

ASN1C

ASN.1 Compiler
Version 5.8
C/C++ PER 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.

CHANGE HISTORY

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

Table of Contents

ASN1C PER RUNTIME CLASSES AND LIBRARY FUNCTIONS	3
MODULE DOCUMENTATION	4
PER C++ RUNTIME CLASSES.....	4
PER MESSAGE BUFFER CLASSES.....	4
PER RUNTIME LIBRARY FUNCTIONS.....	4
PER C DECODE FUNCTIONS.....	7
PER C ENCODE FUNCTIONS.....	21
PER C UTILITY FUNCTIONS.....	35
CLASS DOCUMENTATION	42
ASN1PERDECODEBUFFER.....	42
ASN1PERENCODEBUFFER.....	44
ASN1PERMESSAGEBUFFER.....	46
FILE DOCUMENTATION	49
ASN1PER.H	49
ASN1PERCPPTYPES.H.....	54
INDEX.....	55

ASN1C PER 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 Packed Encoding Rules (PER) as defined in the X.691 ITU-T standard. These PER specific C++ runtime classes include the PER message buffer classes.

The **ASN.1 PER 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 ("asn1per.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 PER Runtime Module Documentation

PER C++ Runtime Classes.

Modules

- PER Message Buffer Classes

PER Message Buffer Classes

Detailed Description

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

Classes

- class **ASN1PERMessageBuffer**
- class **ASN1PEREncodeBuffer**
- class **ASN1PERDecodeBuffer**

PER Runtime Library Functions.

Detailed Description

The ASN.1 Packed Encoding Rules (PER) runtime library contains the low-level constants, types, and functions that are assembled by the compiler to encode/decode more complex structures. The PER low-level C encode/decode functions are identified by their prefixes: `pe_` for PER encode, `pd_` for PERdecode, and `pu_` for PER utility functions.

Modules

- PER C Decode Functions.
- PER C Encode Functions.
- PER C Utility Functions

Classes

- struct **PERField**
- struct **BinDumpBuffer**

Defines

- `#define ASN_K_EXTENUM 999`
- `#define PU_SETCHARSET(csetvar, canset, abits, ubits)`
- `#define PU_INSLENFLD(ctxt_p)`
- `#define PU_NEWFIELD(ctxt_p, suffix)`
- `#define PU_PUSHNAME(ctxt_p, name)`

- #define **PU_POPNAME**(ctxt_p)
- #define **PU_SETBITOFFSET**(ctxt_p)
- #define **PU_SETBITCOUNT**(ctxt_p)
- #define **PU_PUSHELEMNAME**(ctxt_p, idx)
- #define **EXTERNPER**
- #define **PD_INCRBITIDX**(ctxt_p)
- #define **PD_BIT**(ctxt_p, pvalue)
- #define **PU_GETSIZECONSTRAINT**(ctxt_p, extbit, pSize)
- #define **PU_GETCTXTBITOFFSET**(ctxt_p) (((ctxt_p)->buffer.byteIndex * 8) + (8 - (ctxt_p)->buffer.bitOffset))
- #define **PU_SETCTXTBITOFFSET**(ctxt_p, _bitOffset)
- #define **PD_BYTE_ALIGN0**(ctxt_p)
- #define **PD_BYTE_ALIGN** PD_BYTE_ALIGN0
- #define **PD_CHECKSEQOFLN**(pctxt, numElements, minElemBits)
- #define **pe_GeneralString**(ctxt_p, value, permCharSet) pe_VarWidthCharString(ctxt_p, value)
- #define **pe_GraphicString**(ctxt_p, value, permCharSet) pe_VarWidthCharString(ctxt_p, value)
- #define **pe_T61String**(ctxt_p, value, permCharSet) pe_VarWidthCharString(ctxt_p, value)
- #define **pe_TeletexString**(ctxt_p, value, permCharSet) pe_VarWidthCharString(ctxt_p, value)
- #define **pe_VideotexString**(ctxt_p, value, permCharSet) pe_VarWidthCharString(ctxt_p, value)
- #define **pe_ObjectDescriptor**(ctxt_p, value, permCharSet) pe_VarWidthCharString(ctxt_p, value)
- #define **pe_UTF8String**(ctxt_p, value, permCharSet) pe_VarWidthCharString(ctxt_p, value)
- #define **pe_IA5String**(ctxt_p, value, permCharSet) pe_ConstrainedStringEx (ctxt_p, value, permCharSet, 8, 7, 7)
- #define **pe_NumericString**(ctxt_p, value, permCharSet)
- #define **pe_PrintableString**(ctxt_p, value, permCharSet) pe_ConstrainedStringEx (ctxt_p, value, permCharSet, 8, 7, 7)
- #define **pe_VisibleString**(ctxt_p, value, permCharSet)
- #define **pe_ISO646String** pe_IA5String
- #define **pe_GeneralizedTime** pe_IA5String
- #define **pe_UTCTime** pe_GeneralizedTime
- #define **pd_GeneralString**(ctxt_p, pvalue, permCharSet) pd_VarWidthCharString (ctxt_p, pvalue)
- #define **pd_GraphicString**(ctxt_p, pvalue, permCharSet) pd_VarWidthCharString (ctxt_p, pvalue)
- #define **pd_VideotexString**(ctxt_p, pvalue, permCharSet) pd_VarWidthCharString (ctxt_p, pvalue)
- #define **pd_TeletexString**(ctxt_p, pvalue, permCharSet) pd_VarWidthCharString (ctxt_p, pvalue)
- #define **pd_T61String**(ctxt_p, pvalue, permCharSet) pd_VarWidthCharString (ctxt_p, pvalue)
- #define **pd_ObjectDescriptor**(ctxt_p, pvalue, permCharSet) pd_VarWidthCharString (ctxt_p, pvalue)
- #define **pd_UTF8String**(ctxt_p, pvalue, permCharSet) pd_VarWidthCharString (ctxt_p, pvalue)
- #define **pd_IA5String**(ctxt_p, pvalue, permCharSet) pd_ConstrainedStringEx (ctxt_p, pvalue, permCharSet, 8, 7, 7)
- #define **pd_NumericString**(ctxt_p, pvalue, permCharSet)
- #define **pd_PrintableString**(ctxt_p, pvalue, permCharSet) pd_ConstrainedStringEx (ctxt_p, pvalue, permCharSet, 8, 7, 7)
- #define **pd_VisibleString**(ctxt_p, pvalue, permCharSet) pd_ConstrainedStringEx (ctxt_p, pvalue, permCharSet, 8, 7, 7)
- #define **pd_ISO646String** pd_IA5String
- #define **pd_GeneralizedTime** pd_IA5String
- #define **pd_UTCTime** pd_GeneralizedTime
- #define **pe_GetMsgLen** pu_getMsgLen

Typedefs

- typedef PERField **PERField**

Define Documentation

#define PD_BIT(ctxt_p, pvalue)

```
Value: ((PD_INCRBITIDX (ctxt_p) != ASN_OK) ? ASN_E_ENDOFBUF : ((pvalue) ? \
((*(pvalue) = (ASN1OCTET)((!(ctxt_p)->buffer.data[(ctxt_p)->buffer.byteIndex]) & \
(1 << (ctxt_p)->buffer.bitOffset)) != 0)), ASN_OK) : ASN_OK )
```

#define PD_BYTE_ALIGN0(ctxt_p)

```
Value: (!(ctxt_p)->buffer.aligned) ? ASN_OK : \
(((ctxt_p)->buffer.bitOffset != 8) ? ( \
(ctxt_p)->buffer.byteIndex++, \
(ctxt_p)->buffer.bitOffset = 8, \
ASN_OK) : ASN_OK \
))
```

#define PD_CHECKSEQOFLEN(pctxt, numElements, minElemBits)

```
Value: ((pctxt->buffer.size > 0) ? \
((numElements * minElemBits) > (pctxt->buffer.size * 8)) ? \
ASN_E_INVLEN : ASN_OK) : ASN_OK
```

#define PD_INCRBITIDX(ctxt_p)

```
Value: ((--(ctxt_p)->buffer.bitOffset < 0) ? \
(++(ctxt_p)->buffer.byteIndex >= (ctxt_p)->buffer.size) ? ASN_E_ENDOFBUF : \
((ctxt_p)->buffer.bitOffset = 7, ASN_OK)) : ASN_OK
```

#define pd_NumericString(ctxt_p, pvalue, permCharSet)

```
Value: pd_ConstrainedStringEx (ctxt_p, pvalue, \
(permCharSet == 0)?NUM_CANSET:permCharSet, 4, 4, 4)
```

#define pe_GeneralString(ctxt_p, value, permCharSet) pe_VarWidthCharString(ctxt_p, value)

perutil

#define pe_NumericString(ctxt_p, value, permCharSet)

```
Value: pe_ConstrainedStringEx (ctxt_p, value, \
(permCharSet == 0)?NUM_CANSET:permCharSet, 4, 4, 4)
```

#define PU_GETSIZECONSTRAINT(ctxt_p, extbit, pSize)

```
Value: (((pSize) = ctxt_p->pSizeConstraint) != 0) ? \
(((pSize) = ctxt_p->pSizeConstraint)->extended == extbit)?(pSize) : \
pu_getSizeConstraint (ctxt_p, extbit) : NULL
```

#define PU_SETCHARSET(csetvar, canset, abits, ubits)

```
Value: csetvar.charSet.nchars = 0; \
csetvar.canonicalSet = canset; \
csetvar.canonicalSetSize = sizeof(canset)-1; \
csetvar.canonicalSetBits = pu_bitcnt(csetvar.canonicalSetSize); \
csetvar.charSetUnalignedBits = ubits; \
csetvar.charSetAlignedBits = abits;
```

#define PU_SETCTXTBITOFFSET(ctxt_p, _bitOffset)

```
Value: do { \
(ctxt_p)->buffer.byteIndex = ( bitOffset / 8); \
(ctxt_p)->buffer.bitOffset = (ASN1USINT)(8 - ( bitOffset % 8)); \
} while(0)
```

PER C Decode Functions.

Detailed Description

PER 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 item exists.

The procedure to decode a primitive data item is as follows:

1. Call the `rtInitContext` function to initialize a context.
2. Call `pu_setBuffer` to specify the address of the buffer containing the encoded ASN.1 data to be decoded and whether the data is aligned, or unaligned.
3. Call the specific decode function to decode the value.

Defines

- `#define pd_UnconsInteger(ctxt_p, pvalue) pd_SemiConsInteger(ctxt_p, pvalue, ASN1INT_MIN)`
- `#define pd_UnconsUnsigned(ctxt_p, pvalue) pd_SemiConsUnsigned(ctxt_p, pvalue, 0U)`
- `#define pd_UnconsInt64(ctxt_p, pvalue) pd_SemiConsInt64(ctxt_p, pvalue, ASN1INT64MIN)`
- `#define pd_UnconsUInt64(ctxt_p, pvalue) pd_SemiConsUInt64(ctxt_p, pvalue, 0)`

Functions

- `int pd_bit (ASN1CTXT *ctxt_p, ASN1BOOL *pvalue)`
- `int pd_bits (ASN1CTXT *ctxt_p, ASN1UINT *pvalue, ASN1UINT nbits)`
- `int pd_BigInteger (ASN1CTXT *ctxt_p, ASN1ConstCharPtr *ppvalue)`
- `int pd_BitString (ASN1CTXT *ctxt_p, ASN1UINT *numbits_p, ASN1OCTET *buffer, ASN1UINT bufsiz)`
- `int pd_BMPString (ASN1CTXT *ctxt_p, ASN1BMPString *pvalue, Asn116BitCharSet *permCharSet)`
- `int pd_UniversalString (ASN1CTXT *ctxt_p, ASN1UniversalString *pvalue, Asn132BitCharSet *permCharSet)`
- `int pd_byte_align (ASN1CTXT *ctxt_p)`
- `int pd_ChoiceOpenTypeExt (ASN1CTXT *ctxt_p, ASN1ConstOctetPtr *object_p2, ASN1UINT *numocts_p)`
- `int pd_ConInteger (ASN1CTXT *ctxt_p, ASN1INT *pvalue, ASN1INT lower, ASN1INT upper)`
- `int pd_ConInt8 (ASN1CTXT *ctxt_p, ASN1INT8 *pvalue, ASN1INT lower, ASN1INT upper)`
- `int pd_ConInt16 (ASN1CTXT *ctxt_p, ASN1SINT *pvalue, ASN1INT lower, ASN1INT upper)`
- `int pd_ConInt64 (ASN1CTXT *ctxt_p, ASN1INT64 *pvalue, ASN1INT64 lower, ASN1INT64 upper)`
- `int pd_ConUnsigned (ASN1CTXT *ctxt_p, ASN1UINT *pvalue, ASN1UINT lower, ASN1UINT upper)`
- `int pd_ConUInt8 (ASN1CTXT *ctxt_p, ASN1UINT8 *pvalue, ASN1UINT lower, ASN1UINT upper)`
- `int pd_ConUInt16 (ASN1CTXT *ctxt_p, ASN1USINT *pvalue, ASN1UINT lower, ASN1UINT upper)`
- `int pd_ConUInt64 (ASN1CTXT *ctxt_p, ASN1UINT64 *pvalue, ASN1UINT64 lower, ASN1UINT64 upper)`
- `int pd_ConWholeNumber (ASN1CTXT *ctxt_p, ASN1UINT *padjusted_value, ASN1UINT range_value)`

- int **pd_ConsWholeNumber64** (ASN1CTXT *ctxt_p, ASN1UINT64 *padjusted_value, ASN1UINT64 range_value)
- int **pd_ConstrainedString** (ASN1CTXT *ctxt_p, ASN1ConstCharPtr *string, Asn1CharSet *pCharSet)
- int **pd_ConstrainedStringEx** (ASN1CTXT *ctxt_p, ASN1ConstCharPtr *string, ASN1ConstCharPtr charSet, ASN1UINT abits, ASN1UINT ubits, ASN1UINT canSetBits)
- int **pd_16BitConstrainedString** (ASN1CTXT *ctxt_p, Asn116BitCharString *pString, Asn116BitCharSet *pCharSet)
- int **pd_32BitConstrainedString** (ASN1CTXT *ctxt_p, Asn132BitCharString *pString, Asn132BitCharSet *pCharSet)
- int **pd_DynBitString** (ASN1CTXT *ctxt_p, ASN1DynBitStr *pBitStr)
- int **pd_DynOctetString** (ASN1CTXT *ctxt_p, ASN1DynOctStr *pOctStr)
- int **pd_GetComponentLength** (ASN1CTXT *ctxt_p, ASN1UINT itemBits)
- int **pd_Length** (ASN1CTXT *ctxt_p, ASN1UINT *pvalue)
- int **pd_moveBitCursor** (ASN1CTXT *ctxt_p, int bitOffset)
- int **pd_ObjectIdentifier** (ASN1CTXT *ctxt_p, ASN1OBJID *pvalue)
- int **pd_oid64** (ASN1CTXT *ctxt_p, ASN1OID64 *pvalue)
- int **pd_RelativeOID** (ASN1CTXT *ctxt_p, ASN1OBJID *pvalue)
- int **pd_OctetString** (ASN1CTXT *ctxt_p, ASN1UINT *numocts_p, ASN1OCTET *buffer, ASN1UINT bufsiz)
- int **pd_OpenType** (ASN1CTXT *ctxt_p, ASN1ConstOctetPtr *object_p2, ASN1UINT *numocts_p)
- int **pd_OpenTypeExt** (ASN1CTXT *ctxt_p, ASN1ConstOctetPtr *object_p2, ASN1UINT *numocts_p)
- int **pd_Real** (ASN1CTXT *ctxt_p, ASN1REAL *pvalue)
- int **pd_SmallNonNegWholeNumber** (ASN1CTXT *ctxt_p, ASN1UINT *pvalue)
- int **pd_SemiConsInteger** (ASN1CTXT *ctxt_p, ASN1INT *pvalue, ASN1INT lower)
- int **pd_SemiConsUnsigned** (ASN1CTXT *ctxt_p, ASN1UINT *pvalue, ASN1UINT lower)
- int **pd_SemiConsInt64** (ASN1CTXT *ctxt_p, ASN1INT64 *pvalue, ASN1INT64 lower)
- int **pd_SemiConsUInt64** (ASN1CTXT *ctxt_p, ASN1UINT64 *pvalue, ASN1UINT64 lower)
- int **pd_VarWidthCharString** (ASN1CTXT *pctxt, ASN1ConstCharPtr *pvalue)

Define Documentation

#define pd_UnconsInt64(ctxt_p, pvalue) pd_SemiConsInt64(ctxt_p, pvalue, ASN1INT64MIN)

This function will decode an unconstrained 64-bit integer.

Parameters:

ctxt_p Pointer to context block structure.

pvalue Pointer to 64-bit integer variable to receive decoded value.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

#define pd_UnconsInteger(ctxt_p, pvalue) pd_SemiConsInteger(ctxt_p, pvalue, ASN1INT_MIN)

This function will decode an unconstrained integer.

Parameters:

ctxt_p Pointer to context block structure.

pvalue Pointer to integer variable to receive decoded value.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

#define pd_UnconsUInt64(ctxt_p, pvalue) pd_SemiConsUInt64(ctxt_p, pvalue, 0)

This function will decode an unconstrained unsigned 64-bit integer.

Parameters:

ctxt_p Pointer to context block structure.

pvalue Pointer to unsigned 64-bit integer variable to receive decoded value.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

#define pd_UnconsUnsigned(ctxt_p, pvalue) pd_SemiConsUnsigned(ctxt_p, pvalue, 0U)

This function will decode an unconstrained unsigned integer.

Parameters:

ctxt_p Pointer to context block structure.

pvalue Pointer to unsigned integer variable to receive decoded value.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

Function Documentation

int pd_16BitConstrainedString (ASN1CTXT * ctxt_p, Asn116BitCharString * pString, Asn116BitCharSet * pCharSet)

This function will encode a constrained ASN.1 character string. This function is normally not called directly but rather is called from Useful Type Character String encode functions that deal with 16-bit strings. The only function that does not release is the pe_BMPString function.

Parameters:

ctxt_p 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.

pString Character string to be encoded. The structure includes a count field containing the number of characters to encode and an array of unsigned short integers to hold the 16-bit characters to be encoded.

pCharSet Pointer to the constraining character set. This contains an array containing all valid characters in the set as well as the aligned and unaligned bit counts required to encode the characters.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_32BitConstrainedString (ASN1CTXT * *ctxt_p*, Asn132BitCharString * *pString*, Asn132BitCharSet * *pCharSet*)

This function will encode a constrained ASN.1 character string. This function is normally not called directly but rather is called from Useful Type Character String encode functions that deal with 32-bit strings. The only function that does not release is the `pe_UniversalString` function.

Parameters:

- ctxt_p* 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.
- pString* Character string to be encoded. The structure includes a count field containing the number of characters to encode and an array of unsigned short integers to hold the 32-bit characters to be encoded.
- pCharSet* Pointer to the constraining character set. This contains an array containing all valid characters in the set as well as the aligned and unaligned bit counts required to encode the characters.

Returns:

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

int pd_BigInteger (ASN1CTXT * *ctxt_p*, ASN1ConstCharPtr * *ppvalue*)

This function decodes 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. They are represented as hexadecimal strings starting with "0x" prefix. If it is necessary to convert a hexadecimal string to another radix, then use the `rtBigIntSetStr / rtBigIntToString` functions.

Parameters:

- ctxt_p* 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 decimal string starting with no prefix.

Returns:

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

int pd_bit (ASN1CTXT * *ctxt_p*, ASN1BOOL * *pvalue*)

This function will decode a single bit and place the result in an ASN.1 BOOLEAN type variable.

Parameters:

- ctxt_p* 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 the BOOLEAN value to receive the decoded result.

Returns:

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

int pd_bits (ASN1CTXT * *ctxt_p*, ASN1UINT * *pvalue*, ASN1UINT *nbits*)

This function will decode a series of multiple bits and place the results in an unsigned integer variable.

Parameters:

ctxt_p 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 unsigned integer variable to receive the decoded result.
nbits The number of bits to decode.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_BitString (ASN1CTXT * *ctxt_p*, ASN1UINT * *numbits_p*, ASN1OCTET * *buffer*, ASN1UINT *bufsiz*)

This function will decode a value of the ASN.1 bit string type whose maximum size is known in advance. The ASN1C compiler generates a call to this function to decode bit string productions or elements that contain a size constraint.

Parameters:

ctxt_p 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.
numbits_p Pointer to an unsigned integer variable to receive decoded number of bits.
buffer Pointer to a fixed-size or pre-allocated array of *bufsiz* octets to receive a decoded bit string.
bufsiz Length (in octets) of the buffer to receive the decoded bit string.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_BMPString (ASN1CTXT * *ctxt_p*, ASN1BMPString * *pvalue*, Asn116BitCharSet * *permCharSet*)

This function will decode a variable of the ASN.1 BMP character string. This differs from the decode routines for the character strings previously described in that the BMP string type is based on 16-bit characters. A 16-bit character string is modeled using an array of unsigned short integers.

Parameters:

ctxt_p 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 Pointer to character string structure to receive the decoded result The structure includes a count field containing the number of characters and an array of unsigned short integers to hold the 16-bit character values.
permCharSet A pointer to the constraining character set. This contains an array containing all valid characters in the set as well as the aligned and unaligned bit counts required to encode the characters.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_byte_align (ASN1CTXT * *ctxt_p*)

This function will position the decode bit cursor on the next byte boundary.

Parameters:

ctxt_p 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 pd_ChoiceOpenTypeExt (ASN1CTXT * *ctxt_p*, ASN1ConstOctetPtr * *object_p2*, ASN1UINT * *numocts_p*)

Parameters:

ctxt_p 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.

object_p2 A pointer to an open type variable to receive the decoded data.

numocts_p A pointer to an unsigned buffer of bufsiz octets to receive decoded data.

int pd_ConstInt16 (ASN1CTXT * *ctxt_p*, ASN1SINT * *pvalue*, ASN1INT *lower*, ASN1INT *upper*)

This function will decode a 16-bit integer constrained either by a value or value range constraint.

Parameters:

ctxt_p Pointer to context block structure.

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

lower Lower range value.

upper Upper range value.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_ConstInt64 (ASN1CTXT * *ctxt_p*, ASN1INT64 * *pvalue*, ASN1INT64 *lower*, ASN1INT64 *upper*)

This function will decode a 64-bit integer constrained either by a value or value range constraint.

Parameters:

ctxt_p Pointer to context block structure.

pvalue Pointer to 64-bit integer variable to receive decoded value.

lower Lower range value, represented as 64-bit integer.

upper Upper range value, represented as 64-bit integer.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_ConstInt8 (ASN1CTXT * *ctxt_p*, ASN1INT8 * *pvalue*, ASN1INT *lower*, ASN1INT *upper*)

This function will decode an 8-bit integer constrained either by a value or value range constraint.

Parameters:

ctxt_p Pointer to context block structure.
pvalue Pointer to 8-bit integer variable to receive decoded value.
lower Lower range value.
upper Upper range value.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_ConstInteger (ASN1CTXT * *ctxt_p*, ASN1INT * *pvalue*, ASN1INT *lower*, ASN1INT *upper*)

This function will decode an integer constrained either by a value or value range constraint.

Parameters:

ctxt_p Pointer to context block structure.
pvalue Pointer to integer variable to receive decoded value.
lower Lower range value.
upper Upper range value.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_ConstrainedString (ASN1CTXT * *ctxt_p*, ASN1ConstCharPtr * *string*, Asn1CharSet * *pCharSet*)

This function decodes a constrained string value. This is a deprecated version of the function provided for backward compatibility.

Parameters:

ctxt_p Pointer to context block structure.
string Pointer to const char* to receive decoded string. Memory will be allocated for this variable using internal memory management functions.
pCharSet Pointer to a character set descriptor structure. This contains an array containing all valid characters in the set as well as the aligned and unaligned bit counts required to encode the characters.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_ConstrainedStringEx (ASN1CTXT * *ctxt_p*, ASN1ConstCharPtr * *string*, ASN1ConstCharPtr *charSet*, ASN1UINT *abits*, ASN1UINT *ubits*, ASN1UINT *canSetBits*)

This function decodes a constrained string value. This version of the function allows all of the required permitted alphabet constraint parameters to be passed in as arguments.

Parameters:

ctxt_p Pointer to context block structure.

string Pointer to const char* to receive decoded string. Memory will be allocated for this variable using internal memory management functions.

charSet String containing permitted alphabet character set. Can be null if no character set was specified.

abits Number of bits in a character set character (aligned).

ubits Number of bits in a character set character (unaligned).

canSetBits Number of bits in a character from the canonical set representing this string.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_ConstUInt16 (ASN1CTXT * *ctxt_p*, ASN1USINT * *pvalue*, ASN1UINT *lower*, ASN1UINT *upper*)

This function will decode a 16-bit unsigned integer constrained either by a value or value range constraint.

Parameters:

ctxt_p Pointer to context block structure.

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

lower Lower range value.

upper Upper range value.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_ConstUInt64 (ASN1CTXT * *ctxt_p*, ASN1UINT64 * *pvalue*, ASN1UINT64 *lower*, ASN1UINT64 *upper*)

This function will decode a 64-bit unsigned integer constrained either by a value or value range constraint.

Parameters:

ctxt_p Pointer to context block structure.

pvalue Pointer to 64-bit unsigned integer variable to receive decoded value.

lower Lower range value.

upper Upper range value.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_ConstUInt8 (ASN1CTXT * *ctxt_p*, ASN1UINT8 * *pvalue*, ASN1UINT *lower*, ASN1UINT *upper*)

This function will decode an 8-bit unsigned integer constrained either by a value or value range constraint.

Parameters:

ctxt_p Pointer to context block structure.

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

lower Lower range value.

upper Upper range value.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_ConsUnsigned (ASN1CTXT * *ctxt_p*, ASN1UINT * *pvalue*, ASN1UINT *lower*, ASN1UINT *upper*)

This function will decode an unsigned integer constrained either by a value or value range constraint.

Parameters:

ctxt_p Pointer to context block structure.

pvalue Pointer to unsigned integer variable to receive decoded value.

lower Lower range value.

upper Upper range value.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_ConsWholeNumber (ASN1CTXT * *ctxt_p*, ASN1UINT * *padjusted_value*, ASN1UINT *range_value*)

This function decodes a constrained whole number as specified in Section 10.5 of the X.691 standard.

Parameters:

ctxt_p Pointer to context block structure.

adjusted_value Pointer to unsigned adjusted integer value to receive decoded result. To get the final value, this value is added to the lower boundary of the range.

range_value Unsigned integer value specifying the total size of the range. This is obtained by subtracting the lower range value from the upper range value.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_ConsWholeNumber64 (ASN1CTXT * *ctxt_p*, ASN1UINT64 * *padjusted_value*, ASN1UINT64 *range_value*)

This function decodes a constrained whole number as specified in Section 10.5 of the X.691 standard, represented as 64-bit integer.

Parameters:

ctxt_p Pointer to context block structure.

adjusted_value Pointer to 64-bit unsigned adjusted integer value to receive decoded result. To get the final value, this value is added to the lower boundary of the range.

range_value Unsigned 64-bit integer value specifying the total size of the range. This is obtained by subtracting the lower range value from the upper range value.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_DynBitString (ASN1CTXT * *ctxt_p*, ASN1DynBitStr * *pBitStr*)

This function will decode a variable of the ASN.1 BIT STRING type. This function allocates dynamic memory to store the decoded result. The ASN1C compiler generates a call to this function to decode an unconstrained bit string production or element.

Parameters:

ctxt_p 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.
pBitStr Pointer to a dynamic bit string structure to receive the decoded result. This structure contains a field to hold the number of decoded bits and a pointer to an octet string to hold the decoded data. Memory is allocated by the decoder using the `rtMemAlloc` function. This memory is tracked within the context and released when the `pu_freeContext` function is invoked.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_DynOctetString (ASN1CTXT * *ctxt_p*, ASN1DynOctStr * *pOctStr*)

This function will decode a value of the ASN.1 octet string type whose maximum size is known in advance. The ASN1C compiler generates a call to this function to decode octet string productions or elements that contain a size constraint.

Parameters:

ctxt_p 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.
pOctStr A pointer to a dynamic octet string to receive the decoded result.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_GetComponentLength (ASN1CTXT * *ctxt_p*, ASN1UINT *itemBits*)

Parameters:

ctxt_p 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.
itemBits The size of the specific entity.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_Length (ASN1CTXT * *ctxt_p*, ASN1UINT * *pvalue*)

This function will decode a length determinant value.

Parameters:

ctxt_p 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 unsigned integer variable to receive the decoded length value.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_moveBitCursor (ASN1CTXT * *ctxt_p*, int *bitOffset*)

Parameters:

ctxt_p 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.
bitOffset The bit offset inside the message buffer.

int pd_ObjectIdentifier (ASN1CTXT * *ctxt_p*, ASN1OBJID * *pvalue*)

This function decodes a value of the ASN.1 object identifier type.

Parameters:

ctxt_p 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 pd_OctetString (ASN1CTXT * *ctxt_p*, ASN1UINT * *numocts_p*, ASN1OCTET * *buffer*, ASN1UINT *bufsiz*)

This function will decode a value of the ASN.1 octet string type whose maximum size is known in advance. The ASN1C compiler generates a call to this function to decode octet string productions or elements that contain a size constraint.

Parameters:

ctxt_p 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.
numocts_p A pointer to an unsigned buffer of *bufsiz* octets to receive decoded data.
buffer A pointer to a pre-allocated buffer of size *octets* to receive the decoded data.
bufsiz The size of the buffer to receive the decoded result.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_oid64 (ASN1CTXT * *ctxt_p*, ASN1OID64 * *pvalue*)

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

Parameters:

ctxt_p 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 pd_OpenType (ASN1CTXT * *ctxt_p*, ASN1ConstOctetPtr * *object_p2*, ASN1UINT * *numocts_p*)

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:

ctxt_p 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.

numocts_p A pointer to an unsigned buffer of bufsiz octets to receive decoded data.

object_p2 A pointer to an open type variable to receive the decoded data.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_OpenTypeExt (ASN1CTXT * *ctxt_p*, ASN1ConstOctetPtr * *object_p2*, ASN1UINT * *numocts_p*)

This function will decode an ASN.1 open type extension. These are extra fields in a version-2 message that may be present after the ... extension marker. An open type structure (extElem1) is added to a message structure that contains an extension marker but no extension elements. The pd_OpenTypeExt function will populate this structure with the complete extension information (optional bit or choice index, length and data). A subsequent call to pe_OpenTypeExt will cause the saved extension fields to be included in a newly encoded message of the given type.

Parameters:

ctxt_p 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.

object_p2 A pointer to an open type variable to receive the decoded data.

numocts_p A pointer to an unsigned buffer of bufsiz octets to receive decoded data.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_Real (ASN1CTXT * *ctxt_p*, ASN1REAL * *pvalue*)

This function will encode a value of the ASN.1 real type. This function provides support for the plus-infinity special real values. Use the rtGetPlusInfinity or the rtGetMinusInfinity functions to get these special values.

Parameters:

ctxt_p Pointer to a context structure. This provides a storage area for the function to store all workings variables that must be maintained between function calls.

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

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_RelativeOID (ASN1CTXT * *ctxt_p*, ASN1OBJID * *pvalue*)

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

Parameters:

ctxt_p 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 pd_SemiConsInt64 (ASN1CTXT * *ctxt_p*, ASN1INT64 * *pvalue*, ASN1INT64 *lower*)

This function will decode a semi-constrained 64-bit integer.

Parameters:

ctxt_p Pointer to context block structure.

pvalue Pointer to 64-bit integer variable to receive decoded value.

lower Lower range value, represented as signed 64-bit integer.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_SemiConsInteger (ASN1CTXT * *ctxt_p*, ASN1INT * *pvalue*, ASN1INT *lower*)

This function will decode a semi-constrained integer.

Parameters:

ctxt_p Pointer to context block structure.

pvalue Pointer to integer variable to receive decoded value.

lower Lower range value, represented as signed integer.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_SemiConsUInt64 (ASN1CTXT * *ctxt_p*, ASN1UINT64 * *pvalue*, ASN1UINT64 *lower*)

This function will decode a semi-constrained unsigned 64-bit integer.

Parameters:

ctxt_p Pointer to context block structure.

pvalue Pointer to unsigned 64-bit integer variable to receive decoded value.

lower Lower range value, represented as unsigned 64-bit integer.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_SemiConsUnsigned (ASN1CTXT * *ctxt_p*, ASN1UINT * *pvalue*, ASN1UINT *lower*)

This function will decode a semi-constrained unsigned integer.

Parameters:

ctxt_p Pointer to context block structure.
pvalue Pointer to unsigned integer variable to receive decoded value.
lower Lower range value, represented as unsigned integer.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_SmallNonNegWholeNumber (ASN1CTXT * *ctxt_p*, ASN1UINT * *pvalue*)

This function will decode a small non-negative whole number as specified in Section 10.6 of the X.691 standard. This is a number that is expected to be small, but whose size is potentially unlimited due to the presence of an extension maker.

Parameters:

ctxt_p Pointer to a context structure. This provides a storage area for the function to store all workings variables that must be maintained between function calls.
pvalue Pointer to an unsigned integer value to receive decoded results.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_UniversalString (ASN1CTXT * *ctxt_p*, ASN1UniversalString * *pvalue*, Asn132BitCharSet * *permCharSet*)

This function will decode a variable of the ASN.1 32-bit character string. This differs from the decode routines for the character strings previously described because the universal string type is based on 32-bit characters. A 32-bit character string is modeled using an array of unsigned integers.

Parameters:

ctxt_p 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 Pointer to character string structure to receive the decoded result. The structure includes a count field containing the number of characters and an array of unsigned short integers to hold the 32-bit character values.
permCharSet A pointer to the constraining character set. This contains an array containing all valid characters in the set as well as the aligned and unaligned bit counts required to encode the characters.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pd_VarWidthCharString (ASN1CTXT * *pctxt*, ASN1ConstCharPtr * *pvalue*)**Parameters:**

pctxt Pointer to context block structure.
pvalue Pointer to unsigned 64-bit integer variable to receive decoded value.

PER C Encode Functions.

Detailed Description

The Per low-level encode functions handle the PER encoding of the 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 `pu_newContext` function must first be called to set a pointer to the buffer into which the variable is to be encoded. A static encode buffer is specified by assigning a pointer to a buffer and a 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, only be obtained by calling `pe_GetMsgPtr`. The length of the encoded compound is obtained by calling `pe_GetMsgLen`.

Defines

- `#define pe_UnconsInteger(ctxt_p, value) pe_SemiConsInteger(ctxt_p,value,ASN1INT_MIN)`
- `#define pe_UnconsInt64(ctxt_p, value) pe_SemiConsInt64(ctxt_p,value,ASN1INT64MIN)`
- `#define pe_UnconsUnsigned(ctxt_p, value) pe_SemiConsUnsigned(ctxt_p,value,0)`
- `#define pe_UnconsUInt64(ctxt_p, value) pe_SemiConsUInt64(ctxt_p,value,0)`

Functions

- `int pe_16BitConstrainedString (ASN1CTXT *ctxt_p, Asn116BitCharString value, Asn116BitCharSet *pCharSet)`
- `int pe_32BitConstrainedString (ASN1CTXT *ctxt_p, Asn132BitCharString value, Asn132BitCharSet *pCharSet)`
- `int pe_2sCompBinInt (ASN1CTXT *ctxt_p, ASN1INT value)`
- `int pe_2sCompBinInt64 (ASN1CTXT *ctxt_p, ASN1INT64 value)`
- `int pe_aligned_octets (ASN1CTXT *ctxt_p, ASN1OCTET *pvalue, ASN1UINT nocts)`
- `int pe_BigInteger (ASN1CTXT *ctxt_p, ASN1ConstCharPtr pvalue)`
- `int pe_bit (ASN1CTXT *ctxt_p, ASN1BOOL value)`
- `int pe_bits (ASN1CTXT *ctxt_p, ASN1UINT value, ASN1UINT nbits)`
- `int pe_bits64 (ASN1CTXT *ctxt_p, ASN1UINT64 value, ASN1UINT nbits)`
- `int pe_BitString (ASN1CTXT *ctxt_p, ASN1UINT numocts, ASN1ConstOctetPtr data)`
- `int pe_BMPString (ASN1CTXT *ctxt_p, ASN1BMPString value, Asn116BitCharSet *permCharSet)`
- `int pe_UniversalString (ASN1CTXT *ctxt_p, ASN1UniversalString value, Asn132BitCharSet *permCharSet)`
- `int pe_byte_align (ASN1CTXT *ctxt_p)`
- `int pe_CheckBuffer (ASN1CTXT *ctxt_p, size_t nbytes)`
- `int pe_ChoiceTypeExt (ASN1CTXT *ctxt_p, ASN1UINT numocts, ASN1ConstOctetPtr data)`
- `int pe_ConsInteger (ASN1CTXT *ctxt_p, ASN1INT value, ASN1INT lower, ASN1INT upper)`
- `int pe_ConsInt64 (ASN1CTXT *ctxt_p, ASN1INT64 value, ASN1INT64 lower, ASN1INT64 upper)`
- `int pe_ConstrainedString (ASN1CTXT *ctxt_p, ASN1ConstCharPtr string, Asn1CharSet *pCharSet)`

- int **pe_ConstrainedStringEx** (ASN1CTXT *ctxt_p, ASN1ConstCharPtr string, ASN1ConstCharPtr charSet, ASN1UINT abits, ASN1UINT ubits, ASN1UINT canSetBits)
- int **pe_ConsUnsigned** (ASN1CTXT *ctxt_p, ASN1UINT value, ASN1UINT lower, ASN1UINT upper)
- int **pe_ConsUInt64** (ASN1CTXT *ctxt_p, ASN1UINT64 value, ASN1UINT64 lower, ASN1UINT64 upper)
- int **pe_ConsWholeNumber** (ASN1CTXT *ctxt_p, ASN1UINT adjusted_value, ASN1UINT range_value)
- int **pe_ConsWholeNumber64** (ASN1CTXT *ctxt_p, ASN1UINT64 adjusted_value, ASN1UINT64 range_value)
- int **pe_ExpandBuffer** (ASN1CTXT *ctxt_p, size_t nbytes)
- ASN1UINT **pe_GetIntLen** (ASN1UINT value)
- size_t **pe_GetMsgBitCnt** (ASN1CTXT *ctxt_p)
- ASN1OCTET * **pe_GetMsgPtr** (ASN1CTXT *ctxt_p, int *pLength)
- ASN1OCTET * **pe_GetMsgPtrU** (ASN1CTXT *ctxt_p, ASN1UINT *pLength)
- int **pe_Length** (ASN1CTXT *ctxt_p, ASN1UINT value)
- int **pe_NonNegBinInt** (ASN1CTXT *ctxt_p, ASN1UINT value)
- int **pe_NonNegBinInt64** (ASN1CTXT *ctxt_p, ASN1UINT64 value)
- int **pe_ObjectIdentifier** (ASN1CTXT *ctxt_p, ASN1OBJID *pvalue)
- int **pe_oid64** (ASN1CTXT *ctxt_p, ASN1OID64 *pvalue)
- int **pe_RelativeOID** (ASN1CTXT *ctxt_p, ASN1OBJID *pvalue)
- int **pe_octets** (ASN1CTXT *ctxt_p, ASN1ConstOctetPtr pvalue, ASN1UINT nbits)
- int **pe_OctetString** (ASN1CTXT *ctxt_p, ASN1UINT numocts, ASN1ConstOctetPtr data)
- int **pe_OpenType** (ASN1CTXT *ctxt_p, ASN1UINT numocts, ASN1ConstOctetPtr data)
- int **pe_OpenTypeExt** (ASN1CTXT *ctxt_p, Asn1RTDList *pElemList)
- int **pe_OpenTypeExtBits** (ASN1CTXT *ctxt_p, Asn1RTDList *pElemList)
- int **pe_Real** (ASN1CTXT *ctxt_p, ASN1REAL value)
- int **pe_SmallNonNegWholeNumber** (ASN1CTXT *ctxt_p, ASN1UINT value)
- int **pe_SemiConsInteger** (ASN1CTXT *ctxt_p, ASN1INT value, ASN1INT lower)
- int **pe_SemiConsInt64** (ASN1CTXT *ctxt_p, ASN1INT64 value, ASN1INT64 lower)
- int **pe_SemiConsUnsigned** (ASN1CTXT *ctxt_p, ASN1UINT value, ASN1UINT lower)
- int **pe_SemiConsUInt64** (ASN1CTXT *ctxt_p, ASN1UINT64 value, ASN1UINT64 lower)
- int **pe_UnconsLength** (ASN1CTXT *ctxt_p, ASN1UINT value)
- int **pe_VarWidthCharString** (ASN1CTXT *pctxt, ASN1ConstCharPtr value)

Define Documentation

#define pe_UnconsInt64(ctxt_p, value) pe_SemiConsInt64(ctxt_p,value,ASN1INT64MIN)

This function encodes an unconstrained 64-bit integer.

Parameters:

ctxt_p Pointer to context block structure.

value Value to be encoded, represented as 64-bit integer.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

#define pe_UnconsInteger(ctxt_p, value) pe_SemiConsInteger(ctxt_p,value,ASN1INT_MIN)

This function encodes an unconstrained integer.

Parameters:

ctxt_p Pointer to context block structure.
value Value to be encoded.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

#define pe_UnconsUInt64(ctxt_p, value) pe_SemiConsUInt64(ctxt_p,value,0)

This function encodes an unconstrained unsigned 64-bit integer.

Parameters:

ctxt_p Pointer to context block structure.
value Value to be encoded, represented as unsigned 64-bit integer.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

#define pe_UnconsUnsigned(ctxt_p, value) pe_SemiConsUnsigned(ctxt_p,value,0)

This function encodes an unconstrained unsigned integer.

Parameters:

ctxt_p Pointer to context block structure.
value Value to be encoded.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

Function Documentation

int pe_16BitConstrainedString (ASN1CTXT * *ctxt_p*, Asn116BitCharString *value*, Asn116BitCharSet * *pCharSet*)

This function will encode a constrained ASN.1 character string. This function is normally not called directly but rather is called from Useful Type Character String encode functions that deal with 16-bit strings. The only function that does that in this release is the `pe_BMPString` function.

Parameters:

ctxt_p 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 Character string to be encoded. The structure includes a count field containing the number of characters to encode and an array of unsigned short integers to hold the 16-bit characters to be encoded.
pCharSet Pointer to the constraining character set. The contains an array containing all valid characters in the set as well as the aligned and unaligned bit counts required to encode the characters.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,

- negative return value is error.

int pe_2sCompBinInt (ASN1CTXT * *ctxt_p*, ASN1INT *value*)

This function encodes a two's complement binary integer as specified in Section 10.4 of the X.691 standard.

Parameters:

ctxt_p Pointer to context block structure.
value Signed integer value to be encoded.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_2sCompBinInt64 (ASN1CTXT * *ctxt_p*, ASN1INT64 *value*)

This function encodes a two's complement binary 64-bit integer as specified in Section 10.4 of the X.691 standard.

Parameters:

ctxt_p Pointer to context block structure.
value Signed 64-bit integer value to be encoded.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_32BitConstrainedString (ASN1CTXT * *ctxt_p*, Asn132BitCharString *value*, Asn132BitCharSet * *pCharSet*)

This function will encode a constrained ASN.1 character string. This function is normally not called directly but rather is called from Useful Type Character String encode functions that deal with 32-bit strings. The only function that does that in this release is the `pe_UniversalString` function.

Parameters:

ctxt_p 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 Character string to be encoded. The structure includes a count field containing the number of characters to encode and an array of unsigned short integers to hold the 32-bit characters to be encoded.
pCharSet Pointer to the constraining character set. The contains an array containing all valid characters in the set as well as the aligned and unaligned bit counts required to encode the characters.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_aligned_octets (ASN1CTXT * *ctxt_p*, ASN1OCTET * *pvalue*, ASN1UINT *nocts*)

Parameters:

ctxt_p Pointer to context block structure.

pvalue A pointer to a character string containing the value to be encoded.
nocts The number of octets.

int pe_BigInteger (ASN1CTXT * *ctxt_p*, ASN1ConstCharPtr *pvalue*)

The `pe_BigInteger` function will encode 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 radices currently are not supported. It is highly recommended to use the hexadecimal or binary strings for better performance.

Parameters:

ctxt_p Pointer to context block structure.
pvalue A pointer to a character string containing the value to be encoded.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_bit (ASN1CTXT * *ctxt_p*, ASN1BOOL *value*)

This function will encode a variable of the ASN.1 BOOLEAN type in single bit,

Parameters:

ctxt_p 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 The BOOLEAN value to be encoded.

int pe_bits (ASN1CTXT * *ctxt_p*, ASN1UINT *value*, ASN1UINT *nbits*)

This function encodes multiple bits.

Parameters:

ctxt_p Pointer to context block structure.
value Unsigned integer containing the bits to be encoded.
nbits Number of bits in value to encode.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_bits64 (ASN1CTXT * *ctxt_p*, ASN1UINT64 *value*, ASN1UINT *nbits*)

This function encodes multiple bits, using unsigned 64-bit integer to hold bits.

Parameters:

ctxt_p Pointer to context block structure.
value Unsigned 64-bit integer containing the bits to be encoded.
nbits Number of bits in value to encode.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_BitString (ASN1CTXT * *ctxt_p*, ASN1UINT *numocts*, ASN1ConstOctetPtr *data*)

This function will encode a value of the ASN.1 bit string type.

Parameters:

ctxt_p 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.

numocts The number of bits in the string to be encoded.

data Pointer to the bit string data to be encoded.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_BMPString (ASN1CTXT * *ctxt_p*, ASN1BMPString *value*, Asn116BitCharSet * *permCharSet*)

This function will encode a variable of the ASN.1 BMP character string. This differs from the encode routines for the character strings previously described in that the BMP string type is based on 16-bit characters. A 16-bit character string is modeled using an array of unsigned short integers.

Parameters:

ctxt_p 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 Character string to be encoded. This structure includes a count field containing the number of characters to encode and an array of unsigned short integers to hold the 16-bit characters to be encoded.

permCharSet Pointer to the constraining character set. This contains an array containing all valid characters in the set as well as the aligned and unaligned bit counts required to encode the characters.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_byte_align (ASN1CTXT * *ctxt_p*)

This function will position the encode bit cursor on the next byte boundary.

Parameters:

ctxt_p 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 pe_CheckBuffer (ASN1CTXT * *ctxt_p*, size_t *nbytes*)

This function will determine if the given number of bytes will fit in the encode buffer. If not, either the buffer is expanded (if it is a dynamic buffer) or an error is signaled.

Parameters:

ctxt_p 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.

nbytes Number of bytes of space required to hold the variable to be encoded.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_ChoiceTypeExt (ASN1CTXT * *ctxt_p*, ASN1UINT *numocts*, ASN1ConstOctetPtr *data*)

Parameters:

ctxt_p 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.

numocts Number of octets in the string to be encoded.

data Pointer to octet string data to be encoded.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_ConstInt64 (ASN1CTXT * *ctxt_p*, ASN1INT64 *value*, ASN1INT64 *lower*, ASN1INT64 *upper*)

This function encodes a 64-bit integer constrained either by a value or value range constraint.

Parameters:

ctxt_p Pointer to context block structure.

value Value to be encoded, represented as 64-bit integer.

lower Lower range value, represented as 64-bit integer.

upper Upper range value, represented as 64-bit integer.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_ConstInteger (ASN1CTXT * *ctxt_p*, ASN1INT *value*, ASN1INT *lower*, ASN1INT *upper*)

This function encodes an integer constrained either by a value or value range constraint.

Parameters:

ctxt_p Pointer to context block structure.

value Value to be encoded.

lower Lower range value.

upper Upper range value.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_ConstrainedString (ASN1CTXT * *ctxt_p*, ASN1ConstCharPtr *string*, Asn1CharSet * *pCharSet*)

This function encodes a constrained string value. This is a deprecated version of the function provided for backward compatibility.

Parameters:

ctxt_p Pointer to context block structure.
string Pointer to string to be encoded.
pCharSet Pointer to a character set descriptor structure.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_ConstrainedStringEx (ASN1CTXT * *ctxt_p*, ASN1ConstCharPtr *string*, ASN1ConstCharPtr *charSet*, ASN1UINT *abits*, ASN1UINT *ubits*, ASN1UINT *canSetBits*)

This function encodes a constrained string value. This version of the function allows all of the required permitted alphabet constraint parameters to be passed in as arguments.

Parameters:

ctxt_p Pointer to context block structure.
string Pointer to string to be encoded.
charSet String containing permitted alphabet character set. Can be null if no character set was specified.
abits Number of bits in a character set character (aligned).
ubits Number of bits in a character set character (unaligned).
canSetBits Number of bits in a character from the canonical set representing this string.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_ConsUInt64 (ASN1CTXT * *ctxt_p*, ASN1UINT64 *value*, ASN1UINT64 *lower*, ASN1UINT64 *upper*)

This function encodes an unsigned 64-bit integer constrained either by a value or value range constraint. The constrained unsigned integer option is used if:

1. The lower value of the range is ≥ 0 , and 2. The upper value of the range is $\geq \text{MAXINT}$

Parameters:

ctxt_p Pointer to context block structure.
value Value to be encoded, represented as unsigned 64-bit integer.
lower Lower range value, represented as unsigned 64-bit integer.
upper Upper range value, represented as unsigned 64-bit integer.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_ConsUnsigned (ASN1CTXT * *ctxt_p*, ASN1UINT *value*, ASN1UINT *lower*, ASN1UINT *upper*)

This function encodes an unsigned integer constrained either by a value or value range constraint. The constrained unsigned integer option is used if:

1. The lower value of the range is ≥ 0 , and 2. The upper value of the range is $\geq \text{MAXINT}$

Parameters:

ctxt_p Pointer to context block structure.
value Value to be encoded.

lower Lower range value.

upper Upper range value.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_ConsWholeNumber (ASN1CTXT * *ctxt_p*, ASN1UINT *adjusted_value*, ASN1UINT *range_value*)

This function encodes a constrained whole number as specified in Section 10.5 of the X.691 standard.

Parameters:

ctxt_p Pointer to context block structure.

adjusted_value Unsigned adjusted integer value to be encoded. The adjustment is done by subtracting the lower value of the range from the value to be encoded.

range_value Unsigned integer value specifying the total size of the range. This is obtained by subtracting the lower range value from the upper range value.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_ConsWholeNumber64 (ASN1CTXT * *ctxt_p*, ASN1UINT64 *adjusted_value*, ASN1UINT64 *range_value*)

This function encodes a constrained whole number as specified in Section 10.5 of the X.691 standard, represented as 64-bit integer.

Parameters:

ctxt_p Pointer to context block structure.

adjusted_value Unsigned adjusted integer value to be encoded. The adjustment is done by subtracting the lower value of the range from the value to be encoded.

range_value Unsigned integer value specifying the total size of the range. This is obtained by subtracting the lower range value from the upper range value.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_ExpandBuffer (ASN1CTXT * *ctxt_p*, size_t *nbytes*)

This function will expand the buffer to hold the given number of bytes.

Parameters:

ctxt_p 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.

nbytes The number of bytes the buffer is to be expanded by. Note that the buffer will be expanded by ASN_K_ENCBIFXIZ or *nbytes* (whichever is larger).

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

ASN1UINT pe_GetIntLen (ASN1UINT *value*)

Parameters:

value Length value to be encoded.

size_t pe_GetMsgBitCnt (ASN1CTXT * *ctxt_p*)

Parameters:

ctxt_p 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.

ASN1OCTET* pe_GetMsgPtr (ASN1CTXT * *ctxt_p*, int * *pLength*)

This function will return the message pointer and length of an encoded message. This function is called after a compiler generated encode function to get the pointer and length of the message. It is normally used when dynamic encoding is specified because the message pointer is not known until encoding is complete. If static encoding is used, the message starts at the beginning of the specified buffer and the `pe_GetMsgLen` function can be used to obtain the length of the message.

Parameters:

ctxt_p 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.
pLength Pointer to variable to receive length of the encoded message.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

ASN1OCTET* pe_GetMsgPtrU (ASN1CTXT * *ctxt_p*, ASN1UINT * *pLength*)

Parameters:

ctxt_p 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.
pLength Pointer to variable to receive length of the encoded message.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_Length (ASN1CTXT * *ctxt_p*, ASN1UINT *value*)

This function will encode a length determinant value.

Parameters:

ctxt_p 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 Length value to be encoded.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,

- negative return value is error.

int pe_NonNegBinInt (ASN1CTXT * *ctxt_p*, ASN1UINT *value*)

This function encodes a non-negative binary integer as specified in Section 10.3 of the X.691 standard.

Parameters:

ctxt_p Pointer to context block structure.
value Unsigned integer value to be encoded.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_NonNegBinInt64 (ASN1CTXT * *ctxt_p*, ASN1UINT64 *value*)

This function encodes a non-negative binary 64-bit integer as specified in Section 10.3 of the X.691 standard.

Parameters:

ctxt_p Pointer to context block structure.
value Unsigned 64-bit integer value to be encoded.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_ObjectIdentifier (ASN1CTXT * *ctxt_p*, ASN1OBJID * *pvalue*)

This function encodes a value of the ASN.1 object identifier type.

Parameters:

ctxt_p Pointer to context block structure.
pvalue Pointer to value to be encoded. The ASN1OBJID structure contains a numids fields 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 pe_octets (ASN1CTXT * *ctxt_p*, ASN1ConstOctetPtr *pvalue*, ASN1UINT *nbits*)

This function will encode an array of octets. The Octets will be encoded unaligned starting at the current bit offset within the encode buffer.

Parameters:

ctxt_p A pointer to a context structure. The provides a storage area for the function to store all working variables that must be maintained between function calls.
pvalue A pointer to an array of octets to encode
nbits The number of Octets to encode

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_OctetString (ASN1CTXT * *ctxt_p*, ASN1UINT *numocts*, ASN1ConstOctetPtr *data*)

This function will encode a value of the ASN.1 octet string type.

Parameters:

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

numocts Number of octets in the string to be encoded.

data Pointer to octet string data to be encoded.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_oid64 (ASN1CTXT * *ctxt_p*, ASN1OID64 * *pvalue*)

This function encodes a value of the ASN.1 object identifier type, using 64-bit subidentifiers.

Parameters:

ctxt_p Pointer to context block structure.

pvalue Pointer to value to be encoded. The ASN1OID64 structure contains a numids fields to hold the number of subidentifiers and an array of unsigned 64-bit integers to hold the subidentifier values.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_OpenType (ASN1CTXT * *ctxt_p*, ASN1UINT *numocts*, ASN1ConstOctetPtr *data*)

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

Parameters:

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

numocts Number of octets in the string to be encoded.

data Pointer to octet string data to be encoded.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_OpenTypeExt (ASN1CTXT * *ctxt_p*, Asn1RTDList * *pElemList*)

This function will encode an ASN.1 open type extension. An open type extension field is the data that potentially resides after the ... marker in a version-1 message. The open type structure contains a complete encoded bit set including option element bits or choice index, length, and data. Typically, this data is populated when a version-1 system decodes a version-2 message. The extension fields are retained and can then be re-encoded if a new message is to be sent out (for example, in a store and forward system).

Parameters:

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

pElemList A pointer to the open type to be encoded.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_OpenTypeExtBits (ASN1CTXT * *ctxt_p*, Asn1RTDList * *pElemList*)

Parameters:

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

pElemList A pointer to the open type to be encoded.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_Real (ASN1CTXT * *ctxt_p*, ASN1REAL *value*)

This function will encode a value of the ASN.1 real type. This function provides support for the plus-infinity and minus-infinity special real values. Use the `rtGetPlusInfinity` or `rtGetMinusInfinity` functions to get these special values.

Parameters:

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

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

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_RelativeOID (ASN1CTXT * *ctxt_p*, ASN1OBJID * *pvalue*)

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

Parameters:

ctxt_p Pointer to context block structure.

pvalue Pointer to value to be encoded. The ASN1OBJID structure contains a `numids` fields 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 pe_SemiConsInt64 (ASN1CTXT * *ctxt_p*, ASN1INT64 *value*, ASN1INT64 *lower*)

This function encodes an semi-constrained 64-bit integer.

Parameters:

ctxt_p Pointer to context block structure.

value Value to be encoded, represented as 64-bit integer.

lower Lower range value, represented as signed 64-bit integer.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_SemiConsInteger (ASN1CTXT * *ctxt_p*, ASN1INT *value*, ASN1INT *lower*)

This function encodes an semi-constrained integer.

Parameters:

ctxt_p Pointer to context block structure.

value Value to be encoded.

lower Lower range value, represented as signed integer.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_SemiConsUInt64 (ASN1CTXT * *ctxt_p*, ASN1UINT64 *value*, ASN1UINT64 *lower*)

This function encodes an semi-constrained unsigned 64-bit integer.

Parameters:

ctxt_p Pointer to context block structure.

value Value to be encoded, represented as unsigned 64-bit integer.

lower Lower range value, represented as unsigned 64-bit integer.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_SemiConsUnsigned (ASN1CTXT * *ctxt_p*, ASN1UINT *value*, ASN1UINT *lower*)

This function encodes an semi-constrained unsigned integer.

Parameters:

ctxt_p Pointer to context block structure.

value Value to be encoded.

lower Lower range value, represented as unsigned integer.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_SmallNonNegWholeNumber (ASN1CTXT * *ctxt_p*, ASN1UINT *value*)

This function will encode a small, non-negative whole number as specified in Section 10.6 of the X.691 standard. This is a number that is expected to be small, but whose size is potentially unlimited due to the presence of an extension marker.

Parameters:

ctxt_p A pointer to a context structure. The 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.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_UnconsLength (ASN1CTXT * *ctxt_p*, ASN1UINT *value*)

Parameters:

ctxt_p 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 Value to be encoded.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_UniversalString (ASN1CTXT * *ctxt_p*, ASN1UniversalString *value*, Asn132BitCharSet * *permCharSet*)

This function will encode 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:

ctxt_p 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 Character string to be encoded. The structure includes a count field containing all valid characters in the set as well as the aligned and unaligned bit counts required to encode the characters.

permCharSet A pointer to the constraining character set. This contains an array containing all valid characters in the set as well as the aligned and the unaligned bit counts required to encode the characters.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

int pe_VarWidthCharString (ASN1CTXT * *pctxt*, ASN1ConstCharPtr *value*)

Parameters:

pctxt Pointer to context block structure.

value Value to be encoded, represented as unsigned 64-bit integer.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

PER C Utility Functions

Detailed Description

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

Defines

- #define **pd_setp**(ctxt_p, bufaddr, bufsiz, aligned) pu_setBuffer(ctxt_p, bufaddr, bufsiz, aligned)
- #define **pe_resetp**(ctxt_p) rtResetContext(ctxt_p)
- #define **pd_resetp**(ctxt_p) rtResetContext(ctxt_p)

Functions

- int **pu_addSizeConstraint** (ASN1CTXT *ctxt_p, Asn1SizeCnst *pSize)
- ASN1BOOL **pu_alignCharStr** (ASN1CTXT *ctxt_p, ASN1UINT len, ASN1UINT nbits, Asn1SizeCnst *pSize)
- int **pu_checkSizeConstraint** (ASN1CTXT *ctxt_p, int size)
- ASN1UINT **pu_bitcnt** (ASN1UINT value)
- Asn1SizeCnst * **pu_checkSize** (Asn1SizeCnst *pSizeList, ASN1UINT value, ASN1BOOL *pExtendable)
- void **pu_freeContext** (ASN1CTXT *ctxt_p)
- int **pu_getBitOffset** (ASN1CTXT *ctxt_p)
- size_t **pu_getMaskAndIndex** (size_t bitOffset, unsigned char *pMask)
- size_t **pu_getMsgLen** (ASN1CTXT *ctxt_p)
- void **pu_hexdump** (ASN1CTXT *ctxt_p)
- int **pu_setBuffer** (ASN1CTXT *ctxt_p, ASN1OCTET *bufaddr, size_t bufsiz, ASN1BOOL aligned)
- int **pe_setp** (ASN1CTXT *ctxt_p, ASN1OCTET *bufaddr, size_t bufsiz, ASN1BOOL aligned)
- int **pu_initContext** (ASN1CTXT *ctxt_p, ASN1OCTET *bufaddr, ASN1UINT bufsiz, ASN1BOOL aligned)
- int **pu_initContextBuffer** (ASN1CTXT *pTarget, ASN1CTXT *pSource)
- ASN1ConstCharPtr **pu_getFullName** (ASN1CTXT *ctxt_p, ASN1ConstCharPtr suffix)
- Asn1SizeCnst * **pu_getSizeConstraint** (ASN1CTXT *ctxt_p, ASN1BOOL extbit)
- void **pu_init16BitCharSet** (Asn116BitCharSet *pCharSet, ASN116BITCHAR first, ASN116BITCHAR last, ASN1UINT abits, ASN1UINT ubits)
- void **pu_insLenField** (ASN1CTXT *ctxt_p)
- ASN1BOOL **pu_isExtendableSize** (Asn1SizeCnst *pSizeList)
- ASN1BOOL **pu_isFixedSize** (Asn1SizeCnst *pSizeList)
- ASN1CTXT * **pu_newContext** (ASN1OCTET *bufaddr, ASN1UINT bufsiz, ASN1BOOL aligned)
- PERField * **pu_newField** (ASN1CTXT *ctxt_p, ASN1ConstCharPtr nameSuffix)
- void **pu_popName** (ASN1CTXT *ctxt_p)
- void **pu_pushElemName** (ASN1CTXT *ctxt_p, int index)
- void **pu_pushName** (ASN1CTXT *ctxt_p, ASN1ConstCharPtr name)
- void **pu_setBitOffset** (ASN1CTXT *ctxt_p, int bitOffset)
- void **pu_setCharSet** (Asn1CharSet *pCharSet, ASN1ConstCharPtr permSet)
- void **pu_set16BitCharSet** (ASN1CTXT *ctxt_p, Asn116BitCharSet *pCharSet, Asn116BitCharSet *pAlphabet)
- void **pu_set16BitCharSetFromRange** (Asn116BitCharSet *pCharSet, ASN1USINT firstChar, ASN1USINT lastChar)
- void **pu_setFldBitCount** (ASN1CTXT *ctxt_p)
- void **pu_setFldBitOffset** (ASN1CTXT *ctxt_p)
- ASN1BOOL **pu_setTrace** (ASN1CTXT *pCtxt, ASN1BOOL value)
- void **pu_bindump** (ASN1CTXT *ctxt_p, ASN1ConstCharPtr varname)
- void **pu_dumpField** (ASN1CTXT *ctxt_p, PERField *pField, ASN1ConstCharPtr varname, size_t nextBitOffset, BinDumpBuffer *pbuf)

- void **pu_init32BitCharSet** (Asn132BitCharSet *pCharSet, ASN132BITCHAR first, ASN132BITCHAR last, ASN1UINT abits, ASN1UINT ubits)
 - void **pu_set32BitCharSet** (ASN1CTXT *ctxt_p, Asn132BitCharSet *pCharSet, Asn132BitCharSet *pAlphabet)
 - void **pu_set32BitCharSetFromRange** (Asn132BitCharSet *pCharSet, ASN1UINT firstChar, ASN1UINT lastChar)
 - int **pu_GetLibVersion** ()
 - const char * **pu_GetLibInfo** ()
-

Function Documentation

int **pu_addSizeConstraint** (ASN1CTXT * *ctxt_p*, Asn1SizeCnst * *pSize*)

This function is used to add size to a context variable.

Parameters:

ctxt_p A pointer to a context structure. The referenced size constraint is added to this structure for use by a subsequent encode or decode function.

pSize A pointer to the size constraint to add the context variable.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

void **pu_bindump** (ASN1CTXT * *ctxt_p*, ASN1ConstCharPtr *varname*)

This function provides a detailed binary dump of the contents of the buffer currently specified in the given context. The list of fields dumped by this function was previously built up within the context using calls `pu_newField`, `pu_pushName`, and `pu_popName`. These calls are built into both compiler-generated and low-level PER encode/decode functions to trace the actual bit encoding of a given construct.

Parameters:

ctxt_p A pointer to a context structure. The contents of the encode or decode buffer are dumped.

varname The name of the top-level variable name of the structure being dumped.

ASN1UINT **pu_bitcnt** (ASN1UINT *value*)

Parameters:

value Value to be encoded.

int **pu_checkSizeConstraint** (ASN1CTXT * *ctxt_p*, int *size*)

Parameters:

ctxt_p A pointer to a context structure. The referenced size constraint is added to this structure for use by a subsequent encode or decode function.

size The size constraint to add the context variable.

void **pu_freeContext** (ASN1CTXT * *ctxt_p*)

This function releases all dynamic memory associated with a context. This function should be called even if the referenced context variable is not dynamic. The reason is because it frees

memory allocated within the context as well as the context structure (it will only try to free the context structure if it detects that it was previously allocated using the `pu_newContext` function).

Parameters:

ctxt_p 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 pu_getBitOffset (ASN1CTXT * *ctxt_p*)

Parameters:

ctxt_p 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.

const char* pu_GetLibInfo ()

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 pu_GetLibVersion ()

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 pu_getMsgLen (ASN1CTXT * *ctxt_p*)

This function will return the number of bits in a encoded message. This function is called after a complier generated encode function is called to get the bit count of the encoded component.

Parameters:

ctxt_p 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:

Length (in bits) of encoded message content.

void pu_hexdump (ASN1CTXT * *ctxt_p*)

This function provides a standard hexadecimal dump of the contents of the buffer currently specified in the given context.

Parameters:

ctxt_p Pointer to a context structure. The contents of the encode or decode buffer defined in the context are dumped.

int pu_initContextBuffer (ASN1CTXT * *pTarget*, ASN1CTXT * *pSource*)

This function is used to initialize the buffer of an ASN1CTXT structure with buffer data from a second context structure. This function copies the buffer information from the source

context buffer to the destination structure. The non-buffer related fields in the context remain untouched.

Parameters:

pTarget A pointer to the target context structure. Buffer information within this structure is updated with data from the source context.

pSource A pointer to the source context structure. Buffer information from the source context structure is copied to the target structure.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

ASN1BOOL pu_isExtendableSize (Asn1SizeCnst * pSizeList)

Parameters:

pSizeList A pointer to the size of the Linked List.

ASN1BOOL pu_isFixedSize (Asn1SizeCnst * pSizeList)

Parameters:

pSizeList A pointer to the size of the Linked List.

void pu_popName (ASN1CTXT * ctxt_p)

Parameters:

ctxt_p A pointer to a context structure.

void pu_pushElemName (ASN1CTXT * ctxt_p, int index)

Pushes an element on the stack.

Parameters:

ctxt_p A pointer to a context structure.

index The location to insert the element.

void pu_pushName (ASN1CTXT * ctxt_p, ASN1ConstCharPtr name)

Parameters:

ctxt_p A pointer to a context structure.

name A pointer to the element to add to the stack.

void pu_set16BitCharSet (ASN1CTXT * ctxt_p, Asn116BitCharSet * pCharSet, Asn116BitCharSet * pAlphabet)

This function sets a permitted alphabet character set for 16-bit character strings. This is the resulting set of character when the character associated with a 16-bit string type is merged with a permitted alphabet constraint.

Parameters:

ctxt_p Pointer to a context structure.

pCharSet Pointer to a character set structure describing the character set currently associated with the character string type. The resulting character set structure after being merged with the permSet parameter.

pAlphabet Pointer to a structure describing the 16-bit permitted alphabet.

void pu_set16BitCharSetFromRange (Asn116BitCharSet * pCharSet, ASN1USINT firstChar, ASN1USINT lastChar)

Parameters:

pCharSet Pointer to a character set structure describing the character set currently associated with the character string type. The resulting character set structure after being merged with the permSet parameter.

firstChar The first character in the range.

lastChar The last character in the range.

void pu_set32BitCharSet (ASN1CTXT * ctxt_p, Asn132BitCharSet * pCharSet, Asn132BitCharSet * pAlphabet)

This function sets a permitted alphabet character set for 32-bit character strings. This is the resulting set of character when the character associated with a 16-bit string type is merged with a permitted alphabet constraint.

Parameters:

ctxt_p Pointer to a context structure.

pCharSet Pointer to a character set structure describing the character set currently associated with the character string type. The resulting character set structure after being merged with the permSet parameter.

pAlphabet Pointer to a structure describing the 32-bit permitted alphabet.

void pu_set32BitCharSetFromRange (Asn132BitCharSet * pCharSet, ASN1UINT firstChar, ASN1UINT lastChar)

Parameters:

pCharSet Pointer to a character set structure describing the character set currently associated with the character string type. The resulting character set structure after being merged with the permSet parameter.

firstChar The first character in the range.

lastChar The last character in the range.

void pu_setCharSet (Asn1CharSet * pCharSet, ASN1ConstCharPtr permSet)

This function sets a permitted alphabet character set. This is the resulting set of characters when the character associated with a standard character string type is merged with a permitted alphabet constraint.

Parameters:

pCharSet A pointer to a character set structure describing the character set currently associated with the character string type. The resulting character set structure after being merged with the permSet parameter.

permSet A null-terminated string of permitted characters.

void pu_setFldBitCount (ASN1CTXT * ctxt_p)

Parameters:

ctxt_p A pointer to a context structure.

void pu_setFldBitOffset (ASN1CTXT * *ctxt_p*)

Parameters:

ctxt_p A pointer to a context structure.

ASN1BOOL pu_setTrace (ASN1CTXT * *pCtxt*, ASN1BOOL *value*)

Parameters:

pCtxt A pointer to a context structure.

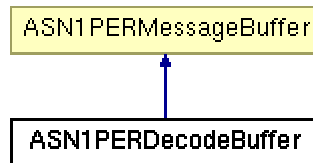
value The BOOLEAN value to be encoded.

ASN1C PER Runtime Class Documentation

ASN1PERDecodeBuffer Class Reference

```
#include <asn1PerCppTypes.h>
```

Inheritance diagram for ASN1PERDecodeBuffer:



Detailed Description

The ASN1PERDecodeBuffer class is derived from the ASN1PERMessageBuffer base class. It contains variables and methods specific to decoding ASN.1 PER messages. It is used to manage the input buffer containing the ASN.1 message to be decoded. This class has 3 overloaded constructors.

Public Member Functions

- ASN1PERDecodeBuffer (ASN1BOOL aligned)
- ASN1PERDecodeBuffer (const ASN1OCTET *pMsgBuf, size_t msgBufLen, ASN1BOOL aligned)
- ASN1PERDecodeBuffer (const char *filePath, ASN1BOOL aligned)
- virtual ASN1BOOL isA (Type bufferType)
- int readBinaryFile (const char *filePath)

Constructor & Destructor Documentation

ASN1PERDecodeBuffer::ASN1PERDecodeBuffer (ASN1BOOL *aligned*) [inline]

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

Parameters:

aligned Flag indicating if the message was encoded using aligned (TRUE)* or unaligned (FALSE) encoding.

ASN1PERDecodeBuffer::ASN1PERDecodeBuffer (const ASN1OCTET * *pMsgBuf*, size_t *msgBufLen*, ASN1BOOL *aligned*) [inline]

This constructor is used to describe the message to be decoded. Use getStatus() method to determine has error occurred during the initialization or not.

Parameters:

pMsgBuf A pointer to the message to be decoded.
msgBufLen Length of the message buffer.

aligned Flag indicating if the message was encoded using aligned (TRUE) * or unaligned (FALSE) encoding.

ASN1PERDecodeBuffer::ASN1PERDecodeBuffer (const char * *filePath*, ASN1BOOL *aligned*)

This constructor takes a pointer to the path of a file containing a binary PER message to be decoded.

Parameters:

filePath Complete file path and name of file to read.

aligned Flag indicating if the message was encoded using aligned (TRUE) * or unaligned (FALSE) encoding.

Member Function Documentation

virtual ASN1BOOL ASN1PERDecodeBuffer::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 PERDecode.

Returns:

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

int ASN1PERDecodeBuffer::readBinaryFile (const char * *filePath*)

This method reads the file into the buffer to decode.

Parameters:

filePath The zero-terminated string containing the path to the file.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

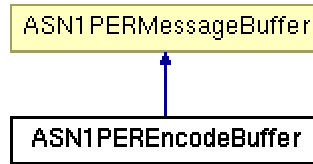
The documentation for this class was generated from the following file:

- **asn1PerCppType.h**

ASN1PEREncodeBuffer Class Reference

```
#include <asn1PerCppTypes.h>
```

Inheritance diagram for ASN1PEREncodeBuffer:



Detailed Description

The ASN1PEREncodeBuffer class is derived from the **ASN1PERMessageBuffer** base class. It contains variables and methods specific to encoding ASN.1 messages. It is used to manage the buffer into which an ASN.1 PER message is to be encoded.

Public Member Functions

- **ASN1PEREncodeBuffer** (ASN1BOOL aligned)
- **ASN1PEREncodeBuffer** (ASN1OCTET *pMsgBuf, size_t msgBufLen, ASN1BOOL aligned)
- size_t **getMsgBitCnt** ()
- virtual ASN1OCTET * **getMsgCopy** ()
- virtual const ASN1OCTET * **getMsgPtr** ()
- int **init** ()
- virtual ASN1BOOL **isA** (Type bufferType)
- int **GetMsgBitCnt** ()

Constructor & Destructor Documentation

ASN1PEREncodeBuffer::ASN1PEREncodeBuffer (ASN1BOOL *aligned*) [inline]

The ASN1PEREncodeBuffer class has two overloaded constructors: This version that takes one argument, aligned flag (dynamic encoding version). Use getStatus() method to determine has error occurred during the initialization or not.

Parameters:

aligned Flag indicating if aligned (TRUE) or unaligned (FALSE) encoding should be done.

ASN1PEREncodeBuffer::ASN1PEREncodeBuffer (ASN1OCTET * *pMsgBuf*, size_t *msgBufLen*, ASN1BOOL *aligned*) [inline]

The ASN1PEREncodeBuffer class has two overloaded constructors: This version that takes a message buffer and size argument and an aligned flag argument (static encoding version). 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.

aligned Flag indicating if aligned (TRUE) or unaligned (FALSE) encoding should be done.

Member Function Documentation

size_t ASN1PEREncodeBuffer::getMsgBitCnt () [inline]

This method returns the length (in bits) of the encoded message.

Returns:

Length(in bits)of encoded message

virtual ASN1OCTET* ASN1PEREncodeBuffer::getMsgCopy () [virtual]

This method returns a copy of the current encoded message. Memory is allocated for the message using the ‘new’ operation. It is the user’s responsibility to free the memory using ‘delete’.

Returns:

Pointer to copy of encoded message. It is the user’s responsibility to release the memory using the ‘delete’ operator (i.e., delete [] ptr;)

virtual const ASN1OCTET* ASN1PEREncodeBuffer::getMsgPtr () [virtual]

This method returns the internal pointer to the current encoded message.

Returns:

Pointer to encoded message.

int ASN1PEREncodeBuffer::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 ASN1PEREncodeBuffer::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 PEREncode.

Returns:

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

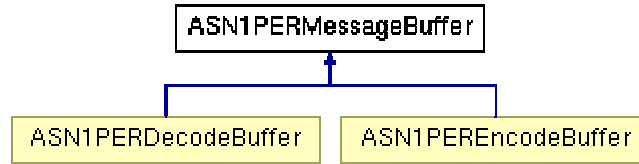
The documentation for this class was generated from the following file:

- **asn1PerCppType.h**

ASN1PERMessageBuffer Class Reference

```
#include <asn1PerCppTypes.h>
```

Inheritance diagram for ASN1PERMessageBuffer:



Detailed Description

The ASN1PERMessageBuffer class is derived from the ASN1MessageBuffer base class. It is the base class for the **ASN1PEREncodeBuffer** and **ASN1PERDecodeBuffer** derived classes. It contains variables and methods specific to encoding or decoding ASN.1 messages using the Packed Encoding Rules (PER). It is used to manage the buffer into which an ASN.1 message is to be encoded or decoded.

Public Member Functions

- void **binDump** (const char *varname)
- void **hexDump** ()
- size_t **getMsgLen** ()
- void **setTrace** (ASN1BOOL value)
- int **setBuffer** (const ASN1OCTET *pMsgBuf, size_t msgBufLen)
- void **BinDump** (const char *varname)
- void **HexDump** ()
- int **GetMsgLen** ()
- void **SetTrace** (ASN1BOOL value)

Protected Member Functions

- **ASN1PERMessageBuffer** (Type bufferType, ASN1BOOL aligned)
- **ASN1PERMessageBuffer** (Type bufferType, ASN1OCTET *pMsgBuf, size_t msgBufLen, ASN1BOOL aligned)

Constructor & Destructor Documentation

ASN1PERMessageBuffer::ASN1PERMessageBuffer (Type *bufferType*, ASN1BOOL *aligned*)
[protected]

This constructor does not set a PER input source. It is used by the derived encode buffer classes. Use the getStatus() method to determine if an error has occurred during initialization.

Parameters:

bufferType Type of message buffer that is being created (for example, PEREncode or PERDecode).

aligned Flag indicating if aligned (TRUE) or unaligned (FALSE) encoding should be done.

ASN1PERMessageBuffer::ASN1PERMessageBuffer (Type *bufferType*, ASN1OCTET * *pMsgBuf*, size_t *msgBufLen*, ASN1BOOL *aligned*) [protected]

This constructor allows a memory buffer holding a binary PER message to be specified. Use the `getStatus()` method to determine if an error has occurred during initialization.

Parameters:

bufferType Type of message buffer that is being created (for example, PEREncode or PERDecode).

pMsgBuf A pointer to a fixed size message buffer to receive the encoded message.

msgBufLen Size of the fixed-size message buffer.

aligned Flag indicating if aligned (TRUE) or unaligned (FALSE) encoding should be done.

Member Function Documentation

void ASN1PERMessageBuffer::binDump (const char * *varname*) [inline]

This method outputs a binary dump of the current buffer contents to stdout.

Parameters:

varname char pointer to current buffer

size_t ASN1PERMessageBuffer::getMsgLen () [inline]

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

Parameters:

- none

void ASN1PERMessageBuffer::hexDump () [inline]

This method outputs a hexadecimal dump of the current buffer contents to stdout.

Parameters:

- none

int ASN1PERMessageBuffer::setBuffer (const ASN1OCTET * *pMsgBuf*, size_t *msgBufLen*)

This method sets a buffer to receive the encoded message.

Parameters:

pMsgBuf A pointer to a memory buffer to use to encode a message. The buffer should be declared as an array of unsigned characters (ASN1OCTETs). This parameter can be set to NULL to specify dynamic encoding (i.e., the encode functions will dynamically allocate a buffer for the message).

msgBufLen The length of the memory buffer in bytes. If *pMsgBuf* is NULL, this parameter specifies the initial size of the dynamic buffer; if 0 - the default size will be used.

Returns:

Completion status of operation:

- 0 (ASN_OK) = success,
- negative return value is error.

void ASN1PERMessageBuffer::setTrace (ASN1BOOL *value*) [inline]

This method turns PER diagnostic tracing on or off.

This enables the collection of the bit statistics inside the PER library functions that can be displayed using the `binDump` method.

Parameters:

value Boolean value indicating whether tracing should be on (true) or off (false).

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

- **asn1PerCppType.h**

ASN1C PER Runtime File Documentation

asn1per.h File Reference

Detailed Description

ASN.1 runtime constants, data structure definitions, and functions to support the Packet Encoding Rules (PER) as defined in the ITU-T X.691 standard.

```
#include "asn1type.h"
#include "asn1CharSet.h"
```

Classes

- struct **PERField**
- struct **BinDumpBuffer**

Defines

- #define **ASN_K_EXTENUM** 999
- #define **PU_SETCHARSET**(csetvar, canset, abits, ubits)
- #define **PU_INSLENFLD**(ctxt_p)
- #define **PU_NEWFIELD**(ctxt_p, suffix)
- #define **PU_PUSHNAME**(ctxt_p, name)
- #define **PU_POPNAME**(ctxt_p)
- #define **PU_SETBITOFFSET**(ctxt_p)
- #define **PU_SETBITCOUNT**(ctxt_p)
- #define **PU_PUSHELEMNAME**(ctxt_p, idx)
- #define **EXTERNPER**
- #define **PD_INCRBITIDX**(ctxt_p)
- #define **PD_BIT**(ctxt_p, pvalue)
- #define **PU_GETSIZECONSTRAINT**(ctxt_p, extbit, pSize)
- #define **PU_GETCTXTBITOFFSET**(ctxt_p) (((ctxt_p)->buffer.byteIndex * 8) + (8 - (ctxt_p)->buffer.bitOffset))
- #define **PU_SETCTXTBITOFFSET**(ctxt_p, _bitOffset)
- #define **PD_BYTE_ALIGN0**(ctxt_p)
- #define **PD_BYTE_ALIGN** PD_BYTE_ALIGN0
- #define **PD_CHECKSEQOFLEN**(pctxt, numElements, minElemBits)
- #define **pd_UnconsInteger**(ctxt_p, pvalue) pd_SemiConsInteger(ctxt_p, pvalue, ASN1INT_MIN)
- #define **pd_UnconsUnsigned**(ctxt_p, pvalue) pd_SemiConsUnsigned(ctxt_p, pvalue, 0U)
- #define **pd_UnconsInt64**(ctxt_p, pvalue) pd_SemiConsInt64(ctxt_p, pvalue, ASN1INT64MIN)
- #define **pd_UnconsUInt64**(ctxt_p, pvalue) pd_SemiConsUInt64(ctxt_p, pvalue, 0)
- #define **pe_UnconsInteger**(ctxt_p, value) pe_SemiConsInteger(ctxt_p, value, ASN1INT_MIN)
- #define **pe_UnconsInt64**(ctxt_p, value) pe_SemiConsInt64(ctxt_p, value, ASN1INT64MIN)
- #define **pe_UnconsUnsigned**(ctxt_p, value) pe_SemiConsUnsigned(ctxt_p, value, 0)
- #define **pe_UnconsUInt64**(ctxt_p, value) pe_SemiConsUInt64(ctxt_p, value, 0)
- #define **pd_setp**(ctxt_p, bufaddr, bufsiz, aligned) pu_setBuffer(ctxt_p, bufaddr, bufsiz, aligned)
- #define **pe_resetp**(ctxt_p) rtResetContext(ctxt_p)
- #define **pd_resetp**(ctxt_p) rtResetContext(ctxt_p)
- #define **pe_GeneralString**(ctxt_p, value, permCharSet) pe_VarWidthCharString(ctxt_p, value)
- #define **pe_GraphicString**(ctxt_p, value, permCharSet) pe_VarWidthCharString(ctxt_p, value)
- #define **pe_T61String**(ctxt_p, value, permCharSet) pe_VarWidthCharString(ctxt_p, value)

- #define **pe_TeletexString**(ctxt_p, value, permCharSet) pe_VarWidthCharString(ctxt_p, value)
- #define **pe_VideotexString**(ctxt_p, value, permCharSet) pe_VarWidthCharString(ctxt_p, value)
- #define **pe_ObjectDescriptor**(ctxt_p, value, permCharSet) pe_VarWidthCharString(ctxt_p, value)
- #define **pe_UTF8String**(ctxt_p, value, permCharSet) pe_VarWidthCharString(ctxt_p, value)
- #define **pe_IA5String**(ctxt_p, value, permCharSet) pe_ConstrainedStringEx (ctxt_p, value, permCharSet, 8, 7, 7)
- #define **pe_NumericString**(ctxt_p, value, permCharSet)
- #define **pe_PrintableString**(ctxt_p, value, permCharSet) pe_ConstrainedStringEx (ctxt_p, value, permCharSet, 8, 7, 7)
- #define **pe_VisibleString**(ctxt_p, value, permCharSet)
- #define **pe_ISO646String** pe_IA5String
- #define **pe_GeneralizedTime** pe_IA5String
- #define **pe_UTCTime** pe_GeneralizedTime
- #define **pd_GeneralString**(ctxt_p, pvalue, permCharSet) pd_VarWidthCharString (ctxt_p, pvalue)
- #define **pd_GraphicString**(ctxt_p, pvalue, permCharSet) pd_VarWidthCharString (ctxt_p, pvalue)
- #define **pd_VideotexString**(ctxt_p, pvalue, permCharSet) pd_VarWidthCharString (ctxt_p, pvalue)
- #define **pd_TeletexString**(ctxt_p, pvalue, permCharSet) pd_VarWidthCharString (ctxt_p, pvalue)
- #define **pd_T61String**(ctxt_p, pvalue, permCharSet) pd_VarWidthCharString (ctxt_p, pvalue)
- #define **pd_ObjectDescriptor**(ctxt_p, pvalue, permCharSet) pd_VarWidthCharString (ctxt_p, pvalue)
- #define **pd_UTF8String**(ctxt_p, pvalue, permCharSet) pd_VarWidthCharString (ctxt_p, pvalue)
- #define **pd_IA5String**(ctxt_p, pvalue, permCharSet) pd_ConstrainedStringEx (ctxt_p, pvalue, permCharSet, 8, 7, 7)
- #define **pd_NumericString**(ctxt_p, pvalue, permCharSet)
- #define **pd_PrintableString**(ctxt_p, pvalue, permCharSet) pd_ConstrainedStringEx (ctxt_p, pvalue, permCharSet, 8, 7, 7)
- #define **pd_VisibleString**(ctxt_p, pvalue, permCharSet) pd_ConstrainedStringEx (ctxt_p, pvalue, permCharSet, 8, 7, 7)
- #define **pd_ISO646String** pd_IA5String
- #define **pd_GeneralizedTime** pd_IA5String
- #define **pd_UTCTime** pd_GeneralizedTime
- #define **pe_GetMsgLen** pu_getMsgLen

Typedefs

- typedef PERField **PERField**

Functions

- int **pd_bits** (ASN1CTXT *ctxt_p, ASN1UINT *pvalue, ASN1UINT nbits)
- int **pd_BigInteger** (ASN1CTXT *ctxt_p, ASN1ConstCharPtr *ppvalue)
- int **pd_BitString** (ASN1CTXT *ctxt_p, ASN1UINT *numbits_p, ASN1OCTET *buffer, ASN1UINT bufsiz)
- int **pd_BMPString** (ASN1CTXT *ctxt_p, ASN1BMPString *pvalue, Asn116BitCharSet *permCharSet)
- int **pd_UniversalString** (ASN1CTXT *ctxt_p, ASN1UniversalString *pvalue, Asn132BitCharSet *permCharSet)
- int **pd_ChoiceOpenTypeExt** (ASN1CTXT *ctxt_p, ASN1ConstOctetPtr *object_p2, ASN1UINT *numocts_p)
- int **pd_ConsInteger** (ASN1CTXT *ctxt_p, ASN1INT *pvalue, ASN1INT lower, ASN1INT upper)
- int **pd_ConsInt8** (ASN1CTXT *ctxt_p, ASN1INT8 *pvalue, ASN1INT lower, ASN1INT upper)
- int **pd_ConsInt16** (ASN1CTXT *ctxt_p, ASN1SINT *pvalue, ASN1INT lower, ASN1INT upper)
- int **pd_ConsInt64** (ASN1CTXT *ctxt_p, ASN1INT64 *pvalue, ASN1INT64 lower, ASN1INT64 upper)
- int **pd_ConsUnsigned** (ASN1CTXT *ctxt_p, ASN1UINT *pvalue, ASN1UINT lower, ASN1UINT upper)

- int **pd_ConsUInt8** (ASN1CTXT *ctxt_p, ASN1UINT8 *pvalue, ASN1UINT lower, ASN1UINT upper)
- int **pd_ConsUInt16** (ASN1CTXT *ctxt_p, ASN1USINT *pvalue, ASN1UINT lower, ASN1UINT upper)
- int **pd_ConsUInt64** (ASN1CTXT *ctxt_p, ASN1UINT64 *pvalue, ASN1UINT64 lower, ASN1UINT64 upper)
- int **pd_ConsWholeNumber** (ASN1CTXT *ctxt_p, ASN1UINT *padjusted_value, ASN1UINT range_value)
- int **pd_ConsWholeNumber64** (ASN1CTXT *ctxt_p, ASN1UINT64 *padjusted_value, ASN1UINT64 range_value)
- int **pd_ConstrainedString** (ASN1CTXT *ctxt_p, ASN1ConstCharPtr *string, Asn1CharSet *pCharSet)
- int **pd_ConstrainedStringEx** (ASN1CTXT *ctxt_p, ASN1ConstCharPtr *string, ASN1ConstCharPtr charSet, ASN1UINT abits, ASN1UINT ubits, ASN1UINT canSetBits)
- int **pd_16BitConstrainedString** (ASN1CTXT *ctxt_p, Asn116BitCharString *pString, Asn116BitCharSet *pCharSet)
- int **pd_32BitConstrainedString** (ASN1CTXT *ctxt_p, Asn132BitCharString *pString, Asn132BitCharSet *pCharSet)
- int **pd_DynBitString** (ASN1CTXT *ctxt_p, ASN1DynBitStr *pBitStr)
- int **pd_DynOctetString** (ASN1CTXT *ctxt_p, ASN1DynOctStr *pOctStr)
- int **pd_GetComponentLength** (ASN1CTXT *ctxt_p, ASN1UINT itemBits)
- int **pd_Length** (ASN1CTXT *ctxt_p, ASN1UINT *pvalue)
- int **pd_moveBitCursor** (ASN1CTXT *ctxt_p, int bitOffset)
- int **pd_ObjectIdentifier** (ASN1CTXT *ctxt_p, ASN1OBJID *pvalue)
- int **pd_oid64** (ASN1CTXT *ctxt_p, ASN1OID64 *pvalue)
- int **pd_RelativeOID** (ASN1CTXT *ctxt_p, ASN1OBJID *pvalue)
- int **pd_OctetString** (ASN1CTXT *ctxt_p, ASN1UINT *numocts_p, ASN1OCTET *buffer, ASN1UINT bufsiz)
- int **pd_OpenType** (ASN1CTXT *ctxt_p, ASN1ConstOctetPtr *object_p2, ASN1UINT *numocts_p)
- int **pd_OpenTypeExt** (ASN1CTXT *ctxt_p, ASN1ConstOctetPtr *object_p2, ASN1UINT *numocts_p)
- int **pd_Real** (ASN1CTXT *ctxt_p, ASN1REAL *pvalue)
- int **pd_SmallNonNegWholeNumber** (ASN1CTXT *ctxt_p, ASN1UINT *pvalue)
- int **pd_SemiConsInteger** (ASN1CTXT *ctxt_p, ASN1INT *pvalue, ASN1INT lower)
- int **pd_SemiConsUnsigned** (ASN1CTXT *ctxt_p, ASN1UINT *pvalue, ASN1UINT lower)
- int **pd_SemiConsInt64** (ASN1CTXT *ctxt_p, ASN1INT64 *pvalue, ASN1INT64 lower)
- int **pd_SemiConsUInt64** (ASN1CTXT *ctxt_p, ASN1UINT64 *pvalue, ASN1UINT64 lower)
- int **pd_VarWidthCharString** (ASN1CTXT *pctxt, ASN1ConstCharPtr *pvalue)
- int **pe_16BitConstrainedString** (ASN1CTXT *ctxt_p, Asn116BitCharString value, Asn116BitCharSet *pCharSet)
- int **pe_32BitConstrainedString** (ASN1CTXT *ctxt_p, Asn132BitCharString value, Asn132BitCharSet *pCharSet)
- int **pe_2sCompBinInt** (ASN1CTXT *ctxt_p, ASN1INT value)
- int **pe_2sCompBinInt64** (ASN1CTXT *ctxt_p, ASN1INT64 value)
- int **pe_aligned_octets** (ASN1CTXT *ctxt_p, ASN1OCTET *pvalue, ASN1UINT nocts)
- int **pe_BigInteger** (ASN1CTXT *ctxt_p, ASN1ConstCharPtr pvalue)
- int **pe_bit** (ASN1CTXT *ctxt_p, ASN1BOOL value)
- int **pe_bits** (ASN1CTXT *ctxt_p, ASN1UINT value, ASN1UINT nbits)
- int **pe_bits64** (ASN1CTXT *ctxt_p, ASN1UINT64 value, ASN1UINT nbits)
- int **pe_BitString** (ASN1CTXT *ctxt_p, ASN1UINT numocts, ASN1ConstOctetPtr data)
- int **pe_BMPString** (ASN1CTXT *ctxt_p, ASN1BMPString value, Asn116BitCharSet *permCharSet)
- int **pe_UniversalString** (ASN1CTXT *ctxt_p, ASN1UniversalString value, Asn132BitCharSet *permCharSet)

- int **pe_byte_align** (ASN1CTXT *ctxt_p)
- int **pe_CheckBuffer** (ASN1CTXT *ctxt_p, size_t nbytes)
- int **pe_ChoiceTypeExt** (ASN1CTXT *ctxt_p, ASN1UINT numocts, ASN1ConstOctetPtr data)
- int **pe_ConsInteger** (ASN1CTXT *ctxt_p, ASN1INT value, ASN1INT lower, ASN1INT upper)
- int **pe_ConsInt64** (ASN1CTXT *ctxt_p, ASN1INT64 value, ASN1INT64 lower, ASN1INT64 upper)
- int **pe_ConstrainedString** (ASN1CTXT *ctxt_p, ASN1ConstCharPtr string, Asn1CharSet *pCharSet)
- int **pe_ConstrainedStringEx** (ASN1CTXT *ctxt_p, ASN1ConstCharPtr string, ASN1ConstCharPtr charSet, ASN1UINT abits, ASN1UINT ubits, ASN1UINT canSetBits)
- int **pe_ConsUnsigned** (ASN1CTXT *ctxt_p, ASN1UINT value, ASN1UINT lower, ASN1UINT upper)
- int **pe_ConsUInt64** (ASN1CTXT *ctxt_p, ASN1UINT64 value, ASN1UINT64 lower, ASN1UINT64 upper)
- int **pe_ConsWholeNumber** (ASN1CTXT *ctxt_p, ASN1UINT adjusted_value, ASN1UINT range_value)
- int **pe_ConsWholeNumber64** (ASN1CTXT *ctxt_p, ASN1UINT64 adjusted_value, ASN1UINT64 range_value)
- int **pe_ExpandBuffer** (ASN1CTXT *ctxt_p, size_t nbytes)
- ASN1UINT **pe_GetIntLen** (ASN1UINT value)
- size_t **pe_GetMsgBitCnt** (ASN1CTXT *ctxt_p)
- ASN1OCTET * **pe_GetMsgPtr** (ASN1CTXT *ctxt_p, int *pLength)
- ASN1OCTET * **pe_GetMsgPtrU** (ASN1CTXT *ctxt_p, ASN1UINT *pLength)
- int **pe_Length** (ASN1CTXT *ctxt_p, ASN1UINT value)
- int **pe_NonNegBinInt** (ASN1CTXT *ctxt_p, ASN1UINT value)
- int **pe_NonNegBinInt64** (ASN1CTXT *ctxt_p, ASN1UINT64 value)
- int **pe_ObjectIdentifier** (ASN1CTXT *ctxt_p, ASN1OBJID *pvalue)
- int **pe_oid64** (ASN1CTXT *ctxt_p, ASN1OID64 *pvalue)
- int **pe_RelativeOID** (ASN1CTXT *ctxt_p, ASN1OBJID *pvalue)
- int **pe_octets** (ASN1CTXT *ctxt_p, ASN1ConstOctetPtr pvalue, ASN1UINT nbits)
- int **pe-OctetString** (ASN1CTXT *ctxt_p, ASN1UINT numocts, ASN1ConstOctetPtr data)
- int **pe_OpenType** (ASN1CTXT *ctxt_p, ASN1UINT numocts, ASN1ConstOctetPtr data)
- int **pe_OpenTypeExt** (ASN1CTXT *ctxt_p, Asn1RTDList *pElemList)
- int **pe_OpenTypeExtBits** (ASN1CTXT *ctxt_p, Asn1RTDList *pElemList)
- int **pe_Real** (ASN1CTXT *ctxt_p, ASN1REAL value)
- int **pe_SmallNonNegWholeNumber** (ASN1CTXT *ctxt_p, ASN1UINT value)
- int **pe_SemiConsInteger** (ASN1CTXT *ctxt_p, ASN1INT value, ASN1INT lower)
- int **pe_SemiConsInt64** (ASN1CTXT *ctxt_p, ASN1INT64 value, ASN1INT64 lower)
- int **pe_SemiConsUnsigned** (ASN1CTXT *ctxt_p, ASN1UINT value, ASN1UINT lower)
- int **pe_SemiConsUInt64** (ASN1CTXT *ctxt_p, ASN1UINT64 value, ASN1UINT64 lower)
- int **pe_UnconsLength** (ASN1CTXT *ctxt_p, ASN1UINT value)
- int **pe_VarWidthCharString** (ASN1CTXT *pctxt, ASN1ConstCharPtr value)
- int **pu_addSizeConstraint** (ASN1CTXT *ctxt_p, Asn1SizeCnst *pSize)
- ASN1BOOL **pu_alignCharStr** (ASN1CTXT *ctxt_p, ASN1UINT len, ASN1UINT nbits, Asn1SizeCnst *pSize)
- int **pu_checkSizeConstraint** (ASN1CTXT *ctxt_p, int size)
- ASN1UINT **pu_bitcnt** (ASN1UINT value)
- Asn1SizeCnst * **pu_checkSize** (Asn1SizeCnst *pSizeList, ASN1UINT value, ASN1BOOL *pExtendable)
- void **pu_freeContext** (ASN1CTXT *ctxt_p)
- int **pu_getBitOffset** (ASN1CTXT *ctxt_p)
- size_t **pu_getMaskAndIndex** (size_t bitOffset, unsigned char *pMask)
- size_t **pu_getMsgLen** (ASN1CTXT *ctxt_p)
- void **pu_hexdump** (ASN1CTXT *ctxt_p)
- int **pu_setBuffer** (ASN1CTXT *ctxt_p, ASN1OCTET *bufaddr, size_t bufsiz, ASN1BOOL aligned)

- int **pe_setp** (ASN1CTXT *ctxt_p, ASN1OCTET *bufaddr, size_t bufsiz, ASN1BOOL aligned)
- int **pu_initContext** (ASN1CTXT *ctxt_p, ASN1OCTET *bufaddr, ASN1UINT bufsiz, ASN1BOOL aligned)
- int **pu_initContextBuffer** (ASN1CTXT *pTarget, ASN1CTXT *pSource)
- ASN1ConstCharPtr **pu_getFullName** (ASN1CTXT *ctxt_p, ASN1ConstCharPtr suffix)
- void **pu_init16BitCharSet** (Asn116BitCharSet *pCharSet, ASN116BITCHAR first, ASN116BITCHAR last, ASN1UINT abits, ASN1UINT ubits)
- void **pu_insLenField** (ASN1CTXT *ctxt_p)
- ASN1BOOL **pu_isExtendableSize** (Asn1SizeCnst *pSizeList)
- ASN1BOOL **pu_isFixedSize** (Asn1SizeCnst *pSizeList)
- ASN1CTXT * **pu_newContext** (ASN1OCTET *bufaddr, ASN1UINT bufsiz, ASN1BOOL aligned)
- void **pu_set16BitCharSet** (ASN1CTXT *ctxt_p, Asn116BitCharSet *pCharSet, Asn116BitCharSet *pAlphabet)
- void **pu_set16BitCharSetFromRange** (Asn116BitCharSet *pCharSet, ASN1USINT firstChar, ASN1USINT lastChar)
- void **pu_setFldBitCount** (ASN1CTXT *ctxt_p)
- void **pu_setFldBitOffset** (ASN1CTXT *ctxt_p)
- ASN1BOOL **pu_setTrace** (ASN1CTXT *pCtxt, ASN1BOOL value)
- void **pu_bindump** (ASN1CTXT *ctxt_p, ASN1ConstCharPtr varname)
- void **pu_dumpField** (ASN1CTXT *ctxt_p, PERField *pField, ASN1ConstCharPtr varname, size_t nextBitOffset, BinDumpBuffer *pbuf)
- void **pu_init32BitCharSet** (Asn132BitCharSet *pCharSet, ASN132BITCHAR first, ASN132BITCHAR last, ASN1UINT abits, ASN1UINT ubits)
- void **pu_set32BitCharSet** (ASN1CTXT *ctxt_p, Asn132BitCharSet *pCharSet, Asn132BitCharSet *pAlphabet)
- void **pu_set32BitCharSetFromRange** (Asn132BitCharSet *pCharSet, ASN1UINT firstChar, ASN1UINT lastChar)
- int **pu_GetLibVersion** ()
- const char * **pu_GetLibInfo** ()

asn1PerCppType.h File Reference

Detailed Description

PER C++ type and class definitions.

```
#include "asn1per.h"
```

```
#include "asn1CppType.h"
```

Classes

- class **ASN1PERMessageBuffer**
- class **ASN1PEREncodeBuffer**
- class **ASN1PERDecodeBuffer**

Index

INDEX