The software described in this document is furnished under a license agreement and may be used only in accordance with the terms of this agreement.

#### Copyright Notice

Copyright © 1997-2005 Objective Systems, Inc.

All Rights Reserved

This document may be distributed in any form, electronic or otherwise, provided that it is distributed in its entirety and that the copyright and this notice are included.

#### **Author's Contact Information:**

Comments, suggestions, and inquiries regarding ASN1C may be submitted via electronic mail to [info@obj-sys.com](mailto:info@obj-sys.com).



## CHANGE HISTORY

<b>Date</b>	<b>Author</b>	<b>Version</b>	<b>Description</b>
11/16/2005	ED	5.8	Initial version

---

# Table of Contents

<b>ASN1C XER RUNTIME CLASSES AND LIBRARY FUNCTIONS.....</b>	<b>3</b>
<b>MODULE DOCUMENTATION .....</b>	<b>4</b>
XER C++ RUNTIME CLASSES.....	4
XER MESSAGE BUFFER CLASSES .....	4
C++ CLASSES FOR STREAMING XER DECODING.....	5
C++ CLASSES FOR STREAMING XER ENCODING.....	5
XER RUNTIME LIBRARY FUNCTIONS.....	5
XER C DECODE FUNCTIONS.....	6
XER C ENCODE FUNCTIONS.....	16
XER C UTILITY FUNCTIONS.....	26
XML C DECODE FUNCTIONS.....	28
XML C ENCODE FUNCTIONS.....	32
<b>CLASS DOCUMENTATION .....</b>	<b>36</b>
ASN1XERDECODEBUFFER .....	36
ASN1XERENCODEBUFFER .....	39
ASN1XERFILEINPUTSTREAM.....	42
ASN1XERFILEOUTPUTSTREAM .....	44
ASN1XERINPUTSTREAM.....	45
ASN1XERMESAGEBUFFER.....	47
ASN1XEROUTPUTSTREAM.....	48
ASN1XERSAXDECODEHANDLER.....	55
ASN1XERSOCKETINPUTSTREAM .....	58
ASN1XERSOCKETOUTPUTSTREAM.....	60
OSRTSAXEXCEPTION.....	62
OSXMLCONTENTHANDLER.....	63
<b>FILE DOCUMENTATION .....</b>	<b>65</b>
ASN1CXEROPENTYPE.H.....	65
ASN1SAX_XEROPENTYPE.H .....	66
ASN1XER.H.....	67
ASN1XERCPPTYPES.H.....	70
ASN1XERCTYPES.H .....	71
ASN1XERINPUTSTREAM.H.....	73
ASN1XEROUTPUTSTREAM.H .....	74
<b>INDEX.....</b>	<b>75</b>

# ASN1C XER Runtime Classes and Library Functions

The **ASN.1 C++ runtime classes** are wrapper classes that provide an object-oriented interface to the ASN.1 C Runtime Library functions. The classes described in this manual are derived from the common classes documented in the ASN1C C/C++ Common Runtime manual. They are specific to the XML Encoding Rules (XER) as defined in the X.693 ITU-T standard.

These XER specific C++ runtime classes include:

- classes for streaming XER decoding
- classes for streaming XER encoding.

The **ASN.1 XER Runtime Library** contains the low-level constants, types, and functions that are assembled by the compiler to encode/decode more complex structures.

This library consists of the following items:

- A global include file ("asn1xer.h") that is compiled into all generated source files.
- An object library of functions that are linked with the C functions after compilation with a C compiler.

In general, programmers will not need to be too concerned with the details of these functions. The ASN.1 compiler generates calls to them in the C or C++ source files that it creates. However, the functions in the library may also be called on their own in applications requiring their specific functionality.

# ASN1C XER Runtime Module Documentation

## XER C++ Runtime Classes.

### Modules

- XER Message Buffer Classes
- C++ Classes for Streaming XER Decoding.
- C++ Classes for Streaming XER Encoding.

### Defines

- #define ASN1SAXTHROW(stat) throwSAXException (stat);
- #define ASN1SAXCATCH0(toCatch, stat)
- #define ASN1SAXCATCH(toCatch, stat)

---

### Define Documentation

#### #define ASN1SAXCATCH(toCatch, stat)

```
Value:ASN1SAXCATCH0(toCatch,stat) \
catch (const ASN1MessageBuffer::RTLError& toCatch) { \
    cerr << "ASN.1 RTL ERROR: " << toCatch.getStatus() << endl; \
    stat = toCatch.getStatus(); \
}
```

#### #define ASN1SAXCATCH0(toCatch, stat)

```
Value:catch (OSRTSAXException& toCatch) { \
    cerr << "ASN1XERSAX ERROR: "; \
    if (toCatch.getMessage()) \
        cerr << StrX (toCatch.getMessage()); \
    stat = toCatch.getStatus(); \
    if (stat) \
        cerr << "\n" << "Status: " << stat << endl; \
}
```

## XER Message Buffer Classes

---

### Detailed Description

These classes manage the buffers for encoding and decoding ASN.1 XER messages.

### Classes

- class ASN1XERMessageBuffer
- class ASN1XEREncodeBuffer
- class ASN1XERSAXDecodeHandler

- class `ASN1XERDecodeBuffer`

## C++ Classes for Streaming XER Decoding.

---

### Detailed Description

These classes are used to perform XER decoding directly from a stream (file, network, memory).

### Classes

- class `ASN1XERInputStream`
- class `ASN1XERFileInputStream`
- class `ASN1XERSocketInputStream`

## C++ Classes for Streaming XER Encoding.

---

### Detailed Description

These classes are used to perform XER encoding directly to a stream (file, network, memory).

### Classes

- class `ASN1XEROutputStream`
- class `ASN1XERFileOutputStream`
- class `ASN1XERSocketOutputStream`

## XER Runtime Library Functions.

---

### Detailed Description

The ASN.1 XML Encoding Rules (XER) runtime library contains low-level constants, types, and functions that are assembled by the ASN1C compiler to encode/decode more complex structures.

The XER low-level C encode/decode functions are identified by their prefixes: `xerEnc` for XER encode, `xerDec` for XER decode, and `xer` for XER utility functions.

### Modules

- **XER C Decode Functions.**
- **XER C Encode Functions.**
- **XER C Utility Functions.**
- **XML C Decode Functions.**
- **XML C Encode Functions.**

### Classes

- struct `XerElemInfo`

- struct **XmlNamedBitsDict**
- struct **OSXMLNamespace**

### Defines

- #define **XERINDENT** 3
- #define **XERBYTECNT**(pctxt) (pctxt)->buffer.byteIndex
- #define **EXTERNXER**

### Typedefs

- typedef XmlNamedBitsDict **XmlNamedBitsDict**
- typedef OSXMLNamespace **OSXMLNamespace**

### Enumerations

- enum **ASN1XERState** { **XERINIT**, **XERSTART**, **XERDATA**, **XEREND**, **XERSTART0**, **XEREND0** }

## XER C Decode Functions.

---

### Detailed Description

XER runtime library decode functions handle the decoding of the primitive ASN.1 data types and length variables. Calls to these functions are assembled in the C source code generated by the ASN1C compiler to decode complex ASN.1 structures. These functions are also directly callable from within a user's application program if the need to decode a primitive data item exists.

### Functions

- int **xerDecBMPStr** (ASN1CTXT \*pctxt, ASN1BMPString \*outdata)
- int **xerDecBase64Str** (ASN1CTXT \*pctxt, ASN1OCTET \*pvalue, ASN1UINT \*pnocts, ASN1INT bufsize)
- int **xerDecBigInt** (ASN1CTXT \*pctxt, char \*\*ppvalue, int radix)
- int **xerDecBitStr** (ASN1CTXT \*pctxt, ASN1OCTET \*pvalue, ASN1UINT \*pnbits, ASN1INT bufsize)
- int **xerDecBitStrMemBuf** (ASN1MemBuf \*pMemBuf, ASN1Const XMLCHAR \*inpdata, int length, ASN1BOOL skipWhitespaces)
- int **xerDecBool** (ASN1CTXT \*pctxt, ASN1BOOL \*pvalue)
- int **xerDecCopyBitStr** (ASN1CTXT \*pctxt, ASN1OCTET \*pvalue, ASN1UINT \*pnbits, ASN1INT bufsize, int lastBitOffset)
- int **xerDecCopyDynBitStr** (ASN1CTXT \*pctxt, ASN1DynBitStr \*pvalue, int lastBitOffset)
- int **xerDecCopyDynOctStr** (ASN1CTXT \*pctxt, ASN1DynOctStr \*pvalue, int lastBitOffset)
- int **xerDecCopyOctStr** (ASN1CTXT \*pctxt, ASN1OCTET \*pvalue, ASN1UINT \*pnocts, ASN1INT bufsize, int lastBitOffset)
- int **xerDecDynAscCharStr** (ASN1CTXT \*pctxt, ASN1ConstCharPtr \*outdata)
- int **xerDecDynBase64Str** (ASN1CTXT \*pctxt, ASN1DynOctStr \*pvalue)
- int **xerDecDynBitStr** (ASN1CTXT \*pctxt, ASN1DynBitStr \*pvalue)
- int **xerDecDynOctStr** (ASN1CTXT \*pctxt, ASN1DynOctStr \*pvalue)
- int **xerDecDynUTF8Str** (ASN1CTXT \*pctxt, ASN1UTF8String \*outdata)
- int **xerDecInt** (ASN1CTXT \*pctxt, ASN1INT \*pvalue)
- int **xerDecInt8** (ASN1CTXT \*pctxt, ASN1INT8 \*pvalue)
- int **xerDecInt16** (ASN1CTXT \*pctxt, ASN1SINT \*pvalue)

- `int xerDecInt64 (ASN1CTXT *pctx, ASN1INT64 *pvalue)`
- `int xerDecObjId (ASN1CTXT *pctx, ASN1OBJID *pvalue)`
- `int xerDecObjId64 (ASN1CTXT *pctx, ASN1OID64 *pvalue)`
- `int xerDecOctStr (ASN1CTXT *pctx, ASN1OCTET *pvalue, ASN1UINT *pnocts, ASN1INT bufsize)`
- `int xerDecOctStrMemBuf (ASN1MemBuf *pMemBuf, ASN1Const XMLCHAR *inpdata, int length, ASN1BOOL skipWhitespaces)`
- `int xerDecOpenType (ASN1CTXT *pctx, ASN1OpenType *pvalue)`
- `int xerDecReal (ASN1CTXT *pctx, ASN1REAL *pvalue)`
- `int xerDecRelativeOID (ASN1CTXT *pctx, ASN1OBJID *pvalue)`
- `int xerDecUInt (ASN1CTXT *pctx, ASN1UINT *pvalue)`
- `int xerDecUInt8 (ASN1CTXT *pctx, ASN1UINT8 *pvalue)`
- `int xerDecUInt16 (ASN1CTXT *pctx, ASN1USINT *pvalue)`
- `int xerDecUInt64 (ASN1CTXT *pctx, ASN1UINT64 *pvalue)`
- `int xerDecUnivStr (ASN1CTXT *pctx, ASN1UniversalString *outdata)`
- `int xerSetDecBufPtr (ASN1CTXT *pctx, ASN1ConstOctetPtr bufaddr, size_t bufsiz)`

## Function Documentation

**`int xerDecBase64Str (ASN1CTXT * pctx, ASN1OCTET * pvalue, ASN1UINT * pnocts, ASN1INT bufsize)`**

This function will decode a variable of the ASN.1 OCTET STRING type into a static memory structure. The octet string must be Base64 encoded. This function call is used to decode a sized octet string production.

### Parameters:

- pctx* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
- pvalue* A pointer to a variable to receive the decoded bit string. This is assumed to be a static array large enough to hold the number of octets specified in the bufsize input parameter.
- pnocts* A pointer to an integer value to receive the decoded number of octets.
- bufsize* A integer variable containing the size (in octets) of the sized ASN.1 octet string. An error will occur if the number of octets in the decoded string is larger than this value.

### Returns:

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**`int xerDecBigInt (ASN1CTXT * pctx, char ** ppvalue, int radix)`**

This function will decode a variable of the ASN.1 INTEGER type. In this case, the integer is assumed to be of a larger size than can fit in a C or C++ long type (normally 32 or 64 bits). For example, parameters used to calculate security values are typically larger than these sizes.

These variables are stored in character string constant variables. Depending on specified radix, they could be represented as binary, octal, decimal or hexadecimal strings starting with appropriate prefix. If it is necessary to convert to another radix, then use `rtBigIntSetStr` or `rtBigIntToString` functions.

### Parameters:

- pctx* Pointer to context block structure.

*ppvalue* Pointer to a character pointer variable to receive the decoded unsigned value. Dynamic memory is allocated for the variable using the `rtMemAlloc` function. The decoded variable is represented as a string starting with appropriate prefix.

*radix* The expected radix of the decoded value. The only radices 2, 8, 10 and 16 are supported.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecBitStr (ASN1CTXT \* *pctxt*, ASN1OCTET \* *pvalue*, ASN1UINT \* *pnbits*, ASN1INT *bufsize*)**

This function will decode a variable of the ASN.1 BIT STRING type into a static memory structure. This function call is generated by ASN1C to decode a sized bit production.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*pvalue* A pointer to a variable to receive the decoded bit string. This is assumed to be a static array large enough to hold the number of octets specified in the *bufsize* input parameter.

*pnbits* A pointer to an integer value to receive the decoded number of bits.

*bufsize* An integer variable containing the size (in octets) of the sized ASN.1 bit string. An error will occur if the number of octets in the decoded string is larger than this value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecBitStrMemBuf (ASN1MemBuf \* *pMemBuf*, ASN1Const XMLCHAR \* *inpdata*, int *length*, ASN1BOOL *skipWhitespaces*)**

This function decodes a variable of the ASN.1 BIT STRING type to a memory buffer. The decoded data will be put into the memory buffer starting from the current position and bit offset. After all data is decoded the bit string may be fetched out by call to `xerDecCopyBitStr` or `xerDecCopyDynBitStr` functions.

Usually, this function is used in the 'characters' SAX handler.

**Parameters:**

*pMemBuf* Pointer to the destination memory buffer.

*inpdata* Pointer to a source string to be decoded.

*length* Length of the source string (in characters).

*skipWhitespaces* Indicates, could whitespaces be ignored or they are illegal.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecBMPStr (ASN1CTXT \* *pctxt*, ASN1BMPString \* *outdata*)**

This function will decode a variable ASN.1 16-bit character BMPString type. This function will allocate dynamic memory to store the decoded result.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*outdata* A pointer to a structure variable to receive the decoded string. The string is stored as an array of short integer characters. The memory is allocated for the string by the `rtMemAlloc` function.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecBool (ASN1CTXT \* *pctxt*, ASN1BOOL \* *pvalue*)**

This function decodes a variable of the ASN.1 BOOLEAN type.

**Parameters:**

*pctxt* Pointer to context block structure.

*pvalue* Pointer to a variable to receive the decoded BOOLEAN value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecCopyBitStr (ASN1CTXT \* *pctxt*, ASN1OCTET \* *pvalue*, ASN1UINT \* *pnbits*, ASN1INT *bufsize*, int *lastBitOffset*)**

This function copies the decoded BIT STRING from the memory buffer. This function call is generated by ASN1C to decode a sized bit string production.

**Parameters:**

*pctxt* Pointer to context block structure. Its buffer should be set to point to the decoded data.

*pvalue* Pointer to a buffer to receive the decoded data.

*pnbits* Pointer to an integer value to receive the decoded number of bits.

*bufsize* An integer variable containing the size (in octets) of the sized ASN.1 bit string. An error will occur if the number of octets in the decoded string is larger than this value.

*lastBitOffset* A number of actual bits in the last octet.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecCopyDynBitStr (ASN1CTXT \* *pctxt*, ASN1DynBitStr \* *pvalue*, int *lastBitOffset*)**

This function copies the decoded BIT STRING from the memory buffer. This function will allocate dynamic memory to store the decoded result.

**Parameters:**

*pctxt* Pointer to context block structure. Its buffer should be set to point to the decoded data.

*pvalue* Pointer to a dynamic bit string structure to receive the decoded bit string. Dynamic memory is allocated to hold the string using the `rtMemAlloc` function.

*lastBitOffset* A number of actual bits in the last octet.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecCopyDynOctStr (ASN1CTXT \* *pctxt*, ASN1DynOctStr \* *pvalue*, int *lastBitOffset*)**

This function copies the decoded OCTET STRING from the memory buffer. This function will allocate dynamic memory to store the decoded result.

**Parameters:**

*pctxt* Pointer to context block structure. Its buffer should be set to point to the decoded data.  
*pvalue* Pointer to a dynamic octet string structure to receive the decoded octet string. Dynamic memory is allocated to hold the string using the rtMemAlloc function.  
*lastBitOffset* A number of actual bits in the last octet.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecCopyOctStr (ASN1CTXT \* *pctxt*, ASN1OCTET \* *pvalue*, ASN1UINT \* *pnocts*, ASN1INT *bufsize*, int *lastBitOffset*)**

This function copies the decoded OCTET STRING from the memory buffer. This function call is generated by ASN1C to decode a sized octet string production.

**Parameters:**

*pctxt* Pointer to context block structure. Its buffer should be set to point to the decoded data.  
*pvalue* Pointer to a variable to receive the decoded bit string. This is assumed to be a static array large enough to hold the number of octets specified in the 'bufsize' input parameter.  
*pnocts* Pointer to an integer value to receive the decoded number of octets.  
*bufsize* An integer variable containing the size (in octets) of the sized ASN.1 octet string. An error will occur if the number of octets in the decoded string is larger than this value.  
*lastBitOffset* A number of actual bits in the last octet.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecDynAscCharStr (ASN1CTXT \* *pctxt*, ASN1ConstCharPtr \* *outdata*)**

This function will decode a variable of one of the ASN.1 8-bit character string types. These types include IA5String, VisibleString, PrintableString, and NumericString. This function will allocate dynamic memory to store the result.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.  
*outdata* A pointer to a character string pointer variable to receive the decoded string. The string is stored as a standard null-terminated C string. Memory is allocated for the string by the rtMemAlloc function.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecDynBase64Str (ASN1CTXT \* *pctxt*, ASN1DynOctStr \* *pvalue*)**

This function will decode a variable of the ASN.1 OCTET STRING type. The octet string must be Base64 encoded. This function will allocate dynamic memory to store the decoded result.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*pvalue* A pointer to a dynamic octet string structure to receive the decoded octet string. Dynamic memory is allocated to hold the string using the `rtMemAlloc` function.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecDynBitStr (ASN1CTXT \* *pctxt*, ASN1DynBitStr \* *pvalue*)**

This function will decode a variable of the ASN.1 BIT STRING type. This function will allocate dynamic memory to store the decoded result.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*pvalue* A pointer to a dynamic bit string structure to receive the decoded bit string. Dynamic memory is allocated to hold the string using the `rtMemAlloc` function.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecDynOctStr (ASN1CTXT \* *pctxt*, ASN1DynOctStr \* *pvalue*)**

This function will decode a variable of the ASN.1 OCTET STRING type. This function will allocate dynamic memory to store the decoded result.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*pvalue* A pointer to a dynamic bit string structure to receive the decoded bit string. Dynamic memory is allocated to hold the string using the `rtMemAlloc` function.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecDynUTF8Str (ASN1CTXT \* *pctxt*, ASN1UTF8String \* *outdata*)**

This function will decode a variable of UTF8String ASN.1 type. Generally, the SAX parser converts all UTF8 format strings to 16-bit Unicode format automatically. This function converts the Unicode string back to UTF8 format. This function will allocate dynamic memory to store the decoded result.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*outdata* A pointer to a character string pointer variable to receive the decoded string. The string is stored as a UTF8 null-terminated string. Memory is allocated for the string by the `rtMemAlloc` function.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecInt (ASN1CTXT \* *pctxt*, ASN1INT \* *pvalue*)**

This function decodes a variable of the ASN.1 INTEGER type.

**Parameters:**

*pctxt* Pointer to context block structure.

*pvalue* Pointer to a variable to receive the decoded integer value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecInt16 (ASN1CTXT \* *pctxt*, ASN1SINT \* *pvalue*)**

This function decodes a 16-bit variable of the ASN.1 INTEGER type.

**Parameters:**

*pctxt* Pointer to context block structure.

*pvalue* Pointer to a 16-bit variable to receive the decoded integer value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecInt64 (ASN1CTXT \* *pctxt*, ASN1INT64 \* *pvalue*)**

This function decodes a 64-bit variable of the ASN.1 INTEGER type.

**Parameters:**

*pctxt* Pointer to context block structure.

*pvalue* Pointer to a 64-bit variable to receive the decoded integer value. The ASN1INT64 type is set to the C type '`__int64`', '`long long`' or '`long`' in the `asn1type.h` file (depends on the used platform and the compiler).

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecInt8 (ASN1CTXT \* *pctxt*, ASN1INT8 \* *pvalue*)**

This function decodes an 8-bit variable of the ASN.1 INTEGER type.

**Parameters:**

*pctxt* Pointer to context block structure.

*pvalue* Pointer to an 8-bit variable to receive the decoded integer value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecObjId (ASN1CTXT \* *pctxt*, ASN1OBJID \* *pvalue*)**

This function decodes a value of the ASN.1 OBJECT IDENTIFIER type.

**Parameters:**

*pctxt* Pointer to context block structure.

*pvalue* Pointer to value to receive decoded result. The ASN1OBJID structure contains an integer to hold the number of subidentifiers and an array to hold the subidentifier values.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecObjId64 (ASN1CTXT \* *pctxt*, ASN1OID64 \* *pvalue*)**

This function decodes a value of the ASN.1 OBJECT IDENTIFIER type using 64-bit subidentifiers.

**Parameters:**

*pctxt* Pointer to context block structure.

*pvalue* Pointer to value to receive decoded result. The ASN1OID64 structure contains an integer to hold the number of subidentifiers and an array of 64-bit unsigned integers to hold the subidentifier values.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecOctStr (ASN1CTXT \* *pctxt*, ASN1OCTET \* *pvalue*, ASN1UINT \* *pnocts*, ASN1INT *bufsize*)**

This function will decode a variable of the ASN.1 OCTET STRING type into a static memory structure. This function call is generated by ASN1C to decode a sized octet string production.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*pvalue* A pointer to a variable to receive the decoded bit string. This is assumed to be a static array large enough to hold the number of octets specified in the *bufsize* input parameter.

*pnocts* Pointer to an integer value to receive the number of octets.

*bufsize* An integer variable containing the size (in octets) of the sized ASN.1 octet string. An error will occur if the number of octets in the decoded string is larger than this value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecOctStrMemBuf (ASN1MemBuf \* *pMemBuf*, ASN1Const XMLCHAR \* *inpdata*, int *length*, ASN1BOOL *skipWhitespaces*)**

This function decodes a variable of the ASN.1 OCTET STRING type to a memory buffer. The decoded data will be put into the memory buffer starting from the current position and bit offset. After all data is decoded the octet string may be fetched out by call to **xerDecCopyOctStr** or **xerDecCopyDynOctStr** functions.

Usually, this function is used in the 'characters' SAX handler.

**Parameters:**

*pMemBuf* Pointer to the destination memory buffer.

*inpdata* Pointer to a source string to be decoded.  
*length* Length of the source string (in characters).  
*skipWhitespaces* Indicates, could whitespaces be ignored or they are illegal.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecOpenType (ASN1CTXT \* *pctxt*, ASN1OpenType \* *pvalue*)**

This function will decode an ASN.1 open type. This used to be the ASN.1 ANY type, but now is used in a variety of applications requiring an encoding that can be interpreted by a decoder without prior knowledge of the type of the variable.

**Parameters:**

*pctxt* A pointer to a context block structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.  
*pvalue* A pointer to a structure variable to receive the decoded string. The string is stored as an array of unsigned integer characters. Memory is allocated for the string by the `rtMemAlloc` function.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecReal (ASN1CTXT \* *pctxt*, ASN1REAL \* *pvalue*)**

This function will decode a variable of the ASN.1 32-bit character UniversalString type. This includes the UniversalString type.

**Parameters:**

*pctxt* A pointer to a context block structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.  
*pvalue* A pointer to a structure variable to receive the decoded string. The string is stored as an array of unsigned integer characters. Memory is allocated for the string by the `rtMemAlloc` function.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecRelativeOID (ASN1CTXT \* *pctxt*, ASN1OBJID \* *pvalue*)**

This function decodes a value of the ASN.1 RELATIVE-OID type.

**Parameters:**

*pctxt* Pointer to context block structure.  
*pvalue* Pointer to value to receive decoded result. The ASN1OBJID structure contains an integer to hold the number of subidentifiers and an array to hold the subidentifier values.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecUInt (ASN1CTXT \* *pctxt*, ASN1UINT \* *pvalue*)**

This function decodes a variable of the unsigned variant of ASN.1 INTEGER type.

**Parameters:**

*pctxt* Pointer to context block structure.

*pvalue* Pointer to a variable to receive the decoded unsigned integer value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecUInt16 (ASN1CTXT \* *pctxt*, ASN1USINT \* *pvalue*)**

This function decodes a 16-bit variable of the unsigned variant of ASN.1 INTEGER type.

**Parameters:**

*pctxt* Pointer to context block structure.

*pvalue* Pointer to a 16-bit variable to receive the decoded unsigned integer value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecUInt64 (ASN1CTXT \* *pctxt*, ASN1UINT64 \* *pvalue*)**

This function decodes a 64-bit variable of the unsigned variant of ASN.1 INTEGER type.

**Parameters:**

*pctxt* Pointer to context block structure.

*pvalue* Pointer to a 64-bit variable to receive the decoded unsigned integer value. The ASN1UINT64 type is set to the C type 'unsigned \_\_int64', 'unsigned long long' or 'unsigned long' in the `asn1type.h` file (depends on the used platform and the compiler).

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecUInt8 (ASN1CTXT \* *pctxt*, ASN1UINT8 \* *pvalue*)**

This function decodes an 8-bit variable of the unsigned variant of ASN.1 INTEGER type.

**Parameters:**

*pctxt* Pointer to context block structure.

*pvalue* Pointer to an 8-bit variable to receive the decoded unsigned integer value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerDecUnivStr (ASN1CTXT \* *pctxt*, ASN1UniversalString \* *outdata*)**

This function will decode a variable an ASN.1 32-bit character UniversalString type. This includes the UniversalString type.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*outdata* A pointer to a structure variable to receive the decoded string. The string is stored as an array of unsigned integer characters. Memory is allocated for the string by the `rtMemAlloc` function.

**int xerSetDecBufPtr (ASN1CTXT \* *pCtxt*, ASN1ConstOctetPtr *bufaddr*, size\_t *bufsiz*)**

This function is used to set the internal decode buffer pointer within the runtime library decode module. It must be called prior to calling any other compiler generated or runtime library decode functions.

**Parameters:**

*pCtxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*bufaddr* A pointer to a memory buffer containing the ASN.1 message. The pointer must point at the first byte in the message.

*bufsiz* The size of the message that was read. This is used to set an internal message size to check for field length errors. If this size is not known, a zero value can be passed to cause these checks to be bypassed.

## XER C Encode Functions.

---

**Detailed Description**

The XER low-level encode functions handle the XER encoding of primitive ASN.1 data types. Calls to these functions are assembled in the C source code generated by the ASN1C compiler to accomplish the encoding of complex ASN.1 structures. These functions are also directly callable from within a user's application program if the need to accomplish a low level encoding function exists.

The procedure to call a low-level encode function is the same as the procedure to call a compiler generated encode function described above. The `rtInitContext` and `xerSetEncBufPtr` functions must first be called to initialize a context and set a pointer to a buffer into which the variable is to be encoded. A static encode buffer is specified by specifying a pointer to a buffer and buffer size. Setting the buffer address to NULL and buffer size to 0 specifies a dynamic buffer. The encode function is then invoked. The result of the encoding will start at the beginning of the specified buffer, or, if a dynamic buffer was used, it can be obtained by calling `xerGetMsgPtr`. The length of the encoded component is obtained by calling `xerGetMsgLen`.

**Functions**

- int **xerSetEncBufPtr** (ASN1CTXT \**pCtxt*, ASN1OCTET \**bufaddr*, size\_t *bufsiz*, ASN1BOOL *canonical*)
- int **xerEncAscCharStr** (ASN1CTXT \**pctxt*, ASN1ConstCharPtr *value*, ASN1ConstCharPtr *elemName*)
- int **xerEncBase64Str** (ASN1CTXT \**pctxt*, ASN1UINT *nocts*, ASN1ConstOctetPtr *data*, ASN1ConstCharPtr *elemName*)
- int **xerEncBigInt** (ASN1CTXT \**pctxt*, ASN1ConstCharPtr *value*, ASN1ConstCharPtr *elemName*)

- `int xerEncBitStr` (ASN1CTXT \*pctxt, ASN1UINT nbits, ASN1ConstOctetPtr data, ASN1ConstCharPtr elemName, ASN1StrType outputType)
- `int xerEncBoolValue` (ASN1CTXT \*pctxt, ASN1BOOL value)
- `int xerEncBool` (ASN1CTXT \*pctxt, ASN1BOOL value, ASN1ConstCharPtr elemName)
- `int xerEncEndDocument` (ASN1CTXT \*pctxt)
- `int xerEncEndElement` (ASN1CTXT \*pctxt, ASN1ConstCharPtr elemName)
- `int xerEncIndent` (ASN1CTXT \*pctxt)
- `int xerEncInt` (ASN1CTXT \*pctxt, ASN1INT value, ASN1ConstCharPtr elemName)
- `int xerEncInt64` (ASN1CTXT \*pctxt, ASN1INT64 value, ASN1ConstCharPtr elemName)
- `int xerEncNewLine` (ASN1CTXT \*pctxt)
- `int xerEncObjId` (ASN1CTXT \*pctxt, const ASN1OBJID \*pvalue, ASN1ConstCharPtr elemName)
- `int xerEncObjId64` (ASN1CTXT \*pctxt, const ASN1OID64 \*pvalue, ASN1ConstCharPtr elemName)
- `int xerEncRelativeOID` (ASN1CTXT \*pctxt, const ASN1OBJID \*pvalue, ASN1ConstCharPtr elemName)
- `int xerEncOctStr` (ASN1CTXT \*pctxt, ASN1UINT nocts, ASN1ConstOctetPtr data, ASN1ConstCharPtr elemName)
- `int xerEncReal` (ASN1CTXT \*pctxt, ASN1REAL value, ASN1ConstCharPtr elemName)
- `int xerEncStartDocument` (ASN1CTXT \*pctxt)
- `int xerEncStartElement` (ASN1CTXT \*pctxt, ASN1ConstCharPtr elemName, ASN1ConstCharPtr attributes)
- `int xerEncEmptyElement` (ASN1CTXT \*pctxt, ASN1ConstCharPtr elemName, ASN1ConstCharPtr attributes)
- `int xerEncNamedValue` (ASN1CTXT \*pctxt, ASN1ConstCharPtr value, ASN1ConstCharPtr elemName, ASN1ConstCharPtr attributes)
- `int xerEncUInt` (ASN1CTXT \*pctxt, ASN1UINT value, ASN1ConstCharPtr elemName)
- `int xerEncUInt64` (ASN1CTXT \*pctxt, ASN1UINT64 value, ASN1ConstCharPtr elemName)
- `int xerEncBMPStr` (ASN1CTXT \*pctxt, const ASN1BMPString \*value, ASN1ConstCharPtr elemName)
- `int xerEncUnivStr` (ASN1CTXT \*pctxt, const ASN1UniversalString \*value, ASN1ConstCharPtr elemName)
- `int xerEncUniCharData` (ASN1CTXT \*pctxt, ASN1Const16BitCharPtr value, ASN1UINT nchars)
- `int xerEncUniCharStr` (ASN1CTXT \*pctxt, ASN116BITCHAR \*value, ASN1ConstCharPtr elemName)
- `int xerEncOpenType` (ASN1CTXT \*pctxt, ASN1UINT nocts, ASN1ConstOctetPtr data, ASN1ConstCharPtr elemName)
- `int xerEncNull` (ASN1CTXT \*pctxt, ASN1ConstCharPtr elemName)
- `int xerEncXmlCharData` (ASN1CTXT \*pctxt, ASN1Const XMLCHAR \*pvalue, int length)

## Function Documentation

**`int xerEncAscCharStr` (ASN1CTXT \* *pctxt*, ASN1ConstCharPtr *value*, ASN1ConstCharPtr *elemName*)**

This function encodes a variable one of the ASN.1 character string types that are based on 8-bit character sets. This includes IA5String, VisibleString, PrintableString, and UTF8String, and NumericString.

### Parameters:

- pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
- value* A pointer to a null-terminated C character string to be encoded.
- elemName* This argument specifies the name of the element that is wrapped around the encoded value. If an empty string is passed (""), no element tag is added to the encoded value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerEncBase64Str (ASN1CTXT \* *pctxt*, ASN1UINT *nocts*, ASN1ConstOctetPtr *data*, ASN1ConstCharPtr *elemName*)**

This function encodes a variable of the ASN.1 OCTET STRING type using Base64 encoding.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*nocts* The number of octets (bytes) within the OCTET STRING to be encoded.

*data* A pointer to an OCTET STRING containing the octet data to be encoded.

*elemName* This argument specifies the name of the element that is wrapped around the encoded value. If the name is null, the default name for this type (<OCTET\_STRING>) is added. If an empty string is passed (""), no element tag is added to the encoded value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerEncBigInt (ASN1CTXT \* *pctxt*, ASN1ConstCharPtr *value*, ASN1ConstCharPtr *elemName*)**

This function encodes a variable of the ASN.1 INTEGER type. In this case, the integer is assumed to be of a larger size than can fit in a C or C++ long type (normally 32 or 64 bits). For example, parameters used to calculate security values are typically larger than these sizes.

Items of this type are stored in character string constant variables. They can be represented as decimal strings (with no prefixes), as hexadecimal strings starting with a "0x" prefix, as octal strings starting with a "0o" prefix or as binary strings starting with a "0b" prefix. Other radixes are currently not supported.

**Parameters:**

*pctxt* Pointer to context block structure.

*value* A pointer to a character string containing the value to be encoded.

*elemName* This argument specifies the name of the element that is wrapped around the encoded value. If the name is null, the default name for this type (<INTEGER>) is added. If an empty string is passed (""), no element tag is added to the encoded value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerEncBitStr (ASN1CTXT \* *pctxt*, ASN1UINT *nbits*, ASN1ConstOctetPtr *data*, ASN1ConstCharPtr *elemName*, ASN1StrType *outputType*)**

This function encodes a variable of the ASN.1 BIT STRING type.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*nbits* The number of bits within the bit string to be encoded.

*data* A pointer to an OCTET string containing the bit data to be encoded. This string contains bytes having the actual bit settings as they are to be encoded in the message.  
*elemName* This argument specifies the name of the element that is wrapped around the encoded value. If the name is null, the default name for this type (<BIT\_STRING>) is added. If an empty string is passed (""), no element tag is added to the encoded value.  
*outputType* An enumerated type whose value is set to either 'ASN1BIN' (for binary format) or 'ASN1HEX' (for hexadecimal format).

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerEncBMPStr (ASN1CTXT \* *pctxt*, const ASN1BMPString \* *value*, ASN1ConstCharPtr *elemName*)**

This function encodes a variable of the BMPString ASN.1 character string type.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*value* A pointer to a structure representing a 16-bit character string to be encoded. This structure contains a character count element and a pointer to an array of 16-bit character elements represented as 16-bit short integers.

*elemName* A pointer to a name of an element.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerEncBool (ASN1CTXT \* *pctxt*, ASN1BOOL *value*, ASN1ConstCharPtr *elemName*)**

This function encodes a variable of the ASN.1 BOOLEAN type.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*value* A BOOLEAN value to be encoded. A BOOLEAN is defined as a single octet whose value is 0 for False and any other value for TRUE.

*elemName* This argument specifies the name of the element that is wrapped around the encoded value. If the name is null, the default name for this type (<BOOLEAN>) is added. If an empty string is passed (""), no element tag is added to the encoded value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerEncEmptyElement (ASN1CTXT \* *pctxt*, ASN1ConstCharPtr *elemName*, ASN1ConstCharPtr *attributes*)**

This function encodes an empty element, such as <element/>.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*elemName* A pointer to the name of the element.

*attributes* A pointer to the attributes of the element.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerEncEndDocument (ASN1CTXT \* *pctxt*)**

This function should be called at the end of the document's encoding.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerEncEndElement (ASN1CTXT \* *pctxt*, ASN1ConstCharPtr *elemName*)**

This function encodes the ending tag of the XML element, such as </element>.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*elemName* A pointer to the name of the element.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerEncIndent (ASN1CTXT \* *pctxt*)****Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

**int xerEncInt (ASN1CTXT \* *pctxt*, ASN1INT *value*, ASN1ConstCharPtr *elemName*)**

This function encodes a variable of the ASN.1 INTEGER type.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*value* An INTEGER value to be encoded. The ASN1INT type is set to the C type 'int' in the *asn1type.h* file. This is assumed to represent a 32-bit integer value.

*elemName* This argument specifies the name of the element that is wrapped around the encoded value. If the name is null, the default name for this type (<INTEGER>) is added. If an empty string is passed (""), no element tag is added to the encoded value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

### **int xerEncInt64 (ASN1CTXT \* *pctxt*, ASN1INT64 *value*, ASN1ConstCharPtr *elemName*)**

This function encodes a 64-bit variable of the ASN.1 INTEGER type.

#### **Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*value* A 64-bit INTEGER value to be encoded. The ASN1INT64 type is set to the C type '`__int64`', 'long long' or 'long' in the `asn1type.h` file (depends on the used platform and the compiler). This is assumed to represent a 64-bit integer value.

*elemName* This argument specifies the name of the element that is wrapped around the encoded value. If the name is null, the default name for this type (<INTEGER>) is added. If an empty string is passed (""), no element tag is added to the encoded value.

#### **Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

### **int xerEncNamedValue (ASN1CTXT \* *pctxt*, ASN1ConstCharPtr *value*, ASN1ConstCharPtr *elemName*, ASN1ConstCharPtr *attributes*)**

This function encodes a named value, for example an enumerated value, such as <element><value/></element>.

#### **Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*value* A pointer to a value string

*elemName* A pointer to the name of the element.

*attributes* A pointer to the attributes of the element.

#### **Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

### **int xerEncNewLine (ASN1CTXT \* *pctxt*)**

This function inserts a new line symbol into the XML document.

#### **Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

#### **Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

### **int xerEncNull (ASN1CTXT \* *pctxt*, ASN1ConstCharPtr *elemName*)**

This function encodes an ASN.1 NULL placeholder. In XER the NULL value is represented as an empty element, such as <null/>.

#### **Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*elemName* This argument specifies the name of the element that is wrapped around the encoded value. If a null or empty string is passed (""), no element tag is added to the encoded value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerEncObjId (ASN1CTXT \* *pctxt*, const ASN1OBJID \* *pvalue*, ASN1ConstCharPtr *elemName*)**

This function encodes a variable of the ASN.1 OBJECT IDENTIFIER type.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*pvalue* A pointer to an object identifier structure. This structure contains an integer to hold the number of subidentifiers in the object and an array to hold the subidentifier values.

*elemName* This argument specifies the name of the element that is wrapped around the encoded value. If the name is null, the default name for this type (<OBJECT\_IDENTIFIER>) is added. If an empty string is passed (""), no element tag is added to the encoded value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerEncObjId64 (ASN1CTXT \* *pctxt*, const ASN1OID64 \* *pvalue*, ASN1ConstCharPtr *elemName*)**

This function encodes a variable of the ASN.1 OBJECT IDENTIFIER type using 64-bit subidentifiers.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*pvalue* A pointer to a 64-bit object identifier structure. This structure contains an integer to hold the number of subidentifiers in the object and an array of 64-bit unsigned integers to hold the subidentifier values.

*elemName* This argument specifies the name of the element that is wrapped around the encoded value. If the name is null, the default name for this type (<OBJECT\_IDENTIFIER>) is added. If an empty string is passed (""), no element tag is added to the encoded value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerEncOctStr (ASN1CTXT \* *pctxt*, ASN1UINT *nocts*, ASN1ConstOctetPtr *data*, ASN1ConstCharPtr *elemName*)**

This function encodes a variable of the ASN.1 OCTET STRING type.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*nocts* The number of octets (bytes) within the OCTET STRING to be encoded.

*data* A pointer to an OCTET STRING containing the octet data to be encoded.

*elemName* This argument specifies the name of the element that is wrapped around the encoded value. If the name is null, the default name for this type (<OCTET\_STRING>) is added. If an empty string is passed (""), no element tag is added to the encoded value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerEncOpenType (ASN1CTXT \* *pctxt*, ASN1UINT *nocts*, ASN1ConstOctetPtr *data*, ASN1ConstCharPtr *elemName*)**

This function encodes a variable of the old (pre-1994) ASN.1 ANY type or other elements defined in the later standards to be Open Types (for example, a variable type declaration in a CLASS construct as defined in X.681). A variable of this is considered to be a previously encoded ASN.1 message component.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*nocts* The number of octets (bytes) within the OCTET STRING to be encoded.

*data* A pointer to an OCTET STRING containing an encoded ASN.1 message component.

*elemName* This argument specifies the name of the element that is wrapped around the encoded value. If the name is null, the default name for this type (<REAL>) is added. If an empty string is passed (""), no element tag is added to the encoded value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerEncReal (ASN1CTXT \* *pctxt*, ASN1REAL *value*, ASN1ConstCharPtr *elemName*)**

This function encodes a variable of the REAL data type. This function provides support for the plus-infinity and the minus-infinity special real values. Use the `rtGetPlusInfinity` or `rtGetMinusInfinity` functions to get these special values.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*value* A value to be encoded. Special real values plus and minus infinity are encoded by using the `rtGetPlusInfinity` and `rtGetMinusInfinity` functions to set the real value to be encoded.

*elemName* A pointer to the name of the element.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerEncRelativeOID (ASN1CTXT \* *pctxt*, const ASN1OBJID \* *pvalue*, ASN1ConstCharPtr *elemName*)**

This function encodes a variable of the ASN.1 RELATIVE-OID type.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*pvalue* A pointer to an object identifier structure. This structure contains an integer to hold the number of subidentifiers in the object and an array to hold the subidentifier values.  
*elemName* This argument specifies the name of the element that is wrapped around the encoded value. If the name is null, the default name for this type (<RELATIVE\_OID>) is added. If an empty string is passed (""), no element tag is added to the encoded value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerEncStartDocument (ASN1CTXT \* *pctxt*)**

This function encodes the starting record of the XML document (such as the <?xml version = "1.0" encoding = "UTF-8"?>). This function should be called prior to encoding any other fields in the document. After all elements in the document are encoded, xerEncEndDocument should be called.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerEncStartElement (ASN1CTXT \* *pctxt*, ASN1ConstCharPtr *elemName*, ASN1ConstCharPtr *attributes*)**

This function encodes the string tag of the XML element, such as <element>. After the element's data is encoded, the xerEncEndElement function should be called.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*elemName* A pointer to the name of the element.

*attributes* A pointer to the attributes of the element.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerEncUInt (ASN1CTXT \* *pctxt*, ASN1UINT *value*, ASN1ConstCharPtr *elemName*)**

This function encodes an unsigned variable of the ASN.1 INTEGER type.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*value* An unsigned INTEGER value to be encoded. The ASN1UINT type is set to the C type 'unsigned int' in the asn1type.h file. This is assumed to represent a 32-bit integer value.

*elemName* This argument specifies the name of the element that is wrapped around the encoded value. If the name is null, the default name for this type (<INTEGER>) is added. If an empty string is passed (""), no element tag is added to the encoded value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerEncUInt64 (ASN1CTXT \* *pctxt*, ASN1UINT64 *value*, ASN1ConstCharPtr *elemName*)**

This function encodes an unsigned 64-bit variable of the ASN.1 INTEGER type.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.  
*value* An unsigned 64-bit INTEGER value to be encoded. The ASN1UINT64 type is set to the C type 'unsigned \_\_int64', 'unsigned long long' or 'unsigned long' in the asn1type.h file (depends on the used platform and the compiler). This is assumed to represent an unsigned 64-bit integer value.  
*elemName* This argument specifies the name of the element that is wrapped around the encoded value. If the name is null, the default name for this type (<INTEGER>) is added. If an empty string is passed (""), no element tag is added to the encoded value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerEncUniCharData (ASN1CTXT \* *pctxt*, ASN1Const16BitCharPtr *value*, ASN1UINT *nchars*)**

This function encodes a variable of the ASN.1 character string types that are based on 16-bit character sets and are represented as null-terminated Unicode string. This includes IA5String, VisibleString, PrintableString, NumericString, and BMPString.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.  
*value* A pointer to a null-terminated 16-bit string to be encoded.  
*nchars* The number of characters to be encoded.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerEncUniCharStr (ASN1CTXT \* *pctxt*, ASN116BITCHAR \* *value*, ASN1ConstCharPtr *elemName*)**

This function encodes a variable one of the ASN.1 character string types that are based on 16-bit character sets and are represented as null-terminated Unicode strings. This includes IA5String, VisibleString, PrintableString, NumericString, and BMPString.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.  
*value* A pointer to a null-terminated 16-bit character string to be encoded.  
*elemName* This argument specifies the name of the element that is wrapped around the encoded value. If an empty string is passed (""), no element tag is added to the encoded value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerEncUnivStr (ASN1CTXT \* *pctxt*, const ASN1UniversalString \* *value*, ASN1ConstCharPtr *elemName*)**

This function encodes a variable of the ASN.1 Universal character string. This differs from the encode routines for the character strings previously described in that the Universal string type is based on 32-bit characters. A 32-bit character string is modeled using an array of unsigned integers.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*value* A pointer to a structure representing a 16-bit character string to be encoded. This structure contains a character count element and a pointer to an array of 32-bit character elements represented as 32-bit short integers.

*elemName* A pointer to a name of an element.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xerSetEncBufPtr (ASN1CTXT \* *pCtxt*, ASN1OCTET \* *bufaddr*, size\_t *bufsiz*, ASN1BOOL *canonical*)**

This function is used to set the internal buffer within the runtime library encode module. It must be called prior to calling any other generated or runtime library encode functions.

**Parameters:**

*pCtxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*bufaddr* A pointer to a memory buffer containing the ASN.1 message. The pointer must point at the first byte in the message.

*bufsiz* The size of the message that was read. This is used to set an internal message size to check for field length errors. If this size is not known, a zero value can be passed to cause these checks to be bypassed.

*canonical* TRUE, if canonical XML encoding rules (CXER) should be used. Otherwise FALSE.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

## XER C Utility Functions.

---

### Detailed Description

The XER utility functions are common routines used by both XER encode and decode functions.

### Functions

- ASN1BOOL **xerCmpText** (ASN1Const XMLCHAR \*text1, ASN1ConstCharPtr text2)
- int **xerCopyText** (ASN1CTXT \*pctxt, ASN1ConstCharPtr text)
- int **xerTextLength** (ASN1Const XMLCHAR \*text)

- ASN1ConstCharPtr **xerTextToCStr** (ASN1CTXT \*pctxt, ASN1Const XMLCHAR \*text)
  - size\_t **xerGetMsgLen** (ASN1CTXT \*pctxt)
  - ASN1OCTET \* **xerGetMsgPtr** (ASN1CTXT \*pctxt)
  - int **xerGetElemIdx** (ASN1Const XMLCHAR \*elemName, XerElemInfo \*pElemInfo, int numElems)
  - int **xerGetSeqElemIdx** (ASN1Const XMLCHAR \*elemName, XerElemInfo \*pElemInfo, int numElems, int startIndex)
  - int **xerFinalizeMemBuf** (ASN1MemBuf \*pMemBuf)
  - int **xerGetLibVersion** ()
  - const char \* **xerGetLibInfo** ()
- 

## Function Documentation

### ASN1BOOL xerCmpText (ASN1Const XMLCHAR \* text1, ASN1ConstCharPtr text2)

This function is used to compare two strings: the first is represented as a 16-bit character null-terminated string and the second is represented as an 8-bit standard null-terminated string.

#### Parameters:

*text1* A pointer to a 16-bit character null-terminated string.

*text2* A pointer to an 8-bit character null-terminated string.

#### Returns:

The result of the comparison: TRUE, if strings match, otherwise FALSE.

### const char\* xerGetLibInfo ()

Returns information string describing the library. The string contains name of library, its version and flags used for building the library.

#### Returns:

Information string

### int xerGetLibVersion ()

Returns numeric version of run-time library. The format of version is as follows: MmP, where: M - major version number; m - minor version number; p - patch release number. For example, the value 581 means the version 5.81.

#### Returns:

Version of run-time library in numeric format.

### size\_t xerGetMsgLen (ASN1CTXT \* pctxt)

This function is used to get the encoded message length.

#### Parameters:

*pctxt* A pointer to a context structure.

#### Returns:

The length of a message in the buffer.

### ASN1OCTET\* xerGetMsgPtr (ASN1CTXT \* pctxt)

This function is used to obtain a pointer to the start of an encoded message. This function is called after a compiler generated encode function to get the pointer to the start of the encoded message. It is normally used when dynamic encoding is specified because the message pointer is not known until the encoding is complete. If static encoding is used, the message

starts at the beginning of the specified buffer and the `xerGetMsgLen` function can be used to obtain the length of the message.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

**Returns:**

A pointer to the beginning of the encoded message.

**int xerTextLength (ASN1Const XMLCHAR \* text)**

This function returns the length of a 16-bit character null-terminated string.

**Parameters:**

*text* A pointer to a 16-bit character null-terminated string.

**ASN1ConstCharPtr xerTextToCStr (ASN1CTXT \* pctxt, ASN1Const XMLCHAR \* text)**

This function converts a 16-bit character string to a standard null-terminated C-string. This function will allocate dynamic memory to store the decoded result.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*text* A pointer to a 16-bit character null-terminated string.

**Returns:**

The resulting C string. The memory is allocated by using `rtMemAlloc` function.

## XML C Decode Functions.

---

### Detailed Description

XML runtime library decode functions handle the decoding of the primitive ASN.1 data types and length variables. Calls to these functions are assembled in the C source code generated by the ASN1C compiler to decode complex ASN.1 structures. These functions are also directly callable from within a user's application program if the need to decode a primitive data item exists.

### Functions

- `int xmlDecBitStr (ASN1CTXT *pctxt, ASN1OCTET *pvalue, ASN1UINT *pnbits, ASN1INT bufsize)`
- `int xmlDecBool (ASN1CTXT *pctxt, ASN1BOOL *pvalue)`
- `int xmlDecDynBitStr (ASN1CTXT *pctxt, ASN1DynBitStr *pvalue)`
- `int xmlDecDynNamedBitStr (ASN1CTXT *pctxt, ASN1DynBitStr *pvalue, ASN1Const XmlNamedBitsDict *pBitDict)`
- `int xmlDecDynOctStr (ASN1CTXT *pctxt, ASN1DynOctStr *pvalue)`
- `int xmlDecGeneralizedTime (ASN1CTXT *pctxt, ASN1ConstCharPtr *outdata)`
- `int xmlDecNamedBitStr (ASN1MemBuf *pMemBuf, ASN1OCTET *pData, int dataSize, ASN1UINT *pNumbits, ASN1Const XmlNamedBitsDict *pBitDict, ASN1Const XMLCHAR *chars, int length)`

- int **xmlDecOctStr** (ASN1CTXT \*pctxt, ASN1OCTET \*pvalue, ASN1UINT \*pnocts, ASN1INT bufsize)
  - int **xmlDecReal** (ASN1CTXT \*pctxt, ASN1REAL \*pvalue)
  - int **xmlDecUTCTime** (ASN1CTXT \*pctxt, ASN1ConstCharPtr \*outdata)
- 

## Function Documentation

**int xmlDecBitStr (ASN1CTXT \* pctxt, ASN1OCTET \* pvalue, ASN1UINT \* pnbits, ASN1INT bufsize)**

This function will decode a variable of the ASN.1 BIT STRING type into a static memory structure. This function call is generated by ASN1C to decode a sized bit production.

### Parameters:

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*pvalue* A pointer to a variable to receive the decoded bit string. This is assumed to be a static array large enough to hold the number of octets specified in the bufsize input parameter.

*pnbits* A pointer to an integer value to receive the decoded number of bits.

*bufsize* An integer variable containing the size (in octets) of the sized ASN.1 bit string. An error will occur if the number of octets in the decoded string is larger than this value.

### Returns:

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xmlDecBool (ASN1CTXT \* pctxt, ASN1BOOL \* pvalue)**

This function decodes a variable of the ASN.1 BOOLEAN type.

### Parameters:

*pctxt* Pointer to context block structure.

*pvalue* Pointer to a variable to receive the decoded BOOLEAN value.

### Returns:

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xmlDecDynBitStr (ASN1CTXT \* pctxt, ASN1DynBitStr \* pvalue)**

This function will decode a variable of the ASN.1 BIT STRING type. This function will allocate dynamic memory to store the decoded result.

### Parameters:

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*pvalue* A pointer to a dynamic bit string structure to receive the decoded bit string. Dynamic memory is allocated to hold the string using the rtMemAlloc function.

### Returns:

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xmlDecDynNamedBitStr (ASN1CTXT \* *pctxt*, ASN1DynBitStr \* *pvalue*, ASN1Const XmlNamedBitsDict \* *pBitDict*)**

This function will decode a list of identifiers into the array of octets. Identifiers should represent named bits value for BIT STRING. This function call is generated by ASN1C to decode a named bit production.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*pvalue* A pointer to a dynamic bit string structure to receive the decoded bit string. Dynamic memory is allocated to hold the string using the rtMemAlloc function.

*pBitDict* Bits' dictionary to be used to decode each bit. It is an array of name-value pairs, represented by an array of XmlNamedBitsDict structure.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xmlDecDynOctStr (ASN1CTXT \* *pctxt*, ASN1DynOctStr \* *pvalue*)**

This function will decode a variable of the ASN.1 OCTET STRING type. This function will allocate dynamic memory to store the decoded result.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*pvalue* A pointer to a dynamic bit string structure to receive the decoded bit string. Dynamic memory is allocated to hold the string using the rtMemAlloc function.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xmlDecGeneralizedTime (ASN1CTXT \* *pctxt*, ASN1ConstCharPtr \* *outdata*)**

This function will decode a variable of the ASN.1 GeneralizedTime type. This function performs conversion between XML format of dateTime into the ASN.1 format. This function will allocate dynamic memory to store the result.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*outdata* A pointer to a character string pointer variable to receive the decoded string. The string as stored as a standard null-terminated C string. Memory is allocated for the string by the rtMemAlloc function. It will contain time in ASN.1 format.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xmlDecNamedBitStr (ASN1MemBuf \* *pMemBuf*, ASN1OCTET \* *pData*, int *dataSize*, ASN1UINT \* *pNumbits*, ASN1Const XmlNamedBitsDict \* *pBitDict*, ASN1Const XMLCHAR \* *chars*, int *length*)**

This function will decode a list of identifiers into the array of octets. Identifiers should represent named bits value for BIT STRING. This function call is generated by ASN1C to decode a named bit production.

**Parameters:**

*pMemBuf* Pointer to the destination memory buffer.

*pData* A pointer to a variable to receive the decoded bit string. This is assumed to be a static array large enough to hold the number of octets specified in the bufsize input parameter.

*dataSize* An integer variable containing the size (in octets) of the sized ASN.1 bit string. An error will occur if the number of octets in the decoded string is larger than this value.

*pNumbits* A pointer to an integer value to receive the decoded number of bits.

*pBitDict* Bits' dictionary to be used to decode each bit. It is an array of name-value pairs, represented by an array of XmlNamedBitsDict structure.

*chars* XML data to be appended to memory buffer before parsing. Could be NULL, if it is final call to this function.

*length* Number of characters in XML data to be appended to memory buffer before parsing. Could be 0, if it is final call to this function.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xmlDecOctStr (ASN1CTXT \* *pctxt*, ASN1OCTET \* *pvalue*, ASN1UINT \* *pnocts*, ASN1INT *bufsize*)**

This function will decode a variable of the ASN.1 OCTET STRING type into a static memory structure. This function call is generated by ASN1C to decode a sized octet string production.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*pvalue* A pointer to a variable to receive the decoded bit string. This is assumed to be a static array large enough to hold the number of octets specified in the bufsize input parameter.

*pnocts* Pointer to an integer value to receive the number of octets.

*bufsize* An integer variable containing the size (in octets) of the sized ASN.1 octet string. An error will occur if the number of octets in the decoded string is larger than this value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xmlDecReal (ASN1CTXT \* *pctxt*, ASN1REAL \* *pvalue*)**

This function will decode a variable of the ASN.1 32-bit character UniversalString type. This includes the UniversalString type.

**Parameters:**

*pctxt* A pointer to a context block structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*pvalue* A pointer to a structure variable to receive the decoded string. The string is stored as an array of unsigned integer characters. Memory is allocated for the string by the `rtMemAlloc` function.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xmlDecUTCTime (ASN1CTXT \* *pctxt*, ASN1ConstCharPtr \* *outdata*)**

This function will decode a variable of the ASN.1 UTCTime type. This function performs conversion between XML format of `dateTime` into the ASN.1 format. This function will allocate dynamic memory to store the result.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*outdata* A pointer to a character string pointer variable to receive the decoded string. The string is stored as a standard null-terminated C string. Memory is allocated for the string by the `rtMemAlloc` function. It will contain time in ASN.1 format.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

## XML C Encode Functions.

---

### Detailed Description

The XML low-level encode functions handle the XML encoding of primitive ASN.1 data types. In most cases XER encoding functions can be used for encoding XML. But there are some differences between XER and XML encodings, described in X.693 and X.694. These functions are very similar to XER ones, and only functions provide different encoding are added.

### Functions

- int **xmlEncBitStr** (ASN1CTXT \**pctxt*, XmlNamedBitsDict \**namedbits*, ASN1UINT *noofnamedbits*, ASN1UINT *nbits*, ASN1ConstOctetPtr *data*, ASN1ConstCharPtr *elemName*, ASN1StrType *outputType*)
- int **xmlEncBoolValue** (ASN1CTXT \**pctxt*, ASN1BOOL *value*)
- int **xmlEncBool** (ASN1CTXT \**pctxt*, ASN1BOOL *value*, ASN1ConstCharPtr *elemName*)
- int **xmlEncEnum** (ASN1CTXT \**pctxt*, ASN1ConstCharPtr *value*)
- int **xmlEncGeneralizedTime** (ASN1CTXT \**pctxt*, ASN1ConstCharPtr *value*, ASN1ConstCharPtr *elemName*)
- int **xmlEncNamedValue** (ASN1CTXT \**pctxt*, ASN1ConstCharPtr *value*, ASN1ConstCharPtr *elemName*, ASN1ConstCharPtr *attributes*)
- int **xerEncOpenTypeExt** (ASN1CTXT \**pctxt*, Asn1RTDList \**pElemList*)
- int **xmlEncReal** (ASN1CTXT \**pctxt*, ASN1REAL *value*, ASN1ConstCharPtr *elemName*)
- int **xmlEncUTCTime** (ASN1CTXT \**pctxt*, ASN1ConstCharPtr *value*, ASN1ConstCharPtr *elemName*)

---

## Function Documentation

**int xmlEncBitStr (ASN1CTXT \* *pctxt*, XmlNamedBitsDict \* *namedbits*, ASN1UINT *noofnamedbits*, ASN1UINT *nbits*, ASN1ConstOctetPtr *data*, ASN1ConstCharPtr *elemName*, ASN1StrType *outputType*)**

This function encodes a variable of the ASN.1 BIT STRING type.

### Parameters:

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*namedbits* Bits' dictionary to be used to encode each bit. It is an array of name-value pairs, represented by an array of XmlNamedBitsDict structure.

*noofnamedbits* Number of named bits in bits' dictionary.

*nbits* The number of bits within the bit string to be encoded.

*data* A pointer to an OCTET string containing the bit data to be encoded. This string contains bytes having the actual bit settings as they are to be encoded in the message.

*elemName* This argument specifies the name of the element that is wrapped around the encoded value. If the name is null, the default name for this type (<BIT\_STRING>) is added. If an empty string is passed (""), no element tag is added to the encoded value.

*outputType* An enumerated type whose value is set to either 'ASN1BIN' (for binary format) or 'ASN1HEX' (for hexadecimal format).

### Returns:

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xmlEncBool (ASN1CTXT \* *pctxt*, ASN1BOOL *value*, ASN1ConstCharPtr *elemName*)**

This function encodes a variable of the ASN.1 BOOLEAN type.

### Parameters:

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*value* A BOOLEAN value to be encoded. A BOOLEAN is defined as a single octet whose value is 0 for False and any other value for TRUE.

*elemName* This argument specifies the name of the element that is wrapped around the encoded value. If the name is null, the default name for this type (<BOOLEAN>) is added. If an empty string is passed (""), no element tag is added to the encoded value.

### Returns:

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xmlEncEnum (ASN1CTXT \* *pctxt*, ASN1ConstCharPtr *value*)**

This function encodes an enumeration value for the ASN.1 ENUMERATED and INTEGER types.

### Parameters:

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*value* An enumeration identifier to be encoded.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xmlEncGeneralizedTime (ASN1CTXT \* *pctxt*, ASN1ConstCharPtr *value*, ASN1ConstCharPtr *elemName*)**

This function encodes a variable of the ASN.1 GeneralizedTime type. It performs conversion from ASN.1 time format into the XML dateTime format.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*value* A pointer to a null-terminated C character string to be encoded. It should contain GeneralizedTime in ASN.1 format.

*elemName* This argument specifies the name of the element that is wrapped around the encoded value. If an empty string is passed (""), no element tag is added to the encoded value.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xmlEncNamedValue (ASN1CTXT \* *pctxt*, ASN1ConstCharPtr *value*, ASN1ConstCharPtr *elemName*, ASN1ConstCharPtr *attributes*)**

This function encodes a named value, for example an enumerated value, such as <element>value</element>.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*value* A pointer to a value string

*elemName* A pointer to the name of the element.

*attributes* A pointer to the attributes of the element.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xmlEncReal (ASN1CTXT \* *pctxt*, ASN1REAL *value*, ASN1ConstCharPtr *elemName*)**

This function encodes a variable of the REAL data type. This function provides support for the plus-infinity, minus-infinity, NaN and minus zero special real values. Use the rtGetPlusInfinity, rtGetMinusInfinity, rtGetNaN and rtGetMinusZero functions to get these special values.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*value* A value to be encoded. Special real values are encoded by using the appropriate functions to set the real value to be encoded.

*elemName* A pointer to the name of the element.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int xmlEncUTCTime (ASN1CTXT \* *pctxt*, ASN1ConstCharPtr *value*, ASN1ConstCharPtr *elemName*)**

This function encodes a variable of the ASN.1 UTCTime type. It performs conversion from ASN.1 time format into the XML dateTime format.

**Parameters:**

*pctxt* A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.

*value* A pointer to a null-terminated C character string to be encoded. It should contain UTCTime in ASN.1 format.

*elemName* This argument specifies the name of the element that is wrapped around the encoded value. If an empty string is passed (""), no element tag is added to the encoded value.

**Returns:**

Completion status of operation:

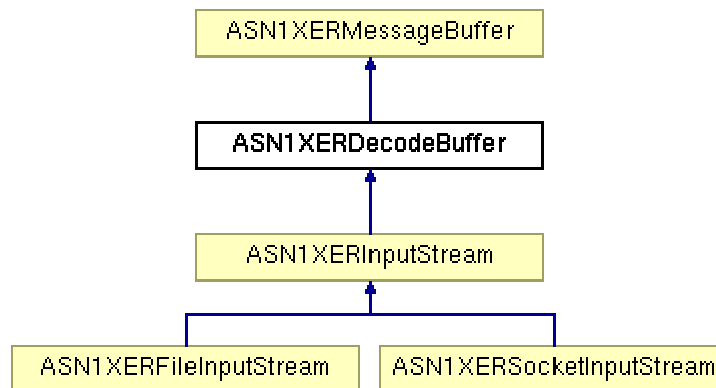
- 0 (ASN\_OK) = success,
- negative return value is error.

# ASN1C XER Runtime Class Documentation

## ASN1XERDecodeBuffer Class Reference

```
#include <asn1XerCppTypes.h>
```

Inheritance diagram for ASN1XERDecodeBuffer:



---

### Detailed Description

The ASN1XERDecodeBuffer class is derived from the **ASN1XERMessageBuffer** base class. It contains variables and methods specific to decoding ASN.1 XER messages. It is used to manage the input buffer containing an ASN.1 message to be decoded.

Note that the XER decode buffer object does not take a message buffer argument because buffer management is handled by the XML parser.

### Public Member Functions

- **ASN1XERDecodeBuffer** (const char \*xmlFile)
- **ASN1XERDecodeBuffer** (const ASN1OCTET \*msgbuf, size\_t numocts, ASN1BOOL openType=FALSE)
- **ASN1XERDecodeBuffer** (OSInputStream &inputStream, ASN1BOOL openType=FALSE)
- int **decodeXML** (OSXMLReader \*pReader)
- const char \* **getXmlFileName** ()
- int **initBuffer** (ASN1MemBuf &membuf)
- int **initBuffer** (char \*str)
- int **initBuffer** (ASN116BITCHAR \*str)
- virtual ASN1BOOL **isA** (Type bufferType)
- virtual void **throwSAXException** (int stat, const char \*file=0, int line=0)
- void **setOpenType** ()

### Protected Types

- enum { **INPUT\_FILE**, **INPUT\_STREAM**, **INPUT\_STREAM\_ATTACHED**, **INPUT\_MEMORY** }

## Protected Attributes

- union {
  - const char \* **fileName**
  - OSCInputStream \* **pInputStream**
  - struct {
  - const ASN1OCTET \* **pMemBuf**
  - int **bufSize**
  - } **memBuf**
  - } **mInput**
  - enum ASN1XERDecodeBuffer:: { ... } **mInputId**
- 

## Constructor & Destructor Documentation

### ASN1XERDecodeBuffer::ASN1XERDecodeBuffer (const char \* *xmlFile*)

The ASN1XERDecodeBuffer class has three overloaded constructors. This version of the ASN1XERDecodeBuffer constructor takes a name of a file that contains the XML data to be decoded, and constructs a buffer. Use getStatus() method to determine has error occurred during the initialization or not.

#### Parameters:

*xmlFile* A pointer to name of file to be decoded.

### ASN1XERDecodeBuffer::ASN1XERDecodeBuffer (const ASN1OCTET \* *msgbuf*, size\_t *numocts*, ASN1BOOL *openType* = FALSE)

This version of the ASN1XERDecodeBuffer constructor takes parameters describing the message to be decoded, and constructs a buffer describing an encoded ASN.1 message. Use getStatus() method to determine has error occurred during the initialization or not.

#### Parameters:

*msgbuf* A pointer to a buffer containing an encoded ASN.1 message.

*numocts* Size of the message buffer.

*openType* Open Type decoding indicator. Set this argument as TRUE to decode an open type.

### ASN1XERDecodeBuffer::ASN1XERDecodeBuffer (OSCInputStream & *inputStream*, ASN1BOOL *openType* = FALSE)

This version of the ASN1XERDecodeBuffer constructor takes a reference to the OSCInputStream object. Use getStatus() method to determine has error occurred during the initialization or not.

#### Parameters:

*inputStream* reference to the OSCInputStream object

*openType* Open Type decoding indicator. Set this argument as TRUE to decode an open type.

---

## Member Function Documentation

### int ASN1XERDecodeBuffer::decodeXML (OSXMLReader \* *pReader*)

This method decodes an XML message associated with this buffer.

**Returns:**

stat Status of the operation. Possible values are ASN\_OK if successful or one of the negative error status codes defined in Appendix A of the C/C++ runtime Common Functions Reference Manual.

**Parameters:**

*pReader* Pointer to OSXMLReader object.

**const char\* ASN1XERDecodeBuffer::getXmlFileName () [inline]**

This method returns the name of the XML file that is associated with the current buffer.

**Returns:**

Name of the XML file that is associated with the current buffer.

**int ASN1XERDecodeBuffer::initBuffer (ASN1MemBuf & *membuf*) [inline]**

This method reinitializes the decode buffer pointer to allow new data to be decoded. This makes it possible to reuse one message buffer object in a loop to decode multiple data messages.

**Returns:**

Status of the operation. Possible values are ASN\_OK if successful or one of the negative error status codes defined in Appendix A of the C/C++ Run-time Common Functions Reference Manual.

**Parameters:**

*membuf* reference to ASN1MemBuf object.

---

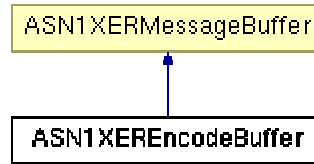
The documentation for this class was generated from the following file:

- **asn1XerCppTypes.h**

## ASN1XEREncodeBuffer Class Reference

```
#include <asn1XerCppTypes.h>
```

Inheritance diagram for ASN1XEREncodeBuffer:



---

### Detailed Description

The ASN1XEREncodeBuffer class is derived from the **ASN1XERMessageBuffer** base class. It contains variables and methods specific to encoding ASN.1 messages using the XML Encoding Rules (XER). It is used to manage the buffer into which an ASN.1 message is to be encoded.

### Public Member Functions

- **ASN1XEREncodeBuffer** ()
- **ASN1XEREncodeBuffer** (ASN1BOOL canonical)
- **ASN1XEREncodeBuffer** (ASN1BOOL canonical, ASN1CTXT \*pSrcCtxt, size\_t initBufSize=0)
- **ASN1XEREncodeBuffer** (ASN1OCTET \*pMsgBuf, size\_t msgBufLen)
- **ASN1XEREncodeBuffer** (ASN1OCTET \*pMsgBuf, size\_t msgBufLen, ASN1BOOL canonical, ASN1BOOL openType=FALSE)
- size\_t **getMsgLen** ()
- int **init** ()
- virtual ASN1BOOL **isA** (Type bufferType)
- void **setCanonical** ()
- void **setOpenType** ()
- long **write** (const char \*filename)
- long **write** (FILE \*fp)

---

### Constructor & Destructor Documentation

#### ASN1XEREncodeBuffer::ASN1XEREncodeBuffer ()

Default constructor Use getStatus() method to determine has error occurred during the initialization or not.

#### ASN1XEREncodeBuffer::ASN1XEREncodeBuffer (ASN1BOOL *canonical*)

A constructor. Use getStatus() method to determine has error occurred during the initialization or not.

#### Parameters:

*canonical* Indicates the usage of Canonical XER(CXER).

**ASN1XEREncodeBuffer::ASN1XEREncodeBuffer (ASN1BOOL *canonical*, ASN1CTXT \* *pSrcCtxt*, size\_t *initBufSize* = 0)**

A constructor. Use getStatus() method to determine has error occurred during the initialization or not.

**Parameters:**

*canonical* Indicates the usage fo Canonical XER(CXER).  
*pSrcCtxt* Source context; its memory heap will be used.  
*initBufSize* Initial size of encode buffer

**ASN1XEREncodeBuffer::ASN1XEREncodeBuffer (ASN1OCTET \* *pMsgBuf*, size\_t *msgBufLen*)**

A constructor. Use getStatus() method to determine has error occurred during the initialization or not.

**Parameters:**

*pMsgBuf* A pointer to a fixed size message buffer to receive the encoded message.  
*msgBufLen* Size of the fixed-size message buffer.

**ASN1XEREncodeBuffer::ASN1XEREncodeBuffer (ASN1OCTET \* *pMsgBuf*, size\_t *msgBufLen*, ASN1BOOL *canonical*, ASN1BOOL *openType* = FALSE)**

A constructor. Use getStatus() method to determine has error occurred during the initialization or not.

**Parameters:**

*pMsgBuf* A pointer to a fixed size message buffer to receive the encoded message.  
*msgBufLen* Size of the fixed-size message buffer.  
*canonical* Indicates the usage fo Canonical XER(CXER).  
*openType* Open Type encoding indicator. Set this argument as TRUE to encode an open type.

---

## Member Function Documentation

**size\_t ASN1XEREncodeBuffer::getMsgLen () [inline]**

This method returns the length of a previously encoded XER message.

**Returns:**

len Length of the XER message encapsulated within this buffer object.

**int ASN1XEREncodeBuffer::init ()**

This method reinitializes the encode buffer pointer to allow a new message to be encoded. This makes it possible to reuse one message buffer object in a loop to encode multiple messages. After this method is called, any previously encoded message in the buffer will be overwritten on the next encode call.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**virtual ASN1BOOL ASN1XEREncodeBuffer::isA (Type *bufferType*) [inline, virtual]**

This method checks the type of the message buffer.

**Parameters:**

*bufferType* Enumerated identifier specifying a derived class. The only possible value for this class is XEREncode.

**Returns:**

Boolean result of the match operation. True if this is the class corresponding to the identifier argument.

**void ASN1XEREncodeBuffer::setCanonical () [inline]**

This method turns Canonical XML Encoding Rules (CXER) usage on.

**void ASN1XEREncodeBuffer::setOpenType () [inline]**

This method turns Open Type encoding on.

**long ASN1XEREncodeBuffer::write (FILE \* fp)**

The second version writes to a file that is specified by a FILE pointer.

**Parameters:**

*fp* The pointer to FILE structure for writing the encoded message.

**Returns:**

Number of octets actually written, which may be less than real message length if an error occurs.

**long ASN1XEREncodeBuffer::write (const char \* filename)**

There are two versions of this method. The first version writes to a file that is specified by file name.

**Parameters:**

*filename* The name of file for writing the encoded message.

**Returns:**

Number of octets actually written, which may be less than real message length if an error occurs.

---

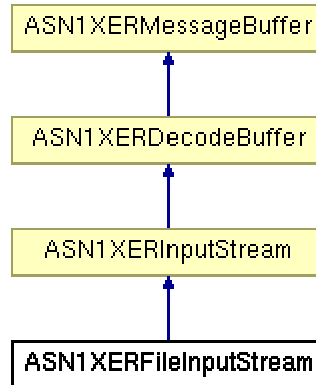
The documentation for this class was generated from the following file:

- **asn1XerCppTypes.h**

## ASN1XERFileInputStream Class Reference

```
#include <ASN1XERInputStream.h>
```

Inheritance diagram for ASN1XERFileInputStream:



---

### Detailed Description

A file input stream for streaming XER decoding. This class reads data from a file.

### Public Member Functions

- `ASN1XERFileInputStream` (const char \*pFilename)
- `ASN1XERFileInputStream` (FILE \*file)

---

### Constructor & Destructor Documentation

**ASN1XERFileInputStream::ASN1XERFileInputStream (const char \* pFilename) [inline]**

Creates and initializes the file input stream using the name of file. Use `getStatus()` method to determine has error occurred during the initialization or not.

**Parameters:**

*pFilename* Name of file.

**See also:**

`rtStreamFileOpen`

**ASN1XERFileInputStream::ASN1XERFileInputStream (FILE \* file) [inline]**

Initializes the file input stream using the opened FILE structure descriptor. Use `getStatus()` method to determine has error occurred during the initialization or not.

**Parameters:**

*file* Pointer to FILE structure.

**See also:**

`rtStreamFileAttach`

---

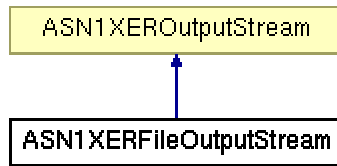
The documentation for this class was generated from the following file:

- **ASN1XERInputStream.h**

## ASN1XERFileOutputStream Class Reference

```
#include <ASN1XEROutputStream.h>
```

Inheritance diagram for ASN1XERFileOutputStream:



---

### Detailed Description

A file output stream for streaming BER encoding. This class writes data to a file.

### Public Member Functions

- **ASN1XERFileOutputStream** (const char \*pFilename)
- **ASN1XERFileOutputStream** (FILE \*file)

---

### Constructor & Destructor Documentation

#### ASN1XERFileOutputStream::ASN1XERFileOutputStream (const char \* pFilename)

Creates and initializes the file output stream using the name of file. Use getStatus() method to determine has error ocured during the initialization or not.

#### Parameters:

*pFilename* Name of file.

#### See also:

rtStreamFileOpen

#### ASN1XERFileOutputStream::ASN1XERFileOutputStream (FILE \* file)

Initializes the file output stream using the opened FILE structure descriptor. Use getStatus() method to determine has error ocured during the initialization or not.

#### Parameters:

*file* Pointer to FILE structure.

#### See also:

rtStreamFileAttach

---

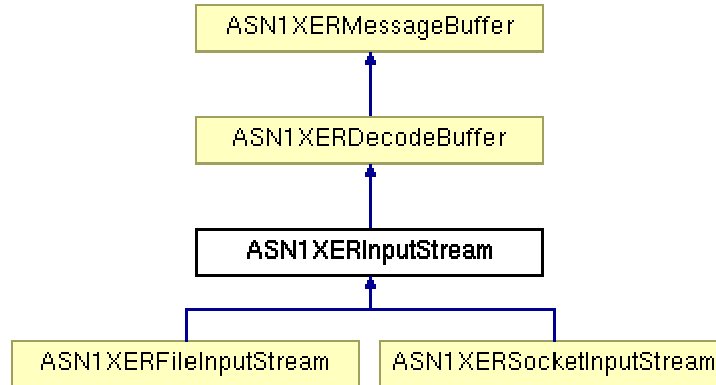
The documentation for this class was generated from the following file:

- **ASN1XEROutputStream.h**

## ASN1XERInputStream Class Reference

```
#include <ASN1XERInputStream.h>
```

Inheritance diagram for ASN1XERInputStream:



---

### Detailed Description

This class is a base class for other ASN.1 XER input stream's classes. It is derived from the ASN1Stream base class. It contains variables and methods specific to streaming decoding of XER messages. It is used to manage the input stream containing the ASN.1 message to be decoded.

### Public Member Functions

- **ASN1XERInputStream** (OSCInputStream &inputStream)
- **ASN1XERInputStream** (const ASN1OCTET \*msgbuf, size\_t numocts)
- **ASN1XERInputStream & operator>>** (ASN1CType &val)
- int **decodeObj** (ASN1CType &val)

### Protected Member Functions

- **ASN1XERInputStream** (const char \*pFilename)

---

### Constructor & Destructor Documentation

#### ASN1XERInputStream::ASN1XERInputStream (OSCInputStream & *inputStream*) [inline]

A constructor. Initializes input stream by using OSCInputStream instance. Use getStatus() method to determine has error occurred during the initialization or not.

#### Parameters:

*inputStream* Input stream object.

**ASN1XERInputStream::ASN1XERInputStream (const ASN1OCTET \* *msgbuf*, size\_t *numocts*) [inline]**

A constructor. Initializes input stream by using memory buffer. Use `getStatus()` method to determine has error occurred during the initialization or not.

**Parameters:**

*msgbuf* A pointer to a buffer containing an encoded ASN.1 message.  
*numocts* Size of the message buffer.

---

**Member Function Documentation**

**int ASN1XERInputStream::decodeObj (ASN1CType & *val*)**

This method decodes an ASN.1 constructed object from the stream.

**Parameters:**

*val* A reference to an object to be decoded.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**ASN1XERInputStream& ASN1XERInputStream::operator>> (ASN1CType & *val*)**

Decodes an ASN.1 constructed object from the stream. Use `getStatus()` method to determine has error occurred during the operation or not.

**Parameters:**

*val* A reference to an object to be decoded.

**Returns:**

reference to this class to perform sequential decoding.

---

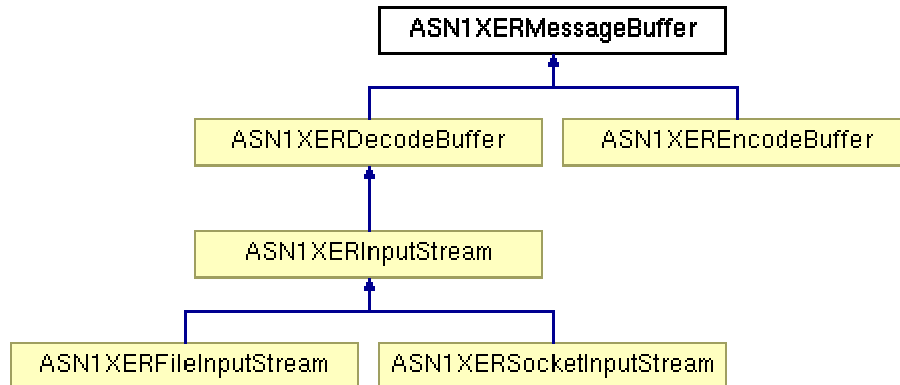
The documentation for this class was generated from the following file:

- **ASN1XERInputStream.h**

## ASN1XERMessageBuffer Class Reference

```
#include <asn1XerCppTypes.h>
```

Inheritance diagram for ASN1XERMessageBuffer:



---

### Detailed Description

The XER message buffer class is derived from the ASN1MessageBuffer base class. It is the base class for the **ASN1XEREncodeBuffer** and **ASN1XERDecodeBuffer** classes. It contains variables and methods specific to encoding or decoding ASN.1 messages using the XML Encoding Rules (XER). It is used to manage the buffer into which an ASN.1 message is to be encoded or decoded.

### Protected Member Functions

- **ASN1XERMessageBuffer** (Type *bufferType*)

---

### Constructor & Destructor Documentation

**ASN1XERMessageBuffer::ASN1XERMessageBuffer (Type *bufferType*)** [*inline*, *protected*]

The protected constructor creates a new context and sets the buffer class type. Use `getStatus()` method to determine has error occurred during the initialization or not.

#### Parameters:

*bufferType* Type of message buffer that is being created (for example, XEREncode or XERDecode).

---

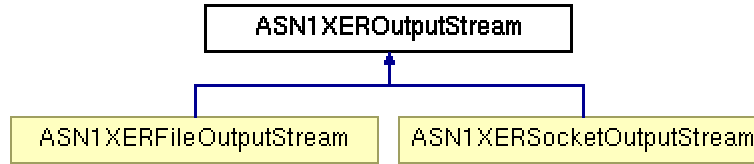
The documentation for this class was generated from the following file:

- **asn1XerCppTypes.h**

## ASN1XEROutputStream Class Reference

```
#include <ASN1XEROutputStream.h>
```

Inheritance diagram for ASN1XEROutputStream:



---

### Detailed Description

This class is a base class for other ASN.1 XER output stream's classes. It is derived from the ASN1Stream base class. It contains variables and methods specific to streaming encoding of XER messages.

### Public Member Functions

- **ASN1XEROutputStream & operator<<** (ASN1CType &val)
- int **encodeStartElement** (const char \*elemName, const char \*attributes=0)
- int **encodeEndElement** (const char \*elemName)
- int **encodeEmptyElement** (const char \*elemName, const char \*attributes=0)
- int **encodeBMPStr** (const Asn116BitCharString &val, const char \*elemName)
- int **encodeBigInt** (const char \*pval, const char \*elemName)
- int **encodeBitStr** (const ASN1OCTET \*pbits, ASN1UINT numbits, const char \*elemName, ASN1StrType outputType)
- int **encodeBitStr** (const ASN1DynBitStr &val, const char \*elemName, ASN1StrType outputType)
- int **encodeBool** (ASN1BOOL val, const char \*elemName)
- int **encodeCharStr** (const char \*pval, const char \*elemName)
- int **encodeInt** (ASN1INT val, const char \*elemName)
- int **encodeInt64** (ASN1INT64 val, const char \*elemName)
- int **encodeNull** (const char \*elemName)
- int **encodeObj** (ASN1CType &val)
- int **encodeObjId** (const ASN1OBJID &val, const char \*elemName)
- int **encodeObjId64** (const ASN1OID64 &val, const char \*elemName)
- int **encodeOctStr** (const ASN1OCTET \*pocets, ASN1UINT numocets, const char \*elemName)
- int **encodeOctStr** (const ASN1DynOctStr &val, const char \*elemName)
- int **encodeReal** (ASN1REAL val, const char \*elemName)
- int **encodeRelativeOID** (const ASN1OBJID &val, const char \*elemName)
- int **encodeUInt** (ASN1UINT val, const char \*elemName)
- int **encodeUInt64** (ASN1UINT64 val, const char \*elemName)
- int **encodeUnivStr** (const Asn132BitCharString &val, const char \*elemName)

### Protected Member Functions

- **ASN1XEROutputStream** ()
-

## Constructor & Destructor Documentation

### ASN1XEROutputStream::ASN1XEROutputStream () [inline, protected]

A default constructor. Use getStatus() method to determine has error occurred during the initialization or not.

---

## Member Function Documentation

### int ASN1XEROutputStream::encodeBigInt (const char \* pval, const char \* elemName)

This method encodes a variable of the ASN.1 INTEGER type. In this case, the integer is assumed to be of a larger size than can fit in a C or C++ long type (normally 32 or 64 bits).

#### Parameters:

- pval* A pointer to a character string containing the value to be encoded.
- elemName* A pointer to the element's name. It can be NULL, no tag will be encoded in this case.

#### Returns:

- Completion status of operation:
- 0 (ASN\_OK) = success,
  - negative return value is error.

### int ASN1XEROutputStream::encodeBitStr (const ASN1DynBitStr & val, const char \* elemName, ASN1StrType outputType)

This method encodes a variable of the ASN.1 BIT STRING type.

#### Parameters:

- val* A reference to the ASN1DynBitStr structure containing a bit data and number of bits to be encoded.
- elemName* A pointer to the element's name. It can be NULL, no tag will be encoded in this case.
- outputType* An enumerated type whose value is set to either 'ASN1BIN' (for binary format) or 'ASN1HEX' (for hexadecimal format).

#### Returns:

- Completion status of operation:
- 0 (ASN\_OK) = success,
  - negative return value is error.

### int ASN1XEROutputStream::encodeBitStr (const ASN1OCTET \* pbits, ASN1UINT numbits, const char \* elemName, ASN1StrType outputType)

This method encodes a variable of the ASN.1 BIT STRING type.

#### Parameters:

- pbits* A pointer to an OCTET string containing the bit data to be encoded. This string contains bytes having the actual bit settings as they are to be encoded in the message.
- numbits* The number of bits within the bit string to be encoded.
- elemName* A pointer to the element's name. It can be NULL, no tag will be encoded in this case.
- outputType* An enumerated type whose value is set to either 'ASN1BIN' (for binary format) or 'ASN1HEX' (for hexadecimal format).

#### Returns:

- Completion status of operation:
- 0 (ASN\_OK) = success,
  - negative return value is error.

**int ASN1XEROutputStream::encodeBMPStr (const Asn116BitCharString & val, const char \* elemName)**

This method encodes a variable of the ASN.1 BMPString type that is based on a 16-bit character sets.

**Parameters:**

*val* A reference to a structure representing a 16-bit character string to be encoded. This structure contains a character count element and a pointer to an array of 16-bit character elements represented as 16-bit short integers.

*elemName* A pointer to the element's name. It can be NULL, no tag will be encoded in this case.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int ASN1XEROutputStream::encodeBool (ASN1BOOL val, const char \* elemName)**

This method encodes a variable of the ASN.1 BOOLEAN type.

**Parameters:**

*val* A BOOLEAN value to be encoded. A BOOLEAN is defined as a single OCTET whose value is 0 for FALSE and any other value for TRUE.

*elemName* A pointer to the element's name. It can be NULL, no tag will be encoded in this case.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int ASN1XEROutputStream::encodeCharStr (const char \* pval, const char \* elemName)**

This method encodes a variable of the ASN.1 character string type.

**Parameters:**

*pval* A pointer to a null-terminated C character string to be encoded.

*elemName* A pointer to the element's name. It can be NULL, no tag will be encoded in this case.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int ASN1XEROutputStream::encodeEmptyElement (const char \* elemName, const char \* attributes = 0)**

This method encodes an empty element.

**Parameters:**

*elemName* A pointer to the element's name.

*attributes* A pointer to attributes.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int ASN1XEROutputStream::encodeEndElement (const char \* *elemName*)**

This method encodes an end element tag.

**Parameters:**

*elemName* A pointer to the element's name.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int ASN1XEROutputStream::encodeInt (ASN1INT *val*, const char \* *elemName*)**

This method encodes a variable of the ASN.1 INTEGER type.

**Parameters:**

*val* A 32-bit INTEGER value to be encoded.

*elemName* A pointer to the element's name. It can be NULL, no tag will be encoded in this case.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int ASN1XEROutputStream::encodeInt64 (ASN1INT64 *val*, const char \* *elemName*)**

This method encodes a 64-bit variable of the ASN.1 INTEGER type.

**Parameters:**

*val* A 64-bit INTEGER value to be encoded.

*elemName* A pointer to the element's name. It can be NULL, no tag will be encoded in this case.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int ASN1XEROutputStream::encodeNull (const char \* *elemName*)**

This method encodes a variable of the ASN.1 NULL type.

**Parameters:**

*elemName* A pointer to the element's name. It can be NULL, no tag will be encoded in this case.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int ASN1XEROutputStream::encodeObj (ASN1CType & *val*)**

This method encodes an ASN.1 constructed object to the stream.

**Parameters:**

*val* A reference to an object to be encoded.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int ASN1XEROutputStream::encodeObjId (const ASN1OBJID & val, const char \* elemName)**

This method encodes a variable of the ASN.1 OBJECT IDENTIFIER type.

**Parameters:**

*val* A reference to an object identifier structure. This structure contains an integer to hold the number of subidentifiers in the object and an array to hold the subidentifier values.  
*elemName* A pointer to the element's name. It can be NULL, no tag will be encoded in this case.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int ASN1XEROutputStream::encodeObjId64 (const ASN1OID64 & val, const char \* elemName)**

This method encodes a variable of the ASN.1 OBJECT IDENTIFIER type using 64-bit subidentifiers.

**Parameters:**

*val* A reference to a 64-bit object identifier structure. This structure contains an integer to hold the number of subidentifiers in the object and an array of 64-bit unsigned integers to hold the subidentifier values.  
*elemName* A pointer to the element's name. It can be NULL, no tag will be encoded in this case.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int ASN1XEROutputStream::encodeOctStr (const ASN1DynOctStr & val, const char \* elemName)**

This method encodes a variable of the ASN.1 OCTET STRING type.

**Parameters:**

*val* A reference to the ASN1DynOctStr structure containing an octet data and number of octets to be encoded.  
*elemName* A pointer to the element's name. It can be NULL, no tag will be encoded in this case.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int ASN1XEROutputStream::encodeOctStr (const ASN1OCTET \* pocts, ASN1UINT numocts, const char \* elemName)**

This method encodes a variable of the ASN.1 OCTET STRING type.

**Parameters:**

*pocts* A pointer to an OCTET STRING containing the octet data to be encoded.  
*numocts* The number of octets (bytes) within the OCTET STRING to be encoded.  
*elemName* A pointer to the element's name. It can be NULL, no tag will be encoded in this case.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,

- negative return value is error.

**int ASN1XEROutputStream::encodeReal (ASN1REAL val, const char \* elemName)**

This method encodes a variable of the REAL data type. It provides support for the plus-infinity and minus-infinity special real values. Use the `rtGetPlusInfinity` or `rtGetMinusInfinity` functions to get these special values.

**Parameters:**

*val* An ASN1REAL data type. This is defined to be the C double type. Special real values plus and minus infinity are encoded by using the `rtGetPlusInfinity` and `rtGetMinusInfinity` functions to set the real value to be encoded.

*elemName* A pointer to the element's name. It can be NULL, no tag will be encoded in this case.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int ASN1XEROutputStream::encodeRelativeOID (const ASN1OBJID & val, const char \* elemName)**

This method encodes a variable of the ASN.1 RELATIVE-OID type.

**Parameters:**

*val* A reference to an object identifier structure. This structure contains an integer to hold the number of subidentifiers in the object and an array to hold the subidentifier values.

*elemName* A pointer to the element's name. It can be NULL, no tag will be encoded in this case.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int ASN1XEROutputStream::encodeStartElement (const char \* elemName, const char \* attributes = 0)**

This method encodes a start element tag.

**Parameters:**

*elemName* A pointer to the element's name.

*attributes* A pointer to attributes.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

**int ASN1XEROutputStream::encodeUInt (ASN1UINT val, const char \* elemName)**

This method encodes an unsigned variable of the ASN.1 INTEGER type.

**Parameters:**

*val* An unsigned INTEGER value to be encoded.

*elemName* A pointer to the element's name. It can be NULL, no tag will be encoded in this case.

**Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

### **int ASN1XEROutputStream::encodeUInt64 (ASN1UINT64 *val*, const char \* *elemName*)**

This method encodes a 64-bit unsigned variable of the ASN.1 INTEGER type.

#### **Parameters:**

*val* A 64-bit unsigned INTEGER value to be encoded.

*elemName* A pointer to the element's name. It can be NULL, no tag will be encoded in this case.

#### **Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

### **int ASN1XEROutputStream::encodeUnivStr (const Asn132BitCharString & *val*, const char \* *elemName*)**

This method encodes a variable of the ASN.1 UniversalString type that is based on a 32-bit character sets.

#### **Parameters:**

*val* A reference to a structure representing a 32-bit character string to be encoded. This structure contains a character count element and a pointer to an array of 32-bit character elements represented as 32-bit unsigned integers.

*elemName* A pointer to the element's name. It can be NULL, no tag will be encoded in this case.

#### **Returns:**

Completion status of operation:

- 0 (ASN\_OK) = success,
- negative return value is error.

### **ASN1XEROutputStream& ASN1XEROutputStream::operator<< (ASN1CType & *val*)**

Encodes an ASN.1 constructed object to the stream. Use getStatus() method to determine has error occurred during the operation or not.

#### **Parameters:**

*val* A reference to an object to be encoded.

#### **Returns:**

reference to this class to perform sequential encoding.

---

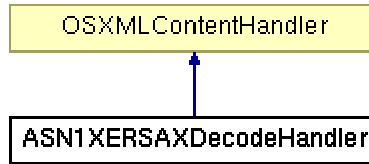
The documentation for this class was generated from the following file:

- **ASN1XEROutputStream.h**

## ASN1XERSAXDecodeHandler Class Reference

```
#include <asn1XerCppTypes.h>
```

Inheritance diagram for ASN1XERSAXDecodeHandler:



---

### Detailed Description

This class is derived from the SAX class DefaultHandler base class. It contains variables and methods specific to XML decoding ASN.1 messages. It is used to intercept standard SAX parser events, such as startElement, characters, endElement. This class is used as a base class for XER ASN1C generated ASN1C\_\* classes.

### Public Member Functions

- **ASN1XERSAXDecodeHandler** ()
- virtual void **startElement** (const XMLCHAR \*const uri, const XMLCHAR \*const localname, const XMLCHAR \*const qname, const XMLCHAR \*const \*attrs)
- virtual void **characters** (const XMLCHAR \*const chars, const unsigned int length)
- virtual void **endElement** (const XMLCHAR \*const uri, const XMLCHAR \*const localname, const XMLCHAR \*const qname)
- virtual void **startDocument** ()
- virtual void **endDocument** ()
- virtual void **resetErrorInfo** ()
- virtual void **setErrorInfo** (int status, const char \*file=0, int line=0)
- virtual int **getErrorInfo** (int \*status, const char \*\*file, int \*line)
- ASN1XERState **getState** ()
- virtual void **finalize** ()
- ASN1CTXT \* **finalizeMemBuf** (ASN1MessageBufferIF \*msgBuf, ASN1MemBuf &memBuf)
- virtual void **init** (int level=0)
- ASN1BOOL **isComplete** ()
- void **setTypeName** (const char \*typeName)
- virtual void **throwSAXException** (int stat, const char \*file=0, int line=0)

### Protected Attributes

- ASN1XERState **mCurrState**
- int **mCurrElemID**
- int **mLevel**
- int **mStartLevel**
- const char \* **mpTypeName**
- ASN1XERSAXDecodeHandler::ErrorInfo **errorInfo**

## Classes

- struct **ErrorInfo**
- 

## Constructor & Destructor Documentation

### **ASN1XERSAXDecodeHandler::ASN1XERSAXDecodeHandler ()** [inline]

Default constructor without parameters.

---

## Member Function Documentation

### **virtual void ASN1XERSAXDecodeHandler::characters (const XMLCHAR \*const *chars*, const unsigned int *length*)** [virtual]

Receive notification of character data.

The Parser will call this method to report each chunk of character data. SAX parsers may return all contiguous character data in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity, so that the Locator provides useful information.

The application must not attempt to read from the array outside of the specified range.

Note that some parsers will report whitespace using the `ignorableWhitespace()` method rather than this one (validating parsers must do so).

#### **Parameters:**

*chars* The characters from the XML document.

*length* The number of characters to read from the array.

### **Implements OSXMLContentHandler (p.63).virtual void ASN1XERSAXDecodeHandler::endElement (const XMLCHAR \*const *uri*, const XMLCHAR \*const *localname*, const XMLCHAR \*const *qname*)** [virtual]

Receive notification of the end of an element.

The SAX parser will invoke this method at the end of every element in the XML document; there will be a corresponding `startElement()` event for every `endElement()` event (even when the element is empty).

#### **Parameters:**

*uri* The URI of the associated namespace for this element

*localname* The local part of the element name

*qname* The QName of this element

### **Implements OSXMLContentHandler (p.64).ASN1XERState ASN1XERSAXDecodeHandler::getState ()** [inline]

This method returns the current state of the decoding process.

#### **Returns:**

The state of the decoding process as type `ASN1XERState`. Can be `XERINIT`, `XERSTART`, `XERDATA`, or `XEREND`.

### **ASN1BOOL ASN1XERSAXDecodeHandler::isComplete () [inline]**

This method returns the completion status of the decoding process.

#### **Returns:**

The completion state of decoding process, as ASN1BOOL. Can be TRUE (completed) or FALSE (not completed).

### **virtual void ASN1XERSAXDecodeHandler::startElement (const XMLCHAR \*const *uri*, const XMLCHAR \*const *localname*, const XMLCHAR \*const *qname*, const XMLCHAR \*const \* *attrs*) [virtual]**

Receive notification of the beginning of an element.

The Parser will invoke this method at the beginning of every element in the XML document; there will be a corresponding **endElement()** event for every **startElement()** event (even when the element is empty). All of the element's content will be reported, in order, before the corresponding **endElement()** event.

Note that the attribute list provided will contain only attributes with explicit values (specified or defaulted): IMPLIED attributes will be omitted.

#### **Parameters:**

*uri* The URI of the associated namespace for this element

*localname* The local part of the element name

*qname* The QName of this element

*attrs* The attributes attached to the element, if any.

Implements **OSXMLContentHandler** (p.64).

---

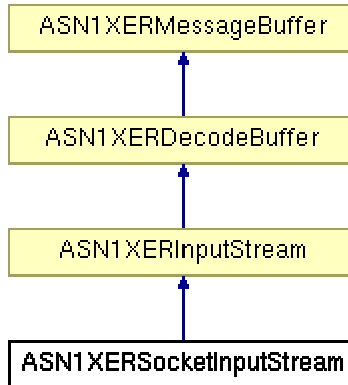
The documentation for this class was generated from the following file:

- **asn1XerCppTypes.h**

## ASN1XERSocketInputStream Class Reference

```
#include <ASN1XERInputStream.h>
```

Inheritance diagram for ASN1XERSocketInputStream:



---

### Detailed Description

A socket input stream for streaming XER decoding. This class reads data directly from a network via TCP/IP socket.

### Public Member Functions

- **ASN1XERSocketInputStream** (OSRTSOCKET socket, ASN1BOOL ownership=FALSE)
- **ASN1XERSocketInputStream** (OSCSocket &socket)

### Protected Attributes

- OSCSocket **mSocket**

---

### Constructor & Destructor Documentation

**ASN1XERSocketInputStream::ASN1XERSocketInputStream (OSRTSOCKET *socket*, ASN1BOOL *ownership* = FALSE) [inline]**

Creates and initializes the socket input stream using the socket's handle. Use `getStatus()` method to determine has error occurred during the initialization or not.

#### Parameters:

*socket* Handle of the socket.

*ownership* Indicates the ownership of the socket's handle.

#### See also:

`rtSocketCreate`, `rtSocketAccept`  
`rtStreamSocketAttach`

**ASN1XERSocketInputStream::ASN1XERSocketInputStream (OSCSocket & socket)**  
[inline]

Initializes the socket input stream using the OSCSocket class' instance. Use getStatus() method to determine has error ocured during the initialization or not.

**Parameters:**

*socket* Reference to OSCSocket class' instance.

---

**Member Data Documentation**

**OSCSocket ASN1XERSocketInputStream::mSocket [protected]**

a socket

---

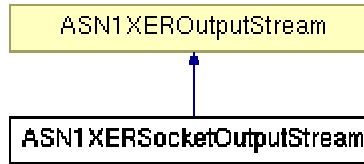
The documentation for this class was generated from the following file:

- **ASN1XERInputStream.h**

## ASN1XERSocketOutputStream Class Reference

```
#include <ASN1XEROutputStream.h>
```

Inheritance diagram for ASN1XERSocketOutputStream:



---

### Detailed Description

A socket output stream for streaming XER encoding. This class writes data directly to a network via TCP/IP socket.

### Public Member Functions

- **ASN1XERSocketOutputStream** (OSRTSOCKET socket, ASN1BOOL ownership=FALSE)
- **ASN1XERSocketOutputStream** (OSCSocket &socket)

### Protected Attributes

- OSCSocket **mSocket**

---

### Constructor & Destructor Documentation

#### **ASN1XERSocketOutputStream::ASN1XERSocketOutputStream (OSRTSOCKET *socket*, ASN1BOOL *ownership* = FALSE)**

Creates and initializes the socket output stream using the socket's handle. Use `getStatus()` method to determine has error occurred during the initialization or not.

#### **Parameters:**

*socket* Handle of the socket.

*ownership* Boolean value, indicates the ownership of the \* socket's handle.

#### **See also:**

`rtSocketCreate`, `rtSocketAccept`

`rtStreamSocketAttach`

#### **ASN1XERSocketOutputStream::ASN1XERSocketOutputStream (OSCSocket & *socket*)**

Initializes the socket output stream using the OSCSocket class instance. Use `getStatus()` method to determine has error occurred during the initialization or not.

#### **Parameters:**

*socket* Reference to OSCSocket class instance.

## Member Data Documentation

**OSCSocket ASN1XERSocketOutputStream::mSocket** [protected]

a socket

---

The documentation for this class was generated from the following file:

- **ASN1XEROutputStream.h**

## OSRTSAXException Class Reference

```
#include <saxParser.h>
```

---

### Detailed Description

Encapsulate a general SAX error or warning.

This class can contain basic error or warning information from either the XML SAX parser or the application: a parser writer or application writer can subclass it to provide additional functionality. SAX handlers may throw this exception or any exception subclassed from it.

### Public Member Functions

- virtual const XMLCHAR \* **getMessage** ()=0
  - virtual int **getStatus** ()=0
  - virtual const XMLCHAR \* **getSrcFileName** ()=0
  - virtual int **getSrcLineNum** ()=0
- 

### Member Function Documentation

**virtual const XMLCHAR\* OSRTSAXException::getMessage () [pure virtual]**

Get the contents of the message.

---

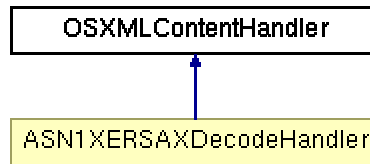
The documentation for this class was generated from the following file:

- saxParser.h

## OSXMLContentHandler Class Reference

```
#include <saxParser.h>
```

Inheritance diagram for OSXMLContentHandler:



---

### Detailed Description

Receive notification of general document events.

This is the main interface that most SAX2 applications implement: if the application needs to be informed of basic parsing events, it implements this interface and registers an instance with the SAX2 parser using the `setDocumentHandler` method. The parser uses the instance to report basic document-related events like the start and end of elements and character data.

The order of events in this interface is very important, and mirrors the order of information in the document itself. For example, all of an element's content (character data, processing instructions, and/or subelements) will appear, in order, between the `startElement` event and the corresponding `endElement` event.

### Public Member Functions

#### The virtual document handler interface

- virtual void **characters** (const XMLCHAR \*const chars, const unsigned int length)=0
- virtual void **endElement** (const XMLCHAR \*const uri, const XMLCHAR \*const localname, const XMLCHAR \*const qname)=0
- virtual void **startElement** (const XMLCHAR \*const uri, const XMLCHAR \*const localname, const XMLCHAR \*const qname, const XMLCHAR \*const \*attrs)=0

---

### Member Function Documentation

**virtual void OSXMLContentHandler::characters (const XMLCHAR \*const *chars*, const unsigned int *length*)** [pure virtual]

Receive notification of character data.

The Parser will call this method to report each chunk of character data. SAX parsers may return all contiguous character data in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity, so that the Locator provides useful information.

The application must not attempt to read from the array outside of the specified range.

Note that some parsers will report whitespace using the `ignorableWhitespace()` method rather than this one (validating parsers must do so).

**Parameters:**

*chars* The characters from the XML document.  
*length* The number of characters to read from the array.

**Implemented in ASN1XERSAXDecodeHandler (p.56).virtual void OSXMLContentHandler::endElement (const XMLCHAR \*const uri, const XMLCHAR \*const localname, const XMLCHAR \*const qname) [pure virtual]**

Receive notification of the end of an element.

The SAX parser will invoke this method at the end of every element in the XML document; there will be a corresponding **startElement()** event for every **endElement()** event (even when the element is empty).

**Parameters:**

*uri* The URI of the associated namespace for this element  
*localname* The local part of the element name  
*qname* The QName of this element

**Implemented in ASN1XERSAXDecodeHandler (p.56).virtual void OSXMLContentHandler::startElement (const XMLCHAR \*const uri, const XMLCHAR \*const localname, const XMLCHAR \*const qname, const XMLCHAR \*const \* attrs) [pure virtual]**

Receive notification of the beginning of an element.

The Parser will invoke this method at the beginning of every element in the XML document; there will be a corresponding **endElement()** event for every **startElement()** event (even when the element is empty). All of the element's content will be reported, in order, before the corresponding **endElement()** event.

Note that the attribute list provided will contain only attributes with explicit values (specified or defaulted): IMPLIED attributes will be omitted.

**Parameters:**

*uri* The URI of the associated namespace for this element  
*localname* The local part of the element name  
*qname* The QName of this element  
*attrs* The attributes attached to the element, if any.

Implemented in **ASN1XERSAXDecodeHandler (p.57)**.

---

The documentation for this class was generated from the following file:

- saxParser.h

# ASN1C XER Runtime File Documentation

## ASN1CXerOpenType.h File Reference

---

### Detailed Description

Holds class that defines standard SAX even handlers for decoding XER open data type

## ASN1SAX\_XEROpenType.h File Reference

---

### Detailed Description

```
#include "asn1XerCTypes.h"
```

### Classes

- struct **ASN1SAX\_XEROpenType**

### Typedefs

- typedef **ASN1SAX\_XEROpenType** **ASN1SAX\_XEROpenType**

### Functions

- void **asn1Sax\_XEROpenType\_startElement** (void \*userData, ASN1Const XMLCHAR \*localname, ASN1Const XMLCHAR \*\*atts)
- void **asn1Sax\_XEROpenType\_characters** (void \*userData, ASN1Const XMLCHAR \*chars, int length)
- void **asn1Sax\_XEROpenType\_endElement** (void \*userData, ASN1Const XMLCHAR \*localname)
- void **asn1Sax\_XEROpenType\_init** (ASN1CTXT \*pctx, ASN1SAX\_XEROpenType \*pSaxHandler, ASN1OpenType \*pvalue, int level)
- void **asn1Sax\_XEROpenType\_free** (ASN1CTXT \*pctx, ASN1SAX\_XEROpenType \*pSaxHandler)

# asn1xer.h File Reference

---

## Detailed Description

ASN.1 runtime constants, data structure definitions, and functions to support the XML Encoding Rules (XER) as defined in the ITU-T X.693 standard.

```
#include "rtsrc/asn1type.h"
#include "rtSAXDefs.h"
```

## Classes

- struct **XerElemInfo**
- struct **XmlNamedBitsDict**
- struct **OSXMLNamespace**

## Defines

- #define **XERINDENT** 3
- #define **XERBYTECNT**(pctxt) (pctxt->buffer.byteIndex)
- #define **EXTERNXER**

## Typedefs

- typedef XmlNamedBitsDict **XmlNamedBitsDict**
- typedef OSXMLNamespace **OSXMLNamespace**

## Enumerations

- enum **ASN1XERState** { **XERINIT**, **XERSTART**, **XERDATA**, **XEREND**, **XERSTART0**, **XEREND0** }

## Functions

- int **xerDecBMPStr** (ASN1CTXT \*pctxt, ASN1BMPString \*outdata)
- int **xerDecBase64Str** (ASN1CTXT \*pctxt, ASN1OCTET \*pvalue, ASN1UINT \*pnocts, ASN1INT bufsize)
- int **xerDecBigInt** (ASN1CTXT \*pctxt, char \*\*ppvalue, int radix)
- int **xerDecBitStr** (ASN1CTXT \*pctxt, ASN1OCTET \*pvalue, ASN1UINT \*pnbits, ASN1INT bufsize)
- int **xerDecBitStrMemBuf** (ASN1MemBuf \*pMemBuf, ASN1Const XMLCHAR \*inpdata, int length, ASN1BOOL skipWhitespaces)
- int **xerDecBool** (ASN1CTXT \*pctxt, ASN1BOOL \*pvalue)
- int **xerDecCopyBitStr** (ASN1CTXT \*pctxt, ASN1OCTET \*pvalue, ASN1UINT \*pnbits, ASN1INT bufsize, int lastBitOffset)
- int **xerDecCopyDynBitStr** (ASN1CTXT \*pctxt, ASN1DynBitStr \*pvalue, int lastBitOffset)
- int **xerDecCopyDynOctStr** (ASN1CTXT \*pctxt, ASN1DynOctStr \*pvalue, int lastBitOffset)
- int **xerDecCopyOctStr** (ASN1CTXT \*pctxt, ASN1OCTET \*pvalue, ASN1UINT \*pnocts, ASN1INT bufsize, int lastBitOffset)
- int **xerDecDynAscCharStr** (ASN1CTXT \*pctxt, ASN1ConstCharPtr \*outdata)
- int **xerDecDynBase64Str** (ASN1CTXT \*pctxt, ASN1DynOctStr \*pvalue)
- int **xerDecDynBitStr** (ASN1CTXT \*pctxt, ASN1DynBitStr \*pvalue)
- int **xerDecDynOctStr** (ASN1CTXT \*pctxt, ASN1DynOctStr \*pvalue)
- int **xerDecDynUTF8Str** (ASN1CTXT \*pctxt, ASN1UTF8String \*outdata)
- int **xerDecInt** (ASN1CTXT \*pctxt, ASN1INT \*pvalue)
- int **xerDecInt8** (ASN1CTXT \*pctxt, ASN1INT8 \*pvalue)

- int **xerDecInt16** (ASN1CTXT \*pctxt, ASN1SINT \*pvalue)
- int **xerDecInt64** (ASN1CTXT \*pctxt, ASN1INT64 \*pvalue)
- int **xerDecObjId** (ASN1CTXT \*pctxt, ASN1OBJID \*pvalue)
- int **xerDecObjId64** (ASN1CTXT \*pctxt, ASN1OID64 \*pvalue)
- int **xerDecOctStr** (ASN1CTXT \*pctxt, ASN1OCTET \*pvalue, ASN1UINT \*pnocts, ASN1INT bufsize)
- int **xerDecOctStrMemBuf** (ASN1MemBuf \*pMemBuf, ASN1Const XMLCHAR \*inpdata, int length, ASN1BOOL skipWhitespaces)
- int **xerDecOpenType** (ASN1CTXT \*pctxt, ASN1OpenType \*pvalue)
- int **xerDecReal** (ASN1CTXT \*pctxt, ASN1REAL \*pvalue)
- int **xerDecRelativeOID** (ASN1CTXT \*pctxt, ASN1OBJID \*pvalue)
- int **xerDecUInt** (ASN1CTXT \*pctxt, ASN1UINT \*pvalue)
- int **xerDecUInt8** (ASN1CTXT \*pctxt, ASN1UINT8 \*pvalue)
- int **xerDecUInt16** (ASN1CTXT \*pctxt, ASN1USINT \*pvalue)
- int **xerDecUInt64** (ASN1CTXT \*pctxt, ASN1UINT64 \*pvalue)
- int **xerDecUnivStr** (ASN1CTXT \*pctxt, ASN1UniversalString \*outdata)
- int **xerSetDecBufPtr** (ASN1CTXT \*pctxt, ASN1ConstOctetPtr bufaddr, size\_t bufsiz)
- int **xerSetEncBufPtr** (ASN1CTXT \*pctxt, ASN1OCTET \*bufaddr, size\_t bufsiz, ASN1BOOL canonical)
- int **xerEncAscCharStr** (ASN1CTXT \*pctxt, ASN1ConstCharPtr value, ASN1ConstCharPtr elemName)
- int **xerEncBase64Str** (ASN1CTXT \*pctxt, ASN1UINT nocts, ASN1ConstOctetPtr data, ASN1ConstCharPtr elemName)
- int **xerEncBigInt** (ASN1CTXT \*pctxt, ASN1ConstCharPtr value, ASN1ConstCharPtr elemName)
- int **xerEncBitStr** (ASN1CTXT \*pctxt, ASN1UINT nbits, ASN1ConstOctetPtr data, ASN1ConstCharPtr elemName, ASN1StrType outputType)
- int **xerEncBoolValue** (ASN1CTXT \*pctxt, ASN1BOOL value)
- int **xerEncBool** (ASN1CTXT \*pctxt, ASN1BOOL value, ASN1ConstCharPtr elemName)
- int **xerEncEndDocument** (ASN1CTXT \*pctxt)
- int **xerEncEndElement** (ASN1CTXT \*pctxt, ASN1ConstCharPtr elemName)
- int **xerEncIndent** (ASN1CTXT \*pctxt)
- int **xerEncInt** (ASN1CTXT \*pctxt, ASN1INT value, ASN1ConstCharPtr elemName)
- int **xerEncInt64** (ASN1CTXT \*pctxt, ASN1INT64 value, ASN1ConstCharPtr elemName)
- int **xerEncNewLine** (ASN1CTXT \*pctxt)
- int **xerEncObjId** (ASN1CTXT \*pctxt, const ASN1OBJID \*pvalue, ASN1ConstCharPtr elemName)
- int **xerEncObjId64** (ASN1CTXT \*pctxt, const ASN1OID64 \*pvalue, ASN1ConstCharPtr elemName)
- int **xerEncRelativeOID** (ASN1CTXT \*pctxt, const ASN1OBJID \*pvalue, ASN1ConstCharPtr elemName)
- int **xerEncOctStr** (ASN1CTXT \*pctxt, ASN1UINT nocts, ASN1ConstOctetPtr data, ASN1ConstCharPtr elemName)
- int **xerEncReal** (ASN1CTXT \*pctxt, ASN1REAL value, ASN1ConstCharPtr elemName)
- int **xerEncStartDocument** (ASN1CTXT \*pctxt)
- int **xerEncStartElement** (ASN1CTXT \*pctxt, ASN1ConstCharPtr elemName, ASN1ConstCharPtr attributes)
- int **xerEncEmptyElement** (ASN1CTXT \*pctxt, ASN1ConstCharPtr elemName, ASN1ConstCharPtr attributes)
- int **xerEncNamedValue** (ASN1CTXT \*pctxt, ASN1ConstCharPtr value, ASN1ConstCharPtr elemName, ASN1ConstCharPtr attributes)
- int **xerEncUInt** (ASN1CTXT \*pctxt, ASN1UINT value, ASN1ConstCharPtr elemName)
- int **xerEncUInt64** (ASN1CTXT \*pctxt, ASN1UINT64 value, ASN1ConstCharPtr elemName)
- int **xerEncBMPStr** (ASN1CTXT \*pctxt, const ASN1BMPString \*value, ASN1ConstCharPtr elemName)

- int **xerEncUnivStr** (ASN1CTXT \*pctxt, const ASN1UniversalString \*value, ASN1ConstCharPtr elemName)
- int **xerEncUniCharData** (ASN1CTXT \*pctxt, ASN1Const16BitCharPtr value, ASN1UINT nchars)
- int **xerEncUniCharStr** (ASN1CTXT \*pctxt, ASN116BITCHAR \*value, ASN1ConstCharPtr elemName)
- int **xerEncOpenType** (ASN1CTXT \*pctxt, ASN1UINT nocts, ASN1ConstOctetPtr data, ASN1ConstCharPtr elemName)
- int **xerEncNull** (ASN1CTXT \*pctxt, ASN1ConstCharPtr elemName)
- int **xerEncXmlCharData** (ASN1CTXT \*pctxt, ASN1Const XMLCHAR \*pvalue, int length)
- ASN1BOOL **xerCmpText** (ASN1Const XMLCHAR \*text1, ASN1ConstCharPtr text2)
- int **xerCopyText** (ASN1CTXT \*pctxt, ASN1ConstCharPtr text)
- int **xerTextLength** (ASN1Const XMLCHAR \*text)
- ASN1ConstCharPtr **xerTextToCStr** (ASN1CTXT \*pctxt, ASN1Const XMLCHAR \*text)
- size\_t **xerGetMsgLen** (ASN1CTXT \*pctxt)
- ASN1OCTET \* **xerGetMsgPtr** (ASN1CTXT \*pctxt)
- int **xerGetElemIdx** (ASN1Const XMLCHAR \*elemName, XerElemInfo \*pElemInfo, int numElems)
- int **xerGetSeqElemIdx** (ASN1Const XMLCHAR \*elemName, XerElemInfo \*pElemInfo, int numElems, int startIndex)
- int **xerFinalizeMemBuf** (ASN1MemBuf \*pMemBuf)
- int **xerGetLibVersion** ()
- const char \* **xerGetLibInfo** ()
- int **xmlDecBitStr** (ASN1CTXT \*pctxt, ASN1OCTET \*pvalue, ASN1UINT \*pnbits, ASN1INT bufsize)
- int **xmlDecBool** (ASN1CTXT \*pctxt, ASN1BOOL \*pvalue)
- int **xmlDecDynBitStr** (ASN1CTXT \*pctxt, ASN1DynBitStr \*pvalue)
- int **xmlDecDynNamedBitStr** (ASN1CTXT \*pctxt, ASN1DynBitStr \*pvalue, ASN1Const XmlNamedBitsDict \*pBitDict)
- int **xmlDecDynOctStr** (ASN1CTXT \*pctxt, ASN1DynOctStr \*pvalue)
- int **xmlDecGeneralizedTime** (ASN1CTXT \*pctxt, ASN1ConstCharPtr \*outdata)
- int **xmlDecNamedBitStr** (ASN1MemBuf \*pMemBuf, ASN1OCTET \*pData, int dataSize, ASN1UINT \*pNumbits, ASN1Const XmlNamedBitsDict \*pBitDict, ASN1Const XMLCHAR \*chars, int length)
- int **xmlDecOctStr** (ASN1CTXT \*pctxt, ASN1OCTET \*pvalue, ASN1UINT \*pnocts, ASN1INT bufsize)
- int **xmlDecReal** (ASN1CTXT \*pctxt, ASN1REAL \*pvalue)
- int **xmlDecUTCTime** (ASN1CTXT \*pctxt, ASN1ConstCharPtr \*outdata)
- int **xmlEncBitStr** (ASN1CTXT \*pctxt, XmlNamedBitsDict \*namedbits, ASN1UINT noofnamedbits, ASN1UINT nbits, ASN1ConstOctetPtr data, ASN1ConstCharPtr elemName, ASN1StrType outputType)
- int **xmlEncBoolValue** (ASN1CTXT \*pctxt, ASN1BOOL value)
- int **xmlEncBool** (ASN1CTXT \*pctxt, ASN1BOOL value, ASN1ConstCharPtr elemName)
- int **xmlEncEnum** (ASN1CTXT \*pctxt, ASN1ConstCharPtr value)
- int **xmlEncGeneralizedTime** (ASN1CTXT \*pctxt, ASN1ConstCharPtr value, ASN1ConstCharPtr elemName)
- int **xmlEncNamedValue** (ASN1CTXT \*pctxt, ASN1ConstCharPtr value, ASN1ConstCharPtr elemName, ASN1ConstCharPtr attributes)
- int **xerEncOpenTypeExt** (ASN1CTXT \*pctxt, Asn1RTDList \*pElemList)
- int **xmlEncReal** (ASN1CTXT \*pctxt, ASN1REAL value, ASN1ConstCharPtr elemName)
- int **xmlEncUTCTime** (ASN1CTXT \*pctxt, ASN1ConstCharPtr value, ASN1ConstCharPtr elemName)

## asn1XerCppTypes.h File Reference

---

### Detailed Description

XER C++ type and class definitions.

```
#include "rtxersrc/saxParser.h"
#include "rtsrc/asn1CppTypes.h"
#include "rtsrc/OSCInputStream.h"
#include "rtxersrc/asn1xer.h"
```

### Classes

- class **ASN1XERMessageBuffer**
- class **ASN1XEREncodeBuffer**
- class **ASN1XERSAXDecodeHandler**
- struct **ASN1XERSAXDecodeHandler::ErrorInfo**
- class **ASN1XERDecodeBuffer**

### Defines

- #define **ASN1SAXTHROW**(stat) throwSAXException (stat);
- #define **ASN1SAXCATCH0**(toCatch, stat)
- #define **ASN1SAXCATCH**(toCatch, stat)

# asn1XerCTypes.h File Reference

---

## Detailed Description

```
#include <stdio.h>
#include <stdlib.h>
#include <setjmp.h>
#include "asn1xer.h"
#include "csaxParser.h"
```

## Classes

- struct **ASN1SAXCDecodeHandlerBase**

## Defines

- #define **ISCOMPLETE**(e) (e->mSaxBase.mLevel == e->mSaxBase.mStartLevel)
- #define **ASN1SAXCTRY**(pctxt, stat) if ((stat = setjmp((pctxt)->jmpMark)) == 0)
- #define **ASN1SAXCTHROW**(pctxt, stat)
- #define **ASN1SAXCCATCH** else
- #define **STRX**(pctxt, pWideStr) xerTextToCStr ((pctxt), (pWideStr))
- #define **LSTRX**(pctxt, pLStr) strcpy ((char\*)ASN1MALLOC (pctxt, strlen (pLStr) + 1), (pLStr))
- #define **XERCDIAGSTRM2**(pctxt, a) RTDIAGSTRM2(pctxt,a)
- #define **XERCDIAGSTRM3**(pctxt, a, b) RTDIAGSTRM3(pctxt,a,b)
- #define **XERCDIAGSTRM4**(pctxt, a, b, c)
- #define **DECLARE\_NON\_COMPACT\_VAR**(type, var) type var

## Typedefs

- typedef void(\* **ASN1XERStartElementHandler** )(void \*userData, ASN1Const XMLCHAR \*name, ASN1Const XMLCHAR \*\*atts)
- typedef void(\* **ASN1XEREndElementHandler** )(void \*userData, ASN1Const XMLCHAR \*name)
- typedef void(\* **ASN1XERCharacterHandler** )(void \*userData, ASN1Const XMLCHAR \*s, int len)
- typedef ASN1SAXCDecodeHandlerBase **ASN1SAXCDecodeHandlerBase**

---

## Define Documentation

### #define ASN1SAXCTHROW(pctxt, stat)

```
Value:do { LOG_ASN1ERR ((pctxt), stat); \
longjmp((pctxt)->jmpMark, stat); } while (0)
```

### #define XERCDIAGSTRM2(pctxt, a) RTDIAGSTRM2(pctxt,a)

VP 6/02/04

### #define XERCDIAGSTRM4(pctxt, a, b, c)

```
Value:do { \
void* p = (void*)b; \
RTDIAGSTRM4 (pctxt,a,p,c ); \
ASN1MEMFREEPTR (pctxt,p); \
} while (0)
```



## ASN1XERInputStream.h File Reference

---

### Detailed Description

The C++ definitions for ASN.1 XER input streams.

```
#include "asn1XerCppType.h"  
#include "ASN1Stream.h"  
#include "OSCSocket.h"
```

### Classes

- class **ASN1XERInputStream**
- class **ASN1XERFileInputStream**
- class **ASN1XERSocketInputStream**

## ASN1XEROutputStream.h File Reference

---

### Detailed Description

The C++ definitions for ASN.1 XER output streams.

```
#include "asn1XerCppType.h"  
#include "ASN1Stream.h"  
#include "OSCSocket.h"
```

### Classes

- class **ASN1XEROutputStream**
- class **ASN1XERFileOutputStream**
- class **ASN1XERSocketOutputStream**

# Index

INDEX