H323FrameworkStack

objsys
Version 0.4
1/21/2005 2:42 PM

Table of Contents

H323FrameworkStack Module Index

H323FrameworkStack Modules

Here is a list of all modules:

H323FrameworkStack Data Structure Index

H323FrameworkStack Data Structures

Here are the data structures with brief descriptions:

H323FrameworkStack File Index

H323FrameworkStack File List

Here is a list of all documented files with brief descriptions:

H323FrameworkStack Module Documentation

Asn1CEventHandler

Asn1CEventHandler is a structure type used for user-defined event handlers.

Data Structures

- struct **Asn1NamedCEventHandler**

This is a basic C based event handler structure, which can be used to define user-defined event handlers.

Typedefs

- typedef void(* **rtStartElement**)(const char *name, int index)

This is a function pointer for a callback function which is invoked from within a decode function when an element of a SEQUENCE, SET, SEQUENCE OF, SET OF, or CHOICE construct is parsed.

- typedef void(* **rtEndElement**)(const char *name, int index)

This is a function pointer for a callback function which is invoked from within a decode function when parsing is complete on an element of a SEQUENCE, SET, SEQUENCE OF, SET OF, or CHOICE construct.

- typedef void(* **rtBoolValue**)(ASN1BOOL value)

This is a function pointer for a callback function which is invoked from within a decode function when a value of the BOOLEAN ASN.1 type is parsed.

- typedef void(* **rtIntValue**)(ASN1INT value)

This is a function pointer for a callback function which is invoked from within a decode function when a value of the INTERGER ASN.1 type is parsed.

- typedef void(* **rtUIntValue**)(ASN1UINT value)

This is a function pointer for a callback function which is invoked from within a decode function when a value of the INTEGER ASN.1 type is parsed.

- typedef void(* **rtBitStrValue**)(ASN1UINT numbits, const ASN1OCTET *data)

This is a function pointer for a callback function which is invoked from within a decode function when a value of the BIT STRING ASN.1 type is parsed.

- typedef void(* **rtOctStrValue**)(ASN1UINT numocts, const ASN1OCTET *data)

This is a function pointer for a callback function which is invoked from within a decode function when a value of one of the OCTET STRING ASN.1 type is parsed.

- typedef void(* **rtCharStrValue**)(const char *value)

This is a function pointer for a callback function which is invoked from within a decode function when a value of one of the 8-bit ASN.1 character string types is parsed.

- typedef void(* **rtCharStrValue16Bit**)(ASN1UINT nchars, ASN116BITCHAR *data)

This is a function pointer for a callback function which is invoked from within a decode function when a value of one of the 16-bit ASN.1 character string types is parsed.

- typedef void(* **rtCharStrValue32Bit**)(ASN1UINT nchars, ASN132BITCHAR *data)

This is a function pointer for a callback function which is invoked from within a decode function when a value of one of the 32-bit ASN.1 characer string types is parsed.

- typedef void(* **rtNullValue**)()

This is a function pointer for a callback function which is invoked from within a decode function when a value of the NULL ASN.1 type is parsed.

- typedef void(* **rtOidValue**)(ASN1UINT numSubIds, ASN1UINT *pSubIds)

This is a function pointer for a callback function which is invoked from within a decode function whn a value the OBJECT IDENTIFIER ASN.1 type is parsed.

- typedef void(* **rtRealValue**)(double value)

This is a function pointer for a callback function which is invoked from within a decode function when a value the REAL ASN.1 type is parsed.

- typedef void(* **rtEnumValue**)(ASN1UINT value)

This is a function pointer for a callback function which is invoked from within a decode function when a value of the ENUMERATED ASN.1 type is parsed.

- typedef void(* **rtOpenTypeValue**)(ASN1UINT numocts, const ASN1OCTET *data)

This is a function pointer for a callback function which is invoked from within a decode function when an ASN.1 open type is parsed.

- typedef **Asn1NamedCEventHandler Asn1NamedCEventHandler**

This is a basic C based event handler structure, which can be used to define user-defined event handlers.

Functions

- EXTERN void **rtAddEventHandler** (OOCTXT *pCtxt, **Asn1NamedCEventHandler** *pHandler)

This function is called to add a new event handler to the context event handler list.

- EXTERN void **rtRemoveEventHandler** (OOCTXT *pCtxt, **Asn1NamedCEventHandler** *pHandler)

This function is called to remove an event handler from the context event handler list.

- EXTERN void **rtInvokeStartElement** (OOCTXT *pCtxt, const char *name, int index)

The following functions are invoked from within the generated code to call the various user-defined event handler methods ..

- EXTERN void **rtInvokeEndElement** (OOCTXT *pCtxt, const char *name, int index)

- EXTERN void **rtInvokeBoolValue** (OOCTXT *pCtxt, ASN1BOOL value)
- EXTERN void **rtInvokeIntValue** (OOCTXT *pCtxt, ASN1INT value)
- EXTERN void **rtInvokeUIntValue** (OOCTXT *pCtxt, ASN1UINT value)
- EXTERN void **rtInvokeBitStrValue** (OOCTXT *pCtxt, ASN1UINT numbits, const ASN1OCTET *data)
- EXTERN void **rtInvokeOctStrValue** (OOCTXT *pCtxt, ASN1UINT numocts, const ASN1OCTET *data)
- EXTERN void **rtInvokeCharStrValue** (OOCTXT *pCtxt, const char *value)
- EXTERN void **rtInvokeCharStr16BitValue** (OOCTXT *pCtxt, ASN1UINT nchars, ASN116BITCHAR *data)
- EXTERN void **rtInvokeCharStr32BitValue** (OOCTXT *pCtxt, ASN1UINT nchars, ASN132BITCHAR *data)
- EXTERN void **rtInvokeNullValue** (OOCTXT *pCtxt)
- EXTERN void **rtInvokeOidValue** (OOCTXT *pCtxt, ASN1UINT numSubIds, ASN1UINT *pSubIds)
- EXTERN void **rtInvokeRealValue** (OOCTXT *pCtxt, double value)
- EXTERN void **rtInvokeEnumValue** (OOCTXT *pCtxt, ASN1UINT value)
- EXTERN void **rtInvokeOpenTypeValue** (OOCTXT *pCtxt, ASN1UINT numocts, const ASN1OCTET *data)

Detailed Description

Asn1CEventHandler is a structure type used for user-defined event handlers.

Typedef Documentation

typedef void(* rtBitStrValue)(ASN1UINT numbits, const ASN1OCTET* data)

This is a function pointer for a callback function which is invoked from within a decode function when a value of the BIT STRING ASN.1 type is parsed.

Parameters:

numbits - Number of bits in the parsed value.
data - Pointer to a byte array that contains the bit string data.

Returns:

- none

Definition at line 126 of file asn1CEvtHndlr.h.

typedef void(* rtBoolValue)(ASN1BOOL value)

This is a function pointer for a callback function which is invoked from within a decode function when a value of the BOOLEAN ASN.1 type is parsed.

Parameters:

value Parsed value.

Returns:

- none

Definition at line 94 of file asn1CEvtHndlr.h.

typedef void(* rtCharStrValue)(const char* value)

This is a function pointer for a callback function which is invoked from within a decode function when a value of one of the 8-bit ASN.1 character string types is parsed.

Parameters:

value Null terminated character string value.

Returns:

- none

Definition at line 148 of file asn1CEvtHndlr.h.

typedef void(* rtCharStrValue16Bit)(ASN1UINT nchars, ASN116BITCHAR* data)

This is a function pointer for a callback function which is invoked from within a decode function when a value of one of the 16-bit ASN.1 character string types is parsed.

This is used for the ASN.1 BmpString type.

Parameters:

nchars Number of characters in the parsed value.
data Pointer to an array containing 16-bit values. These are represented using unsigned short integer values.

Returns:

- none

Definition at line 163 of file asn1CEvtHndlr.h.

typedef void(* rtCharStrValue32Bit)(ASN1UINT nchars, ASN132BITCHAR* data)

This is a function pointer for a callback function which is invoked from within a decode function when a value of one of the 32-bit ASN.1 characer string types is parsed.

This is used for the ASN.1 UniversalString type.

Parameters:

nchars Number of characters in the parsed value.
data Pointer to an array containing 32-bit values. Each 32-bit integer value is a universal character.

Returns:

- none

Definition at line 177 of file asn1CEvtHndlr.h.

typedef void(* rtEndElement)(const char* name, int index)

This is a function pointer for a callback function which is invoked from within a decode function when parsing is complete on an element of a SEQUENCE, SET, SEQUENCE OF, SET OF, or CHOICE construct.

Parameters:

name For SEQUENCE, SET, or CHOICE, this is the name of the element as defined in the ASN.1 defination. For SEQUENCE OF or SET OF, this is set to the name "element".
index For SEQUENCE, SET, or CHOICE, this is not used and is set to the value -1. For SEQUENCE OF or SET OF, this contains the zero-based index of the element in the conceptual array associated with the construct.

Returns:

- none

Definition at line 84 of file asn1CEvtHndlr.h.

typedef void(* rtEnumValue)(ASN1UINT value)

This is a function pointer for a callback function which is invoked from within a decode function when a value of the ENUMERATED ASN.1 type is parsed.

Parameters:

value - Parsed enumerated value

Returns:

- none

Definition at line 216 of file asn1CEvtHndlr.h.

typedef void(* rtIntValue)(ASN1INT value)

This is a function pointer for a callback function which is invoked from within a decode function when a value of the INTERGER ASN.1 type is parsed.

Parameters:

value Parsed value.

Returns:

- none

Definition at line 103 of file asn1CEvtHndlr.h.

typedef void(* rtNullValue)()

This is a function pointer for a callback function which is invoked from within a decode function when a value of the NULL ASN.1 type is parsed.

Parameters:

- none

Returns:

- none

Definition at line 186 of file asn1CEvtHndlr.h.

typedef void(* rtOctStrValue)(ASN1UINT numocts, const ASN1OCTET* data)

This is a function pointer for a callback function which is invoked from within a decode function when a value of one of the OCTET STRING ASN.1 type is parsed.

Parameters:

numocts Number of octets in the parsed value.
data Pointer to byte array containing the octet string data.

Returns:

- none

Definition at line 138 of file asn1CEvtHndlr.h.

typedef void(* rtOidValue)(ASN1UINT numSubIds, ASN1UINT* pSubIds)

This is a function pointer for a callback function which is invoked from within a decode function whn a value the OBJECT IDENTIFIER ASN.1 type is parsed.

Parameters:

numSubIds Number of subidentifiers in the object identifier.
pSubIds Pointer to array containing the subidentifier values.

Returns:

-none

Definition at line 197 of file asn1CEvtHndlr.h.

typedef void(* rtOpenTypeValue)(ASN1UINT numocts, const ASN1OCTET* data)

This is a function pointer for a callback function which is invoked from within a decode function when an ASN.1 open type is parsed.

Parameters:

numocts Number of octets in the parsed value.
data Pointer to byet array contain in tencoded ASN.1 value.

Returns:

- none

Definition at line 227 of file asn1CEvtHndlr.h.

typedef void(* rtRealValue)(double value)

This is a function pointer for a callback function which is invoked from within a decode function when a value the REAL ASN.1 type is parsed.

Parameters:

value Parsed value.

Returns:

- none

Definition at line 206 of file asn1CEvtHndlr.h.

typedef void(* rtStartElement)(const char* name, int index)

This is a function pointer for a callback function which is invoked from within a decode function when an element of a SEQUENCE, SET, SEQUENCE OF, SET OF, or CHOICE construct is parsed.

Parameters:

name For SEQUENCE, SET, or CHOICE, this is the name of the element as defined in the ASN.1 defination. For SEQUENCE OF or SET OF, this is set to the name "element".
index For SEQUENCE, SET, or CHOICE, this is not used and is set to the value -1. For SEQUENCE OF or SET OF, this contains the zero-based index of the element in the conceptual array associated with the construct.

Returns:

- none

Definition at line 65 of file asn1CEvtHndlr.h.

typedef void(* rtUIntValue)(ASN1UINT value)

This is a function pointer for a callback function which is invoked from within a decode function when a value of the INTEGER ASN.1 type is parsed.

In this case, constraints on the integer value forced the use of unsigned integer C type to represent the value.

Parameters:

value Parsed value.

Returns:

- none

Definition at line 114 of file asn1CEvtHndlr.h.

Function Documentation

EXTERN void rtAddEventHandler (OOCTXT * *pCtxt*, Asn1NamedCEventHandler * *pHandler*)

This function is called to add a new event handler to the context event handler list.

Parameters:

pCtxt Context to which event handler has to be added.
pHandler Pointer to the event handler structure.

Returns:

none

EXTERN void rtRemoveEventHandler (OOCTXT * *pCtxt*, Asn1NamedCEventHandler * *pHandler*)

This function is called to remove an event handler from the context event handler list.

Note that it does not delete the event handler object.

Parameters:

pCtxt Context from which event handler has to be removed.
pHandler Pointer to event handler structure.

Returns:

none

C Runtime Common Functions

Modules

- group **Memory Allocation Macros and Functions**
- group **Context Management Functions**
- group **Linked List Utility Functions**

Linked list and stack utility functions are used to maintain linked lists and stacks used within the ASN.1 run-time library functions.

-

Data Structures

- struct **ASN1OBJID**
- struct **ASN1OctStr**
- struct **ASN1DynOctStr**
- struct **ASN1DynBitStr**
- struct **ASN1SeqOf**
- struct **ASN1SeqOfOctStr**
- struct **ASN1OpenType**
- struct **Asn116BitCharString**
- struct **Asn132BitCharString**
- struct **Asn1CharArray**
- struct **Asn1CharSet**
- struct **Asn116BitCharSet**
- struct **_SListNode**
- struct **_SList**
- struct **_DListNode**
- struct **_DList**
- struct **_Asn1SizeCnst**
- struct **ASN1BUFFER**
- struct **ASN1BUFSAVE**
- struct **ASN1ErrLocn**
- struct **ASN1ErrInfo**
- struct **OOCTXT**

Defines

- #define **TV_UNIV** 0 /* universal */
- #define **TV_APPL** 1 /* application-wide */
- #define **TV_CTXT** 2 /* context-specific */
- #define **TV_PRIV** 3 /* private-use */

- #define **TV_PRIM** 0 /* primitive */
- #define **TV_CONS** 1 /* constructor */
- #define **TM_UNIV** 0x00000000 /* universal class */
- #define **TM_APPL** 0x40000000 /* application-wide class */
- #define **TM_CTXT** 0x80000000 /* context-specific class */
- #define **TM_PRIV** 0xC0000000 /* private-use class */
- #define **TM_PRIM** 0x00000000 /* primitive form */
- #define **TM_CONS** 0x20000000 /* constructor form */
- #define **TM_IDCODE** 0x1FFFFFFF /* ID code mask */
- #define **ASN_K_BADTAG** 0xFFFFFFFF /* invalid tag code */
- #define **ASN_K_NOTAG** 0xFFFFFFFF /* no tag input parameter */
- #define **TM_CLASS** 0xC0 /* class mask */
- #define **TM_FORM** 0x20 /* form mask */
- #define **TM_CLASS_FORM** 0xE0 /* class/form mask */
- #define **TM_B_IDCODE** 0x1F /* id code mask (byte) */
- #define **MINMSGLEN** 8 /* minimum message length */
- #define **ASN_OK** 0 /* normal completion status */
- #define **ASN_OK_FRAG** 2 /* message fragment detected */
- #define **ASN_E_BUFOVFLW** -1 /* encode buffer overflow */
- #define **ASN_E_ENDOFBUF** -2 /* unexpected end of buffer on decode */
- #define **ASN_E_IDNOTFOU** -3 /* identifer not found */
- #define **ASN_E_INVOBJID** -4 /* invalid object identifier */
- #define **ASN_E_INVLEN** -5 /* invalid field length */
- #define **ASN_E_INVENUM** -6 /* enumerated value not in defined set */
- #define **ASN_E_SETDUPL** -7 /* duplicate element in set */
- #define **ASN_E_SETMISRQ** -8 /* missing required element in set */
- #define **ASN_E_NOTINSET** -9 /* element not part of set */
- #define **ASN_E_SEQOVFLW** -10 /* sequence of field overflow */
- #define **ASN_E_INVOPT** -11 /* invalid option encountered in choice */
- #define **ASN_E_NOMEM** -12 /* no dynamic memory available */
- #define **ASN_E_INVHEXS** -14 /* invalid hex string */
- #define **ASN_E_INVBINS** -15 /* invalid binary string */
- #define **ASN_E_INVREAL** -16 /* invalid real value */
- #define **ASN_E_STROVFLW** -17 /* octet or bit string field overflow */
- #define **ASN_E_BADVALUE** -18 /* invalid value specification */
- #define **ASN_E_UNDEFVAL** -19 /* no def found for ref'd defined value */
- #define **ASN_E_UNDEFTYP** -20 /* no def found for ref'd defined type */
- #define **ASN_E_BADTAG** -21 /* invalid tag value */
- #define **ASN_E_TOODEEP** -22 /* nesting level is too deep */
- #define **ASN_E_CONSVIO** -23 /* value constraint violation */
- #define **ASN_E_RANGERR** -24 /* invalid range (lower > upper) */
- #define **ASN_E_ENDOFFILE** -25 /* end of file on file decode */
- #define **ASN_E_INVUTF8** -26 /* invalid UTF-8 encoding */
- #define **ASN_E_CONCMODF** -27 /* Concurrent list modification */
- #define **ASN_E_ILLSTATE** -28 /* Illegal state error */
- #define **ASN_E_OUTOFBND** -29 /* out of bounds (of array, etc) */
- #define **ASN_E_INVPARAM** -30 /* invalid parameter */
- #define **ASN_E_INVFORMAT** -31 /* invalid time string format */
- #define **ASN_E_NOTINIT** -32 /* not initialized */
- #define **ASN_E_TOOBIG** -33 /* value is too big for given data type */
- #define **ASN_E_INVCHAR** -34 /* invalid character (not in char set) */
- #define **ASN_E_XMLSTATE** -35 /* XML state error */
- #define **ASN_E_XMLPARSE** -36 /* XML parse error */
- #define **ASN_E_SEQORDER** -37 /* SEQUENCE elements not in order */
- #define **ASN_E_INVINDEX** -38 /* invalid index for TC id */

- #define **ASN_E_INVTCVAL** -39 /* invalid value for TC field */
- #define **ASN_E_FILNOTFOU** -40 /* file not found */
- #define **ASN_E_FILEREAD** -41 /* error occurred reading file */
- #define **ASN_E_FILEWRITE** -42 /* error occurred writing file */
- #define **ASN_E_INVBASE64** -43 /* invalid base64 encoding */
- #define **ASN_E_INVSOCKET** -44 /* invalid socket operation */
- #define **ASN_E_XMLLIBNFOU** -45 /* XML library is not found */
- #define **ASN_E_XMLLIBINV** -46 /* XML library is invalid */
- #define **ASN_E_NOTSUPP** -99 /* non-supported ASN construct */
- #define **ASN_K_INDEFLEN** -9999 /* indefinite length message indicator */
- #define **ASN_ID_EOC** 0 /* end of contents */
- #define **ASN_ID_BOOL** 1 /* boolean */
- #define **ASN_ID_INT** 2 /* integer */
- #define **ASN_ID_BITSTR** 3 /* bit string */
- #define **ASN_ID_OCTSTR** 4 /* byte (octet) string */
- #define **ASN_ID_NULL** 5 /* null */
- #define **ASN_ID_OBJID** 6 /* object ID */
- #define **ASN_ID_OBJDSC** 7 /* object descriptor */
- #define **ASN_ID_EXTERN** 8 /* external type */
- #define **ASN_ID_REAL** 9 /* real */
- #define **ASN_ID_ENUM** 10 /* enumerated value */
- #define **ASN_ID_EPDV** 11 /* EmbeddedPDV type */
- #define **ASN_ID_RELOID** 13 /* relative object ID */
- #define **ASN_ID_SEQ** 16 /* sequence, sequence of */
- #define **ASN_ID_SET** 17 /* set, set of */
- #define **ASN_SEQ_TAG** 0x30 /* SEQUENCE universal tag byte */
- #define **ASN_SET_TAG** 0x31 /* SET universal tag byte */
- #define **ASN_ID_NumericString** 18
- #define **ASN_ID_PrintableString** 19
- #define **ASN_ID_TeletexString** 20
- #define **ASN_ID_T61String** ASN_ID_TeletexString
- #define **ASN_ID_VideotexString** 21
- #define **ASN_ID_IA5String** 22
- #define **ASN_ID_UTCTime** 23
- #define **ASN_ID_GeneralTime** 24
- #define **ASN_ID_GraphicString** 25
- #define **ASN_ID_VisibleString** 26
- #define **ASN_ID_GeneralString** 27
- #define **ASN_ID_UniversalString** 28
- #define **ASN_ID_BMPString** 30
- #define **XM_SEEK** 0x01 /* seek match until found or end-of-buf */
- #define **XM_ADVANCE** 0x02 /* advance pointer to contents on match */
- #define **XM_DYNAMIC** 0x04 /* alloc dyn mem for decoded variable */
- #define **XM_SKIP** 0x08 /* skip to next field after parsing tag */
- #define **ASN_K_MAXDEPTH** 32 /* maximum nesting depth for messages */
- #define **ASN_K_MAXSUBIDS** 128 /* maximum sub-id's in an object ID */
- #define **ASN_K_MAXENUM** 100 /* maximum enum values in an enum type */
- #define **ASN_K_MAXERRP** 5 /* maximum error parameters */
- #define **ASN_K_MAXERRSTK** 8 /* maximum levels on error ctxt stack */
- #define **ASN_K_ENCBUFSIZ** 16*1024 /* dynamic encode buffer extent size */
- #define **ASN_K_MEMBUFSEG** 1024 /* memory buffer extent size */
- #define **NUM_ABITS** 4
- #define **NUM_UBITS** 4
- #define **NUM_CANSET** " 0123456789"
- #define **PRN_ABITS** 8

- #define **PRN_UBITS** 7
- #define **PRN_CANSET** "
'()+,-./0123456789:=?ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
- #define **VIS_ABITS** 8
- #define **VIS_UBITS** 7
- #define **VIS_CANSET**
- #define **T61_ABITS** 8
- #define **T61_UBITS** 7
- #define **T61_CANSET**
- #define **IA5_ABITS** 8
- #define **IA5_UBITS** 7
- #define **IA5_CANSET**
- #define **IA5_RANGE1_LOWER** 0
- #define **IA5_RANGE2_LOWER** 0x5f
- #define **GEN_ABITS** 8
- #define **GEN_UBITS** 7
- #define **GEN_CANSET**
- #define **BMP_ABITS** 16
- #define **BMP_UBITS** 16
- #define **BMP_FIRST** 0
- #define **BMP_LAST** 0xffff
- #define **UCS_ABITS** 32
- #define **UCS_UBITS** 32
- #define **UCS_FIRST** 0
- #define **UCS_LAST** 0xffffffful
- #define **ASN1TAG_LSHIFT** 24
- #define **ASN1UINT_MAX** 4294967295U
- #define **ASN1INT_MAX** ((ASN1INT)2147483647L)
- #define **ASN1INT_MIN** ((ASN1INT)(-ASN1INT_MAX-1))
- #define **ASN1INT64** long
- #define **FALSE** 0
- #define **TRUE** 1
- #define **XM_K_MEMBLKSIZ** (4*1024)
- #define **ASN1DYNCTXT** 0x8000
- #define **ASN1INDEFLEN** 0x4000
- #define **ASN1TRACE** 0x2000
- #define **ASN1LASTEOC** 0x1000
- #define **ASN1FASTCOPY** 0x0800 /* turns on the "fast copy" mode */
- #define **ASN1CONSTAG** 0x0400 /* form of last parsed tag */
- #define **ASN1CANXER** 0x0200 /* canonical XER */
- #define **ASN1SAVEBUF** 0x0100 /* do not free dynamic encode buffer */
- #define **ASN1OPENTYPE** 0x0080 /* item is an open type field */
- #define **ASN1MAX**(a, b) (((a)>(b))?(a):(b))
- #define **ASN1MIN**(a, b) (((a)<(b))?(a):(b))
- #define **ASN1BUFCUR**(cp) (cp)->buffer.data[(cp)->buffer.byteIndex]
- #define **ASN1BUFPTR**(cp) &(cp)->buffer.data[(cp)->buffer.byteIndex]
- #define **EXTERN**
- #define **ASN1CRTMALLOC0**(nbytes) malloc(nbytes)
- #define **ASN1CRTFREE0**(ptr) free(ptr)
- #define **ASN1CRTMALLOC** memHeapAlloc
- #define **ASN1CRTFREE** ASN1MEMFREEPTR
- #define **DE_INCRBITIDX**(pctxt)
- #define **DE_BIT**(pctxt, pvalue)
- #define **encodeIA5String**(pctxt, value, permCharSet) encodeConstrainedStringEx(pctxt, value, permCharSet, 8, 7, 7)

- #define **encodeGeneralizedTime** encodeIA5String
- #define **decodeIA5String**(pctxt, pvalue, permCharSet) decodeConstrainedStringEx(pctxt, pvalue, permCharSet, 8, 7, 7)
- #define **decodeGeneralizedTime** decodeIA5String

Typedefs

- typedef char **ASN1CHAR**
- typedef unsigned char **ASN1OCTET**
- typedef ASN1OCTET **ASN1BOOL**
- typedef signed char **ASN1INT8**
- typedef unsigned char **ASN1UINT8**
- typedef int **ASN1INT**
- typedef unsigned int **ASN1UINT**
- typedef ASN1INT **ASN1ENUM**
- typedef double **ASN1REAL**
- typedef short **ASN1SINT**
- typedef unsigned short **ASN1USINT**
- typedef ASN1UINT **ASN1TAG**
- typedef ASN1USINT **ASN116BITCHAR**
- typedef ASN1UINT **ASN132BITCHAR**
- typedef void * **ASN1ANY**
- typedef const char * **ASN1GeneralizedTime**
- typedef const char * **ASN1GeneralString**
- typedef const char * **ASN1GraphicString**
- typedef const char * **ASN1IA5String**
- typedef const char * **ASN1ISO646String**
- typedef const char * **ASN1NumericString**
- typedef const char * **ASN1ObjectDescriptor**
- typedef const char * **ASN1PrintableString**
- typedef const char * **ASN1TeletexString**
- typedef const char * **ASN1T61String**
- typedef const char * **ASN1UTCTime**
- typedef const char * **ASN1UTF8String**
- typedef const char * **ASN1VideotexString**
- typedef const char * **ASN1VisibleString**
- typedef Asn116BitCharString **ASN1BMPString**
- typedef Asn132BitCharString **ASN1UniversalString**
- typedef _SListNode **SListNode**
- typedef _SList **SList**
- typedef _DListNode **DListNode**
- typedef _DList **DList**
- typedef _Asn1SizeCnst **Asn1SizeCnst**
- typedef OOCTXT **OOCTXT**

Define Documentation

#define DE_BIT(pctxt, pvalue)

Value:

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

Definition at line 603 of file ooasn1.h.

#define DE_INCRBITIDX(pctxt)

Value:

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

Definition at line 597 of file ooasn1.h.

#define GEN_CANSET

Value:

```
"\000\001\002\003\004\005\006\007\010\011\012\013\014\015\016\017"\
"\020\021\022\023\024\025\026\027\030\031\032\033\034\035\036\037"\
" !\"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_"\
"`abcdefghijklmnopqrstuvwxyz{|}~\177\200\201\202\203\204\205\206\207"\
"\220\221\222\223\224\225\226\227\230\231\232\233\234\235\236\237"\
"\240\241\242\243\244\245\246\247\250\251\252\253\254\255\256\257"\
"\260\261\262\263\264\265\266\267\270\271\272\273\274\275\276\277"\
"\300\301\302\303\304\305\306\307\310\311\312\313\314\315\316\317"\
"\320\321\322\323\324\325\326\327\330\331\332\333\334\335\336\337"\
"\340\341\342\343\344\345\346\347\350\351\352\353\354\355\356\357"\
"\360\361\362\363\364\365\366\367\370\371\372\373\374\375\376\377"
```

Definition at line 212 of file ooasn1.h.

#define IA5_CANSET

Value:

```
"\000\001\002\003\004\005\006\007\010\011\012\013\014\015\016\017"\
"\020\021\022\023\024\025\026\027\030\031\032\033\034\035\036\037"\
" !\"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]"\
"^_`abcdefghijklmnopqrstuvwxyz{|}~"
```

Definition at line 201 of file ooasn1.h.

#define T61_CANSET

Value:

```
" !\"%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[]"\
"_abcdefghijklmnopqrstuvwxyz"
```

Definition at line 195 of file ooasn1.h.

#define VIS_CANSET

Value:

```
" !\"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]"\
"^_`abcdefghijklmnopqrstuvwxyz{|}~"
```

Definition at line 189 of file ooasn1.h.

Memory Allocation Macros and Functions

Defines

- #define **ALLOC_ASN1ARRAY**(pctxt, pseqof, type)

Allocate a dynamic array.

- #define **ALLOC_ASN1ELEM**(pctxt, type) (type*) memHeapAllocZ (&(pctxt)->pTypeMemHeap, sizeof(type))

Allocate and zero an ASN.1 element.

- #define **ALLOC_ASN1ELEMDNODE**(pctxt, type)
- #define **ASN1MALLOC**(pctxt, nbytes) memHeapAlloc(&(pctxt)->pTypeMemHeap, nbytes)

Allocate memory.

- #define **ASN1MEMFREE**(pctxt) memHeapFreeAll(&(pctxt)->pTypeMemHeap)

Free memory associated with a context.

- #define **ASN1MEMFREEPTR**(pctxt, pmem) memHeapFreePtr(&(pctxt)->pTypeMemHeap, (void*)pmem)

Free memory pointer.

Detailed Description

Memory allocation functions and macros handle memory management for the ASN1C run-time. Special algorithms are used for allocation and deallocation of memory to improve the run-time performance.

Define Documentation

#define ALLOC_ASN1ARRAY(pctxt, pseqof, type)

Value:

```
do {\
if (sizeof(type)*(pseqof)->n < (pseqof)->n) return ASN_E_NOMEM; \
if (((pseqof)->elem = (type*) memHeapAlloc \
(&(pctxt)->pTypeMemHeap, sizeof(type)*(pseqof)->n)) == 0) return ASN_E_NOMEM;
\
} while (0)
```

Allocate a dynamic array. This macro allocates a dynamic array of records of the given type. This version of the macro will return the ASN_E_NOMEM error status if the memory request cannot be fulfilled.

Parameters:

pctxt - Pointer to a context block
pseqof - Pointer to a generated SEQUENCE OF array structure. The *n* member variable must be set to the number of records to allocate.
type - Data type of an array record

Definition at line 497 of file ooasn1.h.

#define ALLOC_ASN1ELEM(pctxt, type) (type*) memHeapAllocZ (&(pctxt)->pTypeMemHeap, sizeof(type))

Allocate and zero an ASN.1 element.

This macro allocates and zeros a single element of the given type.

Parameters:

pctxt - Pointer to a context block
type - Data type of record to allocate

Definition at line 510 of file ooasn1.h.

#define ALLOC_ASN1ELEMDNODE(pctxt, type)

Value:

```
(type*) (((char*)memHeapAllocZ (&(pctxt)->pTypeMemHeap, sizeof(type) + \
sizeof(DListNode))) + sizeof(DListNode))
```

Definition at line 513 of file ooasn1.h.

#define ASN1MALLOC(pctxt, nbytes) memHeapAlloc(&(pctxt)->pTypeMemHeap, nbytes)

Allocate memory.

This macro allocates the given number of bytes. It is similar to the C `malloc` run-time function.

Parameters:

pctxt - Pointer to a context block
nbytes - Number of bytes of memory to allocate

Returns:

- Void pointer to allocated memory or NULL if insufficient memory was available to fulfill the request.

Definition at line 527 of file ooasn1.h.

#define ASN1MEMFREE(pctxt) memHeapFreeAll(&(pctxt)->pTypeMemHeap)

Free memory associated with a context.

This macro frees all memory held within a context. This is all memory allocated using the ASN1MALLOC (and similar macros) and the mem memory allocation functions using the given context variable.

Parameters:

pctxt - Pointer to a context block

Definition at line 538 of file ooasn1.h.

#define ASN1MEMFREEPTR(pctxt, pmem) memHeapFreePtr(&(pctxt)->pTypeMemHeap, (void*)pmem)

Free memory pointer.

This macro frees memory at the given pointer. The memory must have been allocated using the ASN1MALLOC (or similar) macros or the mem memory allocation functions. This macro is similar to the C `free` function.

Parameters:

pctxt - Pointer to a context block
pmem - Pointer to memory block to free. This must have been allocated using the ASN1MALLOC macro or the memHeapAlloc function.

Definition at line 552 of file ooasn1.h.

Context Management Functions

Defines

- #define **ZEROCONTEXT**(pctxt) memset(pctxt,0,sizeof(OOCTXT))

Functions

- EXTERN int **initContextBuffer** (OOCTXT *pctxt, const ASN1OCTET *bufaddr, ASN1UINT bufsiz)

This function assigns a buffer to a context block.

- EXTERN int **initContext** (OOCTXT *pctxt)

This function initializes a context block.

- EXTERN void **freeContext** (OOCTXT *pctxt)

This function frees all dynamic memory associated with a context.

- EXTERN OOCTXT * **newContext** ()

This function allocates a new OOCTXT block and initializes it.

- EXTERN void **copyContext** (OOCTXT *pdest, OOCTXT *psrc)
- EXTERN int **initSubContext** (OOCTXT *pctxt, OOCTXT *psrc)
- EXTERN void **setCtxtFlag** (OOCTXT *pctxt, ASN1USINT mask)
- EXTERN void **clearCtxtFlag** (OOCTXT *pctxt, ASN1USINT mask)
- EXTERN int **setPERBuffer** (OOCTXT *pctxt, ASN1OCTET *bufaddr, ASN1UINT bufsiz, ASN1BOOL aligned)
- EXTERN int **setPERBufferUsingCtxt** (OOCTXT *pTarget, OOCTXT *pSource)

Detailed Description

Context initialization functions handle the allocation, initialization, and destruction of ASN.1 context variables (variables of type OOCTXT). These variables hold all of the working data used during the process of encoding or decoding a message. The context provides thread safe operation by isolating what would otherwise be global variables within this structure. The context variable is passed from function to function as a message is encoded or decoded and maintains state information on the encoding or decoding process.

Function Documentation

EXTERN void freeContext (OOCTXT * *pctxt*)

This function frees all dynamic memory associated with a context.

This includes all memory inside the block (in particular, the list of memory blocks used by the mem functions).

Parameters:

pctxt A pointer to a context structure.

EXTERN int initContext (OOCTXT * *pctxt*)

This function initializes a context block.

It makes sure that if the block was not previosly initialized, that all key working parameters are set to thier correct initial state values (i.e. declared within a function as a normal working variable), it is required that they invoke this function before using it.

Parameters:

pctxt The pointer to the context structure variable to be initialized.

Returns:

Completion status of operation:

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

EXTERN int initContextBuffer (OOCTXT * *pctxt*, const ASN1OCTET * *bufaddr*, ASN1UINT *bufsiz*)

This function assigns a buffer to a context block.

The block should have been previously initialized by initContext.

Parameters:

pctxt The pointer to the context structure variable to be initialized.
bufaddr For encoding, the address of a memory buffer to receive and encode a message. For decoding the address of a buffer that contains the message data to be decoded. This address will be stored within the context structure. For encoding it might be zero, the dynamic buffer will be used in this case.
bufsiz The size of the memory buffer. For encoding, it might be zero; the dynamic buffer will be used in this case.

Returns:

Completion status of operation:

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

EXTERN OOCTXT* newContext ()

This function allocates a new OOCTXT block and initializes it.

Although the block is allocated from the standard heap, it should not be freed using free. The freeContext function should be used because this frees items allocated within the block before freeing the block itself.

Returns:

Pointer to newly created context

Linked List Utility Functions

Linked list and stack utility functions are used to maintain linked lists and stacks used within the ASN.1

run-time library functions.

Modules

- group **Error Formatting and Print Functions**
- group **Rtmem**

Defines

- #define **RT_MH_DONTKEEPFREE** 0x1
- #define **OSRTMH_PROPID_DEFBLKSIZE** 1
- #define **OSRTMH_PROPID_SETFLAGS** 2
- #define **OSRTMH_PROPID_CLEARFLAGS** 3
- #define **OSRTMH_PROPID_USER** 10

Functions

- EXTERN DListNode * **dListAppend** (OOCTXT *pctxt, DList *pList, void *pData)

This function appends an item to the linked list structure.

- EXTERN DListNode * **dListAppendNode** (OOCTXT *pctxt, DList *pList, void *pData)
- EXTERN DListNode * **dListFindByIndex** (DList *pList, int index)
- EXTERN void **dListInit** (DList *pList)

This function initializes a doubly linked list structure.

- EXTERN void **dListFreeNodes** (OOCTXT *pctxt, DList *pList)

This function removes all nodes from the linked list and releases the memory that was allocated for storing the node structures (DListNode).

- EXTERN void **dListFreeAll** (OOCTXT *pctxt, DList *pList)

This function removes all nodes from the linked list structure and releases the memory that was allocated for storing the node structures (DListNode) and for data.

- EXTERN void **dListRemove** (DList *pList, DListNode *node)

This function removes a node from the linked list structure.

- EXTERN void **sListInit** (SList *pList)

This function is used to initialize a singly-linked list.

- EXTERN void **sListInitEx** (OOCTXT *pctxt, SList *pList)

This function is used to initialize a singly-linked list and assigns a context to be used for the list.

- EXTERN void **sListFree** (SList *pList)

This function is used to free-up all the nodes in the singly-linked list.

- EXTERN SList * **sListCreate** ()

This function is used to create a new singly-linked list.

- EXTERN SList * **sListCreateEx** (OOCTXT *pctxt)

This function is used to create a singly-linked list.

- EXTERN SListNode * **sListAppend** (SList *pList, void *pData)

This function is used to append a new data member to the list.

- EXTERN ASN1BOOL **sListFind** (SList *pList, void *pData)

This function is used to search for a particular data in the list.

- EXTERN void **sListRemove** (SList *pList, void *pData)

This function is used to remove a particular data member from the list.

Detailed Description

Linked list and stack utility functions are used to maintain linked lists and stacks used within the ASN.1 run-time library functions.

Function Documentation

EXTERN DListNode* dListAppend (OOCTXT * *pctxt*, DList * *pList*, void * *pData*)

This function appends an item to the linked list structure.

The data item is passed into the function as a void pointer that can point to any object of any type. The memAlloc function is used to allocated the memory for the list node structure; therefore, all internal list memory will be released whenever memFree is called. The pointer to the data item itself is stored in the node structure - a copy is not made.

Parameters:

pctxt A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
pList A pointer to a linked list structure onto which the data item is to be appended. A pointer to an updated linked list structure.
pData A pointer to a data item to be appended to the list.

Returns:

A pointer to an allocated node structure used to link the given data value into the list.

EXTERN void dListFreeAll (OOCTXT * *pctxt*, DList * *pList*)

This function removes all nodes from the linked list structure and releases the memory that was allocated for storing the node structures (DListNode) and for data.

The memory for data in each node must have been previously allocated with calls to memAlloc, memAllocZ, or memRealloc functions.

Parameters:

pctxt 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.
pList Pointer to a linked list structure.

EXTERN void dListFreeNodes (OOCTXT * *pctxt*, DList * *pList*)

This function removes all nodes from the linked list and releases the memory that was allocated for storing the node structures (DListNode).

The data will not be released.

Parameters:

pctxt A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
pList A pointer to a linked list structure onto which the data item is to be appended. A pointer to an updated linked list structure.

EXTERN void dListInit (DList * *pList*)

This function initializes a doubly linked list structure.

It sets the number of elements to zero and sets al internal pointer values to NULL. A doubly linked-list structure is described by the DList type defined in **ooasn1.h**. Nodes of the list are of type DListNode.

Memory for the structures is allocated using the memAlloc run-time function and is maintained within the context structure that is a required parameter to all dList functions. This memory is released when memFree is called or the Context is released. Unless otherwise noted, all data passed into the list functions is simply stored on the list by value (i.e. a deep-copy of the data is not done).

Parameters:

pList A pointer to a linked list structure to be initialized.

EXTERN void dListRemove (DList * *pList*, DListNode * *node*)

This function removes a node from the linked list structure.

The memAlloc function was used to allocate the memory for the list node structure, therefore, all internal list memory will be released whenever memFree or memFreePtr is called.

Parameters:

pList A pointer to a linked list structure onto which the data item is to be appended. A pointer to an updated linked list structure.
node A pointer to the node that is to be removed. It should already be in the linked list structure.

EXTERN SListNode* sListAppend (SList * *pList*, void * *pData*)

This function is used to append a new data member to the list.

Parameters:

pList Pointer to the list to which data has to be appended.
pData Pointer to the data to be appended.

Returns:

Returns pointer to the newly appended list node.

EXTERN SList* sListCreate ()

This function is used to create a new singly-linked list.

Parameters:

None

Returns:

Pointer to the newly created list.

EXTERN SList* sListCreateEx (OOCTXT * *pctxt*)

This function is used to create a singly-linked list.

The memory for the list is allocated using the context pointer passed to this function and also the same context will be used for any further memory required by the list.

Parameters:

pctxt Pointer to the OOCTXT context which will be used for list creation

Returns:

Pointer to the newly created list structure.

EXTERN ASN1BOOL sListFind (SList * *pList*, void * *pData*)

This function is used to search for a particular data in the list.

Parameters:

pList Pointer to the list in which data has to be searched.
pData Pointer to the data to be searched.

Returns:

1 if found, 0 otherwise.

EXTERN void sListFree (SList * *pList*)

This function is used to free-up all the nodes in the singly-linked list.

Parameters:

pList Pointer to the list to be freed.

Returns:

None

EXTERN void sListInit (SList * *pList*)

This function is used to initialize a singly-linked list.

Parameters:

pList Pointer to the SList structure.

Returns:

None

EXTERN void sListInitEx (OOCTXT * *pctxt*, SList * *pList*)

This function is used to initialize a singly-linked list and assigns a context to be used for the list.

Parameters:

pctxt Pointer to the context which will be used for memory allocations related to the list.
pList Pointer to the SList structure to be initialized.

Returns:

None

EXTERN void sListRemove (SList * *pList*, void * *pData*)

This function is used to remove a particular data member from the list.

Parameters:

pList Pointer to the list from which the data has to be removed.
pData Pointer to the data to be removed.

Returns:
None

Error Formatting and Print Functions

Defines

- #define **LOG_ASN1ERR**(ctxt, stat) errSetData(&(ctxt)->errInfo,stat,__FILE__,__LINE__)
- #define **LOG_ASN1ERR_AND_FREE**(pctxt, stat, lctxt) freeContext ((lctxt)), LOG_ASN1ERR(pctxt, stat)

Functions

- EXTERN int **errAddIntParm** (ASN1ErrInfo *pErrInfo, int errParm)

This function adds an integer parameter to an error information structure.

- EXTERN int **errAddStrParm** (ASN1ErrInfo *pErrInfo, const char *errprm_p)

This function adds an string parameter to an error information structure.

- EXTERN int **errAddUIntParm** (ASN1ErrInfo *pErrInfo, unsigned int errParm)

This function adds an unsigned integer parameter to an error information structure.

- EXTERN int **errCopyData** (ASN1ErrInfo *pSrcErrInfo, ASN1ErrInfo *pDestErrInfo)
- EXTERN void **errFreeParms** (ASN1ErrInfo *pErrInfo)

This function frees memory associated with the storage of parameters associated with an error message.

- EXTERN char * **errFmtMsg** (ASN1ErrInfo *pErrInfo, char *bufp)
- EXTERN char * **errGetText** (OOCTXT *pctxt)

This function gets the text of the error.

- EXTERN void **errPrint** (ASN1ErrInfo *pErrInfo)

This function prints error information to the standard output device.

- EXTERN int **errReset** (ASN1ErrInfo *pErrInfo)

This function resets the error information in the error information sturcture.

- EXTERN int **errSetData** (ASN1ErrInfo *pErrInfo, int status, const char *module, int lno)

This function sets error information in an error information structure.

Detailed Description

Error formatting and print functions allow information about the encode/decode errors to be added to a context block structure and then printed out when the error is propagated to the top level.

Function Documentation

EXTERN int errAddIntParm (ASN1ErrInfo * *pErrInfo*, int *errParm*)

This function adds an integer parameter to an error information structure.

Parameter substitution is done in much the same way as it is done in C printf statments. The base error message specification that goes along with a particular status code may have variable fields built in using '%' modifiers. These would be replaced with actual parameter data.

Parameters:

pErrInfo A pointer to a structure containing information on the error to be printed. Typically, the error info structure referred to is the one inside the OOCTXT structure. (i.e. &pctxt->errInfo).
errParm The typed error parameter.

Returns:

The status of the operation.

EXTERN int errAddStrParm (ASN1ErrInfo * *pErrInfo*, const char * *errprm_p*)

This function adds an string parameter to an error information structure.

Parameter substitution is done in much the same way as it is done in C printf statments. The base error message specification that goes along with a particular status code may have variable fields built in using '%' modifiers. These would be replaced with actual parameter data.

Parameters:

pErrInfo A pointer to a structure containing information on the error to be printed. Typically, the error info structure referred to is the one inside the OOCTXT structure. (i.e. &pctxt->errInfo).
errprm_p The typed error parameter.

Returns:

The status of the operation.

EXTERN int errAddUIntParm (ASN1ErrInfo * *pErrInfo*, unsigned int *errParm*)

This function adds an unsigned integer parameter to an error information structure.

Parameter substitution is done in much the same way as it is done in C printf statments. The base error

message specification that goes along with a particular status code may have variable fields built in using '%' modifiers. These would be replaced with actual parameter data.

Parameters:

pErrInfo A pointer to a structure containing information on the error to be printed. Typically, the error info structure referred to is the one inside the OOCTXT structure. (i.e. &pctxt->errInfo).
errParm The typed error parameter.

Returns:

The status of the operation.

EXTERN void errFreeParms (ASN1ErrInfo * *pErrInfo*)

This function frees memory associated with the storage of parameters associated with an error message.

These parameters are maintained on an internal linked list maintained within the error information structure. The list memory must be freed when error processing is complete. This function is called from within errPrint after teh error has been printed out. It is also called in teh freeContext function.

Parameters:

pErrInfo A pointer to a structure containing information on the error to be printed. Typically, the error info structure referred to is the one inside the OOCTXT structure. (i.e. &pctxt->errInfo).

EXTERN char* errGetText (OOCTXT * *pctxt*)

This function gets the text of the error.

Parameters:

pctxt A pointer to a context structure.

EXTERN void errPrint (ASN1ErrInfo * *pErrInfo*)

This function prints error information to the standard output device.

The error information is stored in a structure of type ASN1ErrInfo. A structure of the this type is part f the OOCTXT structure. This is where error information is stored within the ASN1C generated and low-level encode/decode functions.

Parameters:

pErrInfo A pointer to a structure containing information on the error to be printed. Typically, the error info structure referred to is the one inside the OOCTXT structure. (i.e. &pctxt->errInfo).

EXTERN int errReset (ASN1ErrInfo * *pErrInfo*)

This function resets the error information in the error information sturcture.

Parameters:

pErrInfo A pointer to a structure containing information on the error to be printed. Typically, the error info structure referred to is the one inside the OOCTXT structure. (i.e. &pctxt->errInfo).

EXTERN int errSetData (ASN1ErrInfo * *pErrInfo*, int *status*, const char * *module*, int *lno*)

This function sets error information in an error information structure.

The information set includes status code, module name, and line number. Location information (i.e. module name and line number) is pushed onto a stack within the error information structure to provide a complete stack trace when the information is printed out.

Parameters:

pErrInfo A pointer to a structure containing information on the error to be printed. Typically, the error info structure referred to is the one inside the OOCTXT structure. (i.e. &pctxt->errInfo).
status The error status code. This is one of the negative error status codes.
module The name of the module (C or C++ source file) in which the module occurred. This is typically obtained by using the _FILE_ macro.
lno The line number at which the error occurred. This is typically obtained by using the _LINE_ macro.

Returns:

The status value passed to the operation in the third argument. This makes it possible to set the error information and return the status value in one line of code.

Call Management

Functions

- EXTERN **ooCallData * ooCreateCall** (char *type, char *callToken)

This function is used to create a new call entry.

- EXTERN int **ooAddCallToList** (**ooEndPoint** *h323ep, **ooCallData** *call)

This function is used to add a call to the list of existing calls.

- EXTERN **ooCallData * ooFindCallByToken** (char *callToken)

This function is used to find a call by using the unique token for the call.

- EXTERN int **ooEndCall** (**ooCallData** *call)

This function is used to clear a call.

- EXTERN int **ooRemoveCallFromList** (**ooEndPoint** *h323ep, **ooCallData** *call)

This function is used to remove a call from the list of existing calls.

- EXTERN int **ooCleanCall** (**ooCallData** *call)

This function is used to clean a call.

- EXTERN **ooLogicalChannel** * **ooAddNewLogicalChannel** (**ooCallData** *call, int channelNo, int sessionID, char *type, char *dir, **ooH323EpCapability** *epCap)

This function is used to add a new logical channel entry into the list of currently active logical channels.

- EXTERN **ooLogicalChannel** * **ooFindLogicalChannelByLogicalChannelNo** (**ooCallData** *call, int channelNo)

This function is used to find a logical channel by logical channel number.

- EXTERN int **ooOnLogicalChannelEstablished** (**ooCallData** *call, **ooLogicalChannel** *pChannel)

This function is called when a new logical channel is established.

- EXTERN ASN1BOOL **ooIsSessionEstablished** (**ooCallData** *call, int sessionID, char *dir)

This fuction is used to check whether a specified session in specified direction is active for the call.

- EXTERN **ooLogicalChannel** * **ooGetLogicalChannel** (**ooCallData** *call, int sessionID)

This function is used to retrieve a logical channel with particular sessionID.

- EXTERN int **ooRemoveLogicalChannel** (**ooCallData** *call, int ChannelNo)

This function is used to remove a logical channel from the list of logical channels.

- EXTERN int **ooClearLogicalChannel** (**ooCallData** *call, int channelNo)

This function is used to cleanup a logical channel.

- EXTERN int **ooClearAllLogicalChannels** (**ooCallData** *call)

This function is used to cleanup all the logical channels associated with the call.

- EXTERN int **ooAddMediaInfo** (**ooCallData** *call, ooMediaInfo mediaInfo)

This function can be used by an application to specify media endpoint information for different types of media.

Function Documentation

EXTERN int ooAddCallToList (ooEndPoint * *h323ep*, ooCallData * *call*)

This function is used to add a call to the list of existing calls.

Parameters:

h323ep Pointer to the H323 Endpoint structure.
call Pointer to the call to be added.

Returns:

OO_OK, on success. OO_FAILED, on failure

EXTERN int ooAddMediaInfo (ooCallData * *call*, ooMediaInfo *mediaInfo*)

This function can be used by an application to specify media endpoint information for different types of media.

The stack by default uses local IP and port for media. An application can provide mediainfo if it wants to override default.

Parameters:

call Handle to the call
mediaInfo mediainfo structure which defines the media endpoint to be used.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN ooLogicalChannel* ooAddNewLogicalChannel (ooCallData * *call*, int *channelNo*, int *sessionID*, char * *type*, char * *dir*, ooH323EpCapability * *epCap*)

This function is used to add a new logical channel entry into the list of currently active logical channels.

Parameters:

call Pointer to the call for which new logical channel entry has to be created.
channelNo Channel number for the new channel entry.
sessionID Session identifier for the new channel.
type Type of the channel(audio/video/data)
dir Direction of the channel(transmit/receive)
epCap Capability to be used for the new channel.

Returns:

Pointer to logical channel, on success. NULL, on failure

EXTERN int ooCleanCall (ooCallData * *call*)

This function is used to clean a call.

It closes all associated sockets, removes call from list and frees up associated memory.

Parameters:

call Pointer to the call to be cleared.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooClearAllLogicalChannels (ooCallData * *call*)

This function is used to cleanup all the logical channels associated with the call.

Parameters:

call Handle to the call which owns the channels.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooClearLogicalChannel (ooCallData * *call*, int *channelNo*)

This function is used to cleanup a logical channel.

It first stops media, if it is still active and then removes the channel from the list, freeing up all the associated memory.

Parameters:

call Handle to the call which owns the logical channel.
channelNo Channel number identifying the channel.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN ooCallData* ooCreateCall (char * *type*, char * *callToken*)

This function is used to create a new call entry.

Parameters:

type Type of the call (incoming/outgoing)
callToken Call Token, an uniques identifier for the call

Returns:

Pointer to a newly created call

EXTERN int ooEndCall (ooCallData * *call*)

This function is used to clear a call.

Based on what stage of clearance the call is it takes appropriate action.

Parameters:

call Handle to the call which has to be cleared.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN ooCallData* ooFindCallByToken (char * *callToken*)

This function is used to find a call by using the unique token for the call.

Parameters:

callToken The unique token for the call.

Returns:

Pointer to the call if found, NULL otherwise.

EXTERN ooLogicalChannel* ooFindLogicalChannelByLogicalChannelNo (ooCallData * *call*, int *channelNo*)

This function is used to find a logical channel by logical channel number.

Parameters:

call Pointer to the call for which logical channel is required.
channelNo Forward Logical Channel number for the logical channel

Returns:

Pointer to the logical channel if found, NULL otherwise.

EXTERN ooLogicalChannel* ooGetLogicalChannel (ooCallData * *call*, int *sessionID*)

This function is used to retrieve a logical channel with particular sessionID.

Note that there can be two entries of logical channel, one in each direction. This function will return the first channel which has the same session ID.

Parameters:

call Handle to the call which owns the channels to be searched.
sessionID Session id of the session which is to be searched for.

Returns:

Returns a pointer to the logical channel if found, NULL otherwise.

EXTERN ASN1BOOL ooIsSessionEstablished (ooCallData * *call*, int *sessionID*, char * *dir*)

This fuction is used to check whether a specified session in specified direction is active for the call.

Parameters:

call Handle to call for which session has to be queried.
sessionID Session id to identify the type of session(1 for audio, 2 for voice and 3 for data)
dir Direction of the session(transmit/receive)

Returns:

1, if session active. 0, otherwise.

EXTERN int ooOnLogicalChannelEstablished (ooCallData * *call*, ooLogicalChannel * *pChannel*)

This function is called when a new logical channel is established.

It is particularly useful in case of faststart. When the remote endpoint selects one of the proposed alternatives, other channels for the same session type need to be closed. This function is used for that.

Parameters:

call Handle to the call which owns the logical channel.
pChannel Handle to the newly established logical channel.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooRemoveCallFromList (ooEndPoint * *h323ep*, ooCallData * *call*)

This function is used to remove a call from the list of existing calls.

Parameters:

h323ep Pointer to the H323 Endpoint.
call Pointer to the call to be removed.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooRemoveLogicalChannel (ooCallData * *call*, int *ChannelNo*)

This function is used to remove a logical channel from the list of logical channels.

Parameters:

call Pointer to the call from which logical channel has to be removed.
ChannelNo Forward logical channel number of the channel to be removed.

Capability Management

Functions

- EXTERN int **ooAddAudioCapability** (H245AudioCapability audioCap, int dir, **cb_StartReceiveChannel** startReceiveChannel, **cb_StartTransmitChannel** startTransmitChannel, **cb_StopReceiveChannel** stopReceiveChannel, **cb_StopTransmitChannel** stopTransmitChannel)

Function to add audio capabilities to the endpoint.

- EXTERN H245AudioCapability * **ooCreateDupAudioCap** (**ooCallData** *call, H245AudioCapability *audioCap, OOCTXT *pctxt)

This function is used to create a duplicate audio capability from the one supplied as parameter.

- EXTERN H245GSMAudioCapability * **ooCreateDupGSMAudioCap** (**ooCallData** *call, H245GSMAudioCapability *gsmCap, OOCTXT *pctxt)

This function is used to create a duplicate GSM audio capability.

- EXTERN **ooH323EpCapability** * **ooIsAudioCapSupported** (**ooCallData** *call, H245AudioCapability *audioCap, int dir)

This function is used to check whether local endpoint supports a particular type of audio capability.

- EXTERN int **ooCompareAudioCaps** (**ooCallData** *call, H245AudioCapability *cap1, H245AudioCapability *cap2)

This function is used to compare two audio capabilities for a match.

- EXTERN int **ooCompareG711Caps** (H245AudioCapability *cap1, H245AudioCapability *cap2)

This function is used to compare two G711 audio capabilities for a match.

- EXTERN int **ooCompareGSMCaps** (H245AudioCapability *cap1, H245AudioCapability *cap2)

This function is used to compare two GSM audio capabilities for a match.

- EXTERN **ooH323EpCapability** * **ooIsDataTypeSupported** (**ooCallData** *call, H245DataType *data, int dir)

Checks whether a particular datatype is supported by the end point.

- EXTERN **ooH323EpCapability * ooIsAudioDataTypeSupported** (**ooCallData** *call, H245AudioCapability *audioData, int dir)

Checks whether a particular audio datatype is supported by the end point.

- EXTERN **ooH323EpCapability * ooIsNonStandardDataTypeSupported** (**ooCallData** *call, H245NonStandardParameter *nonStandard, int dir)
- EXTERN **ooH323EpCapability * ooIsVideoDataTypeSupported** (**ooCallData** *call, H245VideoCapability *videoData, int dir)
- EXTERN **ooH323EpCapability * ooIsApplicationDataTypeSupported** (**ooCallData** *call, H245DataApplicationCapability *data, int dir)
- EXTERN **ooH323EpCapability * ooIsEncryptedDataTypeSupported** (**ooCallData** *call, H245EncryptionMode *encryptionData, int dir)
- EXTERN **ooH323EpCapability * ooIsH235ControlDataTypeSupported** (**ooCallData** *call, H245NonStandardParameter *h235Control, int dir)
- EXTERN **ooH323EpCapability * ooIsH235MediaDataTypeSupported** (**ooCallData** *call, H245H235Media *h235Media, int dir)
- EXTERN **ooH323EpCapability * ooIsMultiplexedStreamDataTypeSupported** (**ooCallData** *call, H245MultiplexedStreamParameter *multiplexedStream, int dir)

Function Documentation

EXTERN int ooAddAudioCapability (H245AudioCapability *audioCap*, int *dir*, cb_StartReceiveChannel *startReceiveChannel*, cb_StartTransmitChannel *startTransmitChannel*, cb_StopReceiveChannel *stopReceiveChannel*, cb_StopTransmitChannel *stopTransmitChannel*)

Function to add audio capabilities to the endpoint.

'dir' indicates whether we have a transmit capability or a receive capability or both. Last four parameters are the callback functions for channel control

Parameters:

audioCap Audio Capability to be added.
dir Direction - Indicates whether endpoint has receive capability, or transmit capability or both.ex T_H245Capability_receiveAudioCapability
startReceiveChannel Callback function to call receive channel.
startTransmitChannel Callback function to start transmit channel.
stopReceiveChannel Callback function to stop receive channel.
stopTransmitChannel Callback function to stop transmit channel.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooCompareAudioCaps (ooCallData * *call*, H245AudioCapability * *cap1*, H245AudioCapability * *cap2*)

This function is used to compare two audio capabilities for a match.

Parameters:

call Handle to the call for which comparison is being carried out.
cap1 First capability to be compared.
cap2 Second capability to be compared.

Returns:

1, on match. 0. otherwise.

EXTERN int ooCompareG711Caps (H245AudioCapability * *cap1*, H245AudioCapability * *cap2*)

This function is used to compare two G711 audio capabilities for a match.

Parameters:

cap1 First G711 capability to be compared.
cap2 Second G711 capability to be compared.

Returns:

1, on match. 0. otherwise.

EXTERN int ooCompareGSMCaps (H245AudioCapability * *cap1*, H245AudioCapability * *cap2*)

This function is used to compare two GSM audio capabilities for a match.

Parameters:

cap1 First GSM capability to be compared.
cap2 Second GSM capability to be compared.

Returns:

1, on match. 0. otherwise.

EXTERN H245AudioCapability* ooCreateDupAudioCap (ooCallData * *call*, H245AudioCapability * *audioCap*, OOCTXT * *pctxt*)

This function is used to create a duplicate audio capability from the one supplied as parameter.

Parameters:

call Handle to the call for which duplicate capability has to be created.
audioCap Source audio capability.
pctxt Pointer to an OOCTXT which will be used to allocate memory for the new capability.

Returns:

Pointer to the newly created audio capability or NULL in case of failure

EXTERN H245GSMAudioCapability* ooCreateDupGSMAudioCap (ooCallData * *call*, H245GSMAudioCapability * *gsmCap*, OOCTXT * *pctxt*)

This function is used to create a duplicate GSM audio capability.

Parameters:

call Handle to the call for which duplicate has to be created.
gsmCap Pointer to the source capability.
pctxt Pointer to the OOCTXT context which will be used for memory allocation.

Returns:

Pointer to the newly created GSM capability or NULL in case of failure.

EXTERN ooH323EpCapability* ooIsAudioCapSupported (ooCallData * *call*, H245AudioCapability * *audioCap*, int *dir*)

This function is used to check whether local endpoint supports a particular type of audio capability.

Parameters:

call Handle to the call.
audioCap Pointer to the audio capability
dir Direction in which support is required. ex. T_H245Capability_receiveAudioCapability

Returns:

Pointer to the matching capability, if found. Null, otherwise.

EXTERN ooH323EpCapability* ooIsAudioDataTypeSupported (ooCallData * *call*, H245AudioCapability * *audioData*, int *dir*)

Checks whether a particular audio datatype is supported by the end point.

Parameters:

call Handle to the call
audioData Pointer to the audio data type
dir Direction in which support is required.

Returns:

Pointer to the matching capability, if supported, else NULL.

EXTERN ooH323EpCapability* ooIsDataTypeSupported (ooCallData * *call*, H245DataType * *data*, int *dir*)

Checks whether a particular datatype is supported by the end point.

Parameters:

call Handle to the call
data Pointer to the data type
dir Direction in which support is required.

Returns:

Pointer to the matching capability, if supported, else NULL.

Channel Management

Functions

- EXTERN int **ooCreateH323Listener** ()

This function is used to create a listener for incoming calls.

- EXTERN int **ooCreateH245Listener** (**ooCallData** *call)

This function is used to create a listener for incoming H.245 connections.

- EXTERN int **ooAcceptH225Connection** ()

This function is used to accept incoming H.225 connections.

- EXTERN int **ooAcceptH245Connection** (**ooCallData** *call)

This function is used to accept an incoming H.245 connection.

- EXTERN int **ooCreateH225Connection** (**ooCallData** *call)

This function is used to create an H.225 connection to the remote end point.

- EXTERN int **ooCreateH245Connection** (**ooCallData** *call)

This function is used to setup an H.245 connection with the remote endpoint for control negotiations.

- EXTERN int **ooCloseH225Connection** (**ooCallData** *call)

This function is used to close an H.225 connection.

- EXTERN int **ooCloseH245Connection** (**ooCallData** *call)

This function is used to close an H.245 connection for a call.

- EXTERN int **ooMonitorChannels** ()

This function is used to start monitoring channels for the calls.

- EXTERN int **ooStopMonitorCalls** ()

This function is called to stop the monitor channels thread.

- EXTERN int **ooH2250Receive** (**ooCallData** *call)

This function is used to receive an H.2250 message received on a calls H.225 channel.

- EXTERN int **ooH245Receive** (**ooCallData** *call)

This function is used to receive an H.245 message received on a calls H.245 channel.

- EXTERN int **ooSendH225Msg** (**ooCallData** *call, **Q931Message** *msg)

This function is used to enqueue an H.225 message into an outgoing queue for the call.

- EXTERN int **ooSendH245Msg** (**ooCallData** *call, **H245Message** *msg)

This function is used to enqueue an H.245 message into an outgoing queue for the call.

- EXTERN int **ooSendMsg** (**ooCallData** *call, int type)

This function is used to Send a message on the channel, when channel is available for write.

- EXTERN int **ooOnSendMsg** (**ooCallData** *call, int msgType)

This function is called after a message is sent on the call's channel.

Function Documentation

EXTERN int ooAcceptH225Connection ()

This function is used to accept incoming H.225 connections.

Parameters:

None

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooAcceptH245Connection (ooCallData * *call*)

This function is used to accept an incoming H.245 connection.

Parameters:

call Pointer to a call for which H.245 connection request has arrived.

Returns:

OO_OK, on succes. OO_FAILED, on failure.

EXTERN int ooCloseH225Connection (ooCallData * *call*)

This function is used to close an H.225 connection.

Parameters:

call Pointer to the call for which H.225 connection has to be closed.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooCloseH245Connection (ooCallData * *call*)

This function is used to close an H.245 connection for a call.

Parameters:

call Pointer to call for which H.245 connection has to be closed.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooCreateH225Connection (ooCallData * *call*)

This function is used to create an H.225 connection to the remote end point.

Parameters:

call Pointer to the call for which H.225 connection has to be setup.

Returns:

OO_OK, on succes. OO_FAILED, on failure.

EXTERN int ooCreateH245Connection (ooCallData * *call*)

This function is used to setup an H.245 connection with the remote endpoint for control negotiations.

Parameters:

call Pointer to call for which H.245 connection has to be setup.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooCreateH245Listener (ooCallData * *call*)

This function is used to create a listener for incoming H.245 connections.

Parameters:

call Pointer to call for which H.245 listener has to be created

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooCreateH323Listener ()

This function is used to create a listener for incoming calls.

Parameters:

None

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooH2250Receive (ooCallData * *call*)

This function is used to receive an H.2250 message received on a calls H.225 channel.

It receives the message, decodes it and calls 'ooHandleH2250Message' to process the message.

Parameters:

call Pointer to the call for which the message has to be received.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooH245Receive (ooCallData * *call*)

This function is used to receive an H.245 message received on a calls H.245 channel.

It receives the message, decodes it and calls 'ooHandleH245Message' to process it.

Parameters:

call Pointer to the call for which the message has to be received.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooMonitorChannels ()

This function is used to start monitoring channels for the calls.

It has an infinite loop which uses select to monitor various channels.

Parameters:

None

EXTERN int ooOnSendMsg (ooCallData * *call*, int *msgType*)

This function is called after a message is sent on the call's channel.

It can be used to some followup action after message has been sent.

Parameters:

call Pointer to call for which message has been sent.
msgType Type of message

Returns:

OO_OK, on success. OO_FAILED, on failure

EXTERN int ooSendH225Msg (ooCallData * *call*, Q931Message * *msg*)

This function is used to enqueue an H.225 message into an outgoing queue for the call.

Parameters:

call Pointer to call for which message has to be enqueued.
msg Pointer to the H.225 message to be sent.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooSendH245Msg (ooCallData * *call*, H245Message * *msg*)

This function is used to enqueue an H.245 message into an outgoing queue for the call.

Parameters:

call Pointer to call for which message has to be enqueued.
msg Pointer to the H.245 message to be sent.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooSendMsg (ooCallData * *call*, int *type*)

This function is used to Send a message on the channel, when channel is available for write.

Parameters:

call Pointer to call for which message has to be sent.
type Type of the message.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooStopMonitorCalls ()

This function is called to stop the monitor channels thread.

It cleans up all the active calls, before stopping monitor thread.

Parameters:

None

Returns:

OO_OK, on success. OO_FAILED, on failure

H245 Message Handling

Functions

- EXTERN int **ooCreateH245Message** (**H245Message** **msg, int type)

Creates an outgoing H245 message of the type specified by the type argument for the Application context.

- EXTERN int **ooFreeH245Message** (**H245Message** *pmsg)

Frees up the memory used by the H245 message.

- EXTERN int **ooGetOutgoingH245Msgbuf** (**ooCallData** *call, ASN1OCTET *msgbuf, int *len, int *msgType)

This function is used to retrieve an H.245 message enqueued in the outgoing queue.

- EXTERN int **ooSendTermCapMsg** (**ooCallData** *call)

This function is used to send out a treminal capability set message.

- EXTERN ASN1UINT **ooGenerateStatusDeterminationNumber** ()

This function is used to generate a random status determination number for MSD procedure.

- EXTERN int **ooHandleMasterSlave** (**ooCallData** *call, void *pmsg, int msgType)

This fuction is used to handle received MasterSlaveDetermination procedure messages.

- EXTERN int **ooSendMasterSlaveDetermination** (**ooCallData** *call)

This function is used to send MSD message.

- EXTERN int **ooSendMasterSlaveDeterminationAck** (**ooCallData** *call, char *status)

This function is used to send a MasterSlaveDeterminationAck message.

- EXTERN int **ooHandleOpenLogicalChannel** (**ooCallData** *call, H245OpenLogicalChannel *olc)

This function is used to handle received OpenLogicalChannel message.

- EXTERN int **ooHandleOpenLogicalAudioChannel** (**ooCallData** *call, H245OpenLogicalChannel *olc)

This function is used to handle a received OpenLogicalChannel message which is trying to open a audio channel.

- EXTERN int **ooOnReceivedOpenLogicalChannelAck** (**ooCallData** *call, H245OpenLogicalChannelAck *olcAck)

This function is used to handle a received OpenLogicalChannelAck message.

- EXTERN int **ooSendEndSessionCommand** (**ooCallData** *call)

This message is used to send an EndSession command.

- EXTERN int **ooHandleH245Command** (**ooCallData** *call, H245CommandMessage *command)

This function is used to handle a received H245Command message.

- EXTERN int **ooOnReceivedTerminalCapabilitySetAck** (**ooCallData** *call)

This function is called on receiving a TreminalCapabilitySetAck message.

- EXTERN int **ooCloseAllLogicalChannels** (**ooCallData** *call)

This function is called to close all the open logical channels.

- EXTERN int **ooSendCloseLogicalChannel** (**ooCallData** *call, **ooLogicalChannel** *logicalChan)

This function is used to send out a CloseLogicalChannel message for a particular logical channel.

- EXTERN int **ooOnReceivedCloseLogicalChannel** (**ooCallData** *call, H245CloseLogicalChannel *clc)

This function is used to process a received closeLogicalChannel request.

- EXTERN int **ooOnReceivedCloseChannelAck** (**ooCallData** *call, H245CloseLogicalChannelAck *clcAck)

This function is used to process a received CloseLogicalChannelAck message.

- EXTERN int **ooHandleH245Message** (**ooCallData** *call, **H245Message** *pmsg)

This function is used to handle received H245 message.

- EXTERN int **ooOnReceivedTerminalCapabilitySet** (**ooCallData** *call, **H245Message** *pmsg)

This function is used to process received TCS message.

- EXTERN int **ooH245AcknowledgeTerminalCapabilitySet** (**ooCallData** *call)

This function is used to send a TCSAck message to remote endpoint.

- EXTERN int **ooOpenLogicalChannels** (**ooCallData** *call)

This function is used to start OpenLogicalChannel procedure for all the channels to be opened for the call.

- EXTERN int **ooOpenLogicalAudioChannel** (**ooCallData** *call)

This function is used to send OpenLogicalChannel message for audio channel.

- EXTERN int **ooOpenG711ULaw64KChannel** (**ooCallData** *call, **ooH323EpCapability** *epCap)

This function is used to build a OpenLogicalChannel message using G711ULaw64K capability.

- EXTERN int **ooSendRequestCloseLogicalChannel** (**ooCallData** *call, **ooLogicalChannel** *logicalChan)

This function is used to request a remote end point to close a logical channel.

- EXTERN int **ooOnReceivedRequestChannelClose** (**ooCallData** *call, H245RequestChannelClose *rclc)

This function handles the received RequestChannelClose message, verifies that the requested channel is forward channel.

- EXTERN int **ooBuildOpenLogicalChannelAudio** (**ooCallData** *call, H245OpenLogicalChannel *olc, **ooH323EpCapability** *epCap, OOCTXT *pctxt)

Builds an OLC with an audio capability passed as parameter.

- EXTERN int **ooSendAsTunneledMessage** (**ooCallData** *call, ASN1OCTET *msgbuf, int len, int

msgType)
This function sends an encoded H.245 message buffer as a tunneled H.245 message.

Function Documentation

EXTERN int ooBuildOpenLogicalChannelAudio (ooCallData * *call*, H245OpenLogicalChannel * *olc*, ooH323EpCapability * *epCap*, OOCTXT * *pctxt*)

Builds an OLC with an audio capability passed as parameter.

Parameters:

call Handle to call for which OLC has to be built.
olc Pointer to an OLC structure which will be populated.
epCap Pointer to the capability which will be used to build OLC.
pctxt Pointer to an OOCTXT structure which will be used to allocate additional memory for OLC.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooCloseAllLogicalChannels (ooCallData * *call*)

This function is called to close all the open logical channels.

It sends CloseLogicalChannel message for all the forward channels and sends RequestCloseLogicalChannel message for all the reverse channels.

Parameters:

call Pointer to call for which logical channels have to be closed.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooCreateH245Message (H245Message ** *msg*, int *type*)

Creates an outgoing H245 message of the type specified by the type argument for the Application context.

Parameters:

msg A pointer to pointer to message which will be assigned to allocated memory.
type Type of the message to be created. (Request/Response/Command/Indication)

Returns:

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

EXTERN int ooFreeH245Message (H245Message * *pmsg*)

Frees up the memory used by the H245 message.

Parameters:

pmsg Pointer to an H245 message structure.

Returns:

OO_OK, on success. OO_FAILED, on failure

EXTERN ASN1UINT ooGenerateStatusDeterminationNumber ()

This function is used to generate a random status determination number for MSD procedure.

Parameters:

None

Returns:

Generated status determination number.

EXTERN int ooGetOutgoingH245Msgbuf (ooCallData * *call*, ASN1OCTET * *msgbuf*, int * *len*, int * *msgType*)

This function is used to retrieve an H.245 message enqueued in the outgoing queue.

Parameters:

call Pointer to the call for which message has to be retrieved.
msgbuf Pointer to a buffer in which the message will be returned.
len Pointer to an int variable which will contain length of the message data after returning.
msgType Pointer to an int variable, which will contain message type on return from the function.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooH245AcknowledgeTerminalCapabilitySet (ooCallData * *call*)

This function is used to send a TCSAck message to remote endpoint.

Parameters:

call Pointer to call on which TCSAck has to be sent.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooHandleH245Command (ooCallData * *call*, H245CommandMessage * *command*)

This function is used to handle a received H245Command message.

Parameters:

call Pointer to call for which an H245Command is received.
command Pointer to a command message.

Returns:

OO_OK, on success. OO_FAILED, on failure

EXTERN int ooHandleH245Message (ooCallData * *call*, H245Message * *pmsg*)

This function is used to handle received H245 message.

Based on the type of message received, it calls helper functions to process those messages.

Parameters:

call Pointer to call for which a message is received.
pmsg Pointer to the received H245 message.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooHandleMasterSlave (ooCallData * *call*, void * *pmsg*, int *msgType*)

This fuction is used to handle received MasterSlaveDetermination procedure messages.

Parameters:

call Pointer to the call for which a message is received.
pmsg Pointer to MSD message
msgType Message type indicating whether received message is MSD, MSDAck, MSDReject etc...

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooHandleOpenLogicalAudioChannel (ooCallData * *call*, H245OpenLogicalChannel * *olc*)

This function is used to handle a received OpenLogicalChannel message which is trying to open a audio channel.

Parameters:

call Pointer to cll for which OLC was received.
olc The received OpenLogicalChannel message.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooHandleOpenLogicalChannel (ooCallData * *call*, H245OpenLogicalChannel * *olc*)

This function is used to handle received OpenLogicalChannel message.

Parameters:

call Pointer to call for which OpenLogicalChannel message is received.
olc Pointer to the received OpenLogicalChannel message.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooOnReceivedCloseChannelAck (ooCallData * *call*, H245CloseLogicalChannelAck * *clcAck*)

This function is used to process a received CloseLogicalChannelAck message.

It closes the channel and removes it from the list of active logical channels.

Parameters:

call Pointer to call for which CLCAck message is received.
clcAck Pointer to the received CloseLogicalChannelAck message.

Returns:

OO_OK, on success. OO_FAILED, on failure

EXTERN int ooOnReceivedCloseLogicalChannel (ooCallData * *call*, H245CloseLogicalChannel * *clc*)

This function is used to process a received closeLogicalChannel request.

It closes the logical channel and removes the logical channel entry from the list. It also, sends

closeLogicalChannelAck message to the remote endpoint.

Parameters:

call Pointer to call for which CloseLogicalChannel message is received.
clc Pointer to received CloseLogicalChannel message.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooOnReceivedOpenLogicalChannelAck (ooCallData * *call*, H245OpenLogicalChannelAck * *olcAck*)

This function is used to handle a received OpenLogicalChannelAck message.

Parameters:

call Pointer to call for which OLCAck is received
olcAck Pointer to received olcAck message.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooOnReceivedRequestChannelClose (ooCallData * *call*, H245RequestChannelClose * *rclc*)

This function handles the received RequestChannelClose message, verifies that the requested channel is forward channel.

It sends an acknowledgement for the message followed by CloseLogicalChannel message.

Parameters:

call Pointer to the call for which RequestChannelClose is received.
rclc Pointer to the received message.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooOnReceivedTerminalCapabilitySet (ooCallData * *call*, H245Message * *pmsg*)

This function is used to process received TCS message.

It builds TCSAck message and queues it into the calls outgoing queue. Also, starts Logical channel opening procedure if TCS and MSD procedures have finished.

Parameters:

call Pointer to call for which TCS is received.
pmsg Pointer to the received message.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooOnReceivedTerminalCapabilitySetAck (ooCallData * *call*)

This function is called on receiving a TreminalCapabilitySetAck message.

If the MasterSlaveDetermination process is also over, this function initiates the process of opening logical channels.

Parameters:

call Pointer to call for which TCSAck is received.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooOpenG711ULaw64KChannel (ooCallData * *call*, ooH323EpCapability * *epCap*)

This function is used to build a OpenLogicalChannel message using G711ULaw64K capability.

Parameters:

call Pointer to call for which OpenLogicalChannel message have to be built.
epCap Pointer to G711ULaw capability

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooOpenLogicalAudioChannel (ooCallData * *call*)

This function is used to send OpenLogicalChannel message for audio channel.

It uses the first capability match in the local and remote audio capabilities for the audio channel and calls corresponding helper function.

Parameters:

call Pointer to call for which audio channel ahs to be opened.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooOpenLogicalChannels (ooCallData * *call*)

This function is used to start OpenLogicalChannel procedure for all the channels to be opened for the call.

Parameters:

call Pointer to call for which logical channels have to be opened.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooSendAsTunneledMessage (ooCallData * *call*, ASN1OCTET * *msgbuf*, int *len*, int *msgType*)

This function sends an encoded H.245 message buffer as a tunneled H.245 message.

If there is an outgoing H.225 message, it is used to tunnel the H.245 message and if there is no H.225 message, then a new Facility message is created for tunneling purpose.

Parameters:

call Pointer to the call for which H.245 message has to be tunneled.
msgbuf Pointer to the encoded H.245 message to be tunneled.
len Length of the encoded H.245 message buffer.
msgType Type of the H245 message

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooSendCloseLogicalChannel (ooCallData * *call*, ooLogicalChannel * *logicalChan*)

This function is used to send out a CloseLogicalChannel message for a particular logical channel.

Parameters:

call Pointer to a call, to which logical channel to be closed belongs.
logicalChan Pointer to the logical channel to be closed.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooSendEndSessionCommand (ooCallData * *call*)

This message is used to send an EndSession command.

It builds a EndSession command message and queues it into the calls outgoing queue.

Parameters:

call Pointer to call for which EndSession command has to be sent.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooSendMasterSlaveDetermination (ooCallData * *call*)

This function is used to send MSD message.

Parameters:

call Pointer to call for which MasterSlaveDetermination has to be sent.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooSendMasterSlaveDeterminationAck (ooCallData * *call*, char * *status*)

This function is used to send a MasterSlaveDeterminationAck message.

Parameters:

call Pointer to call for which MasterSlaveDeterminationAck has to be sent.
status Result of the determination process(Master/Slave as it applies to remote endpoint)

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooSendRequestCloseLogicalChannel (ooCallData * *call*, ooLogicalChannel * *logicalChan*)

This function is used to request a remote end point to close a logical channel.

Parameters:

call Pointer to call for which the logical channel has to be closed.
logicalChan Pointer to the logical channel structure which needs to be closed.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooSendTermCapMsg (ooCallData * *call*)

This function is used to send out a treminal capability set message.

Parameters:

call Pointer to a call, for which TerminalCapabilitySet message has to be sent.

Returns:

OO_OK, on success. OO_FAILED, on failure.

H323 Endpoint management functions

Functions

- EXTERN int **ooInitializeH323Ep** (const char *tracefile, int h245Tunneling, int fastStart, int termType, int t35CountryCode, int t35Extension, int manufacturer, char *productID, char *versionID, int callType, int listenport, char *callerid, char *callername, int callMode)

This function is the first function to be invoked before using stack.

- EXTERN int **ooH323EpRegisterCallbacks** (cb_OnIncomingCall onIncomingCall, cb_OnOutgoingCall onOutgoingCall, cb_OnCallEstablished onCallEstablished, cb_OnCallCleared onCallCleared, cb_OnStartLogicalChannel onStartLogicalChannel)

This function is used to register the H323 Endpoint callback functions.

- EXTERN int **ooDestroyH323Ep** ()

This function is the last function to be invoked after done using the stack.

Function Documentation

EXTERN int ooDestroyH323Ep ()

This function is the last function to be invoked after done using the stack.

It closes the H323 Endpoint for an application, releasing all the associated memory.

Parameters:

None

Returns:

OO_OK on success OO_FAILED on failure

EXTERN int ooH323EpRegisterCallbacks (cb_OnIncomingCall *onIncomingCall*,

cb_OnOutgoingCall *onOutgoingCall*, cb_OnCallEstablished *onCallEstablished*, cb_OnCallCleared *onCallCleared*, cb_OnStartLogicalChannel *onStartLogicalChannel*)

This function is used to register the H323 Endpoint callback functions.

Parameters:

onIncomingCall Callback function to be called when a new incoming call is detected.
onOutgoingCall Callback function to be called when an outgoing call is placed on behalf of the application.
onCallEstablished Callback function to be called when a call is established with the remote end point.
onCallCleared Callback function to be called when a call is cleared.
onStartLogicalChannel Callback function to be called when a logical channel is started.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooInitializeH323Ep (const char * *tracefile*, int *h245Tunneling*, int *fastStart*, int *termType*, int *t35CountryCode*, int *t35Extension*, int *manufacturer*, char * *productID*, char * *versionID*, int *callType*, int *listenport*, char * *callerid*, char * *callername*, int *callMode*)

This function is the first function to be invoked before using stack.

It initializes the H323 Endpoint.

Parameters:

tracefile Absolute path to the trace file to be used for storing traces
h245Tunneling Indicates whether h245Tunneling enabled(1)/disabled(0)
fastStart Indicates whether fast start is enabled(1)/disabled(0)
termType Terminal type of the endpoint.
t35CountryCode Country code to be used
t35Extension t35Extension value
manufacturer manufacturer code
productID Product ID to be used
versionID Version Id of the software
callType Type of the call ex. T_H225CallType_pointToPoint
listenport Port on which to listen for incoming calls
callerid ID to be used for outgoing calls.
callername Caller name to be used for outgoing calls
callMode Type of calls to be made(audio/video/fax). (OO_CALLMODE_AUDIO, OO_CALLMODE_VIDEO)

Returns:

OO_OK, on success. OO_FAILED, on failure

Q931/H.2250 Message Handling

Data Structures

- struct **Q931InformationElement**

Defines

- #define **Q931_E_TOOSHORT** (-1001)
- #define **Q931_E_INVCALLREF** (-1002)
- #define **Q931_E_INVLENGTH** (-1003)

Typedefs

- typedef Q931InformationElement **Q931InformationElement**

Enumerations

- enum **Q931MsgTypes** { **Q931NationalEscapeMsg** = 0x00, **Q931AlertingMsg** = 0x01, **Q931CallProceedingMsg** = 0x02, **Q931ConnectMsg** = 0x07, **Q931ConnectAckMsg** = 0x0f, **Q931ProgressMsg** = 0x03, **Q931SetupMsg** = 0x05, **Q931SetupAckMsg** = 0x0d, **Q931ResumeMsg** = 0x26, **Q931ResumeAckMsg** = 0x2e, **Q931ResumeRejectMsg** = 0x22, **Q931SuspendMsg** = 0x25, **Q931SuspendAckMsg** = 0x2d, **Q931SuspendRejectMsg** = 0x21, **Q931UserInformationMsg** = 0x20, **Q931DisconnectMsg** = 0x45, **Q931ReleaseMsg** = 0x4d, **Q931ReleaseCompleteMsg** = 0x5a, **Q931RestartMsg** = 0x46, **Q931RestartAckMsg** = 0x4e, **Q931SegmentMsg** = 0x60, **Q931CongestionCtrlMsg** = 0x79, **Q931InformationMsg** = 0x7b, **Q931NotifyMsg** = 0x6e, **Q931StatusMsg** = 0x7d, **Q931StatusEnquiryMsg** = 0x75, **Q931FacilityMsg** = 0x62 }
- enum **Q931IECodes** { **Q931BearerCapabilityIE** = 0x04, **Q931CauseIE** = 0x08, **Q931FacilityIE** = 0x1c, **Q931ProgressIndicatorIE** = 0x1e, **Q931CallStateIE** = 0x14, **Q931DisplayIE** = 0x28, **Q931SignalIE** = 0x34, **Q931CallingPartyNumberIE** = 0x6c, **Q931CalledPartyNumberIE** = 0x70, **Q931RedirectingNumberIE** = 0x74, **Q931UserUserIE** = 0x7e }
- enum **Q931InformationTransferCapability** { **Q931TransferSpeech**, **Q931TransferUnrestrictedDigital** = 8, **Q931TransferRestrictedDigital** = 9, **Q931Transfer3_1kHzAudio** = 16, **Q931TrasnferUnrestrictedDigitalWithTones** = 17, **Q931TransferVideo** = 24 }
- enum **Q931CauseValues** { **Q931NoRouteToNetwork** = 0x02, **Q931NoRouteToDestination** = 0x03, **Q931ChannelUnacceptable** = 0x06, **Q931NormalCallClearing** = 0x10, **Q931UserBusy** = 0x11, **Q931NoResponse** = 0x12, **Q931NoAnswer** = 0x13, **Q931SubscriberAbsent** = 0x14, **Q931CallRejected** = 0x15, **Q931NumberChanged** = 0x16, **Q931Redirection** = 0x17, **Q931DestinationOutOfOrder** = 0x1b, **Q931InvalidNumberFormat** = 0x1c, **Q931StatusEnquiryResponse** = 0x1e, **Q931NoCircuitChannelAvailable** = 0x22, **Q931Congestion** = 0x2a, **Q931InvalidCallReference** = 0x51, **Q931ErrorInCauseIE** = 0 }
- enum **Q931SignalInfo** { **Q931SignalDialToneOn**, **Q931SignalRingBackToneOn**, **Q931SignalInterceptToneOn**, **Q931SignalNetworkCongestionToneOn**, **Q931SignalBusyToneOn**, **Q931SignalConfirmToneOn**, **Q931SignalAnswerToneOn**, **Q931SignalCallWaitingTone**, **Q931SignalOffhookWarningTone**, **Q931SignalPreemptionToneOn**, **Q931SignalTonesOff** = 0x3f, **Q931SignalAlertingPattern0** = 0x40, **Q931SignalAlertingPattern1**, **Q931SignalAlertingPattern2**, **Q931SignalAlertingPattern3**, **Q931SignalAlertingPattern4**, **Q931SignalAlertingPattern5**, **Q931SignalAlertingPattern6**, **Q931SignalAlertingPattern7**, **Q931SignalAlretingOff** = 0x4f, **Q931SignalErrorInIE** = 0x100 }

- enum **Q931NumberingPlanCodes** { **Q931UnknownPlan** = 0x00, **Q931ISDNPlan** = 0x01, **Q931DataPlan** = 0x03, **Q931TelexPlan** = 0x04, **Q931NationalStandardPlan** = 0x08, **Q931PrivatePlan** = 0x09, **Q931ReservedPlan** = 0x0f }
- enum **Q931TypeOfNumberCodes** { **Q931UnknownType** = 0x00, **Q931InternationalType** = 0x01, **Q931NationalType** = 0x02, **Q931NetworkSpecificType** = 0x03, **Q931SubscriberType** = 0x04, **Q931AbbreviatedType** = 0x06, **Q931ReservedType** = 0x07 }

Functions

- EXTERN int **ooOnReceivedSetup** (**ooCallData** *call, **Q931Message** *q931Msg)

This function is used to process a received SETUP message.

- EXTERN int **ooOnReceivedSignalConnect** (**ooCallData** *call, **Q931Message** *q931Msg)

This function is used to process a received CONNECT message.

- EXTERN int **ooHandleH2250Message** (**ooCallData** *call, **Q931Message** *q931Msg)

This function is used to handle received H.2250 messages.

- EXTERN int **ooOnReceivedFacility** (**ooCallData** *call, **Q931Message** *pQ931Msg)

This function is used to process a received Facility message.

- EXTERN int **ooHandleTunneledH245Messages** (**ooCallData** *call, H225H323_UU_PDU *pH323UUPdu)

This function is used to process tunneled H245 messages.

- EXTERN int **ooHandleStartH245FacilityMessage** (**ooCallData** *call, H225Facility_UUIE *facility)

This is a helper function used to handle an startH245 Facility message.

- EXTERN int **ooRetrieveAliases** (**ooCallData** *call, H225_SeqOfH225AliasAddress *pAddresses, int remote)

This function is used to retrieve the aliases from Sequence of alias addresses.

- EXTERN int **ooQ931Decode** (**Q931Message** *msg, int length, ASN1OCTET *data)

This function is invoked to decode a Q931 message.

- EXTERN int **ooDecodeUUIE** (**Q931Message** *q931Msg)

This function is used to decode the UUIE of the message from the list of ies.

- EXTERN int **ooEncodeUUIE** (**Q931Message** *q931msg)

This function is used to encode the UUIE field of the Q931 message.

- EXTERN Q931InformationElement * **ooQ931GetIE** (const **Q931Message** *q931msg, int ieCode)

This function is invoked to retrieve an IE element from a Q931 message.

- EXTERN void **ooQ931Print** (const **Q931Message** *q931msg)

This function is invoked to print a Q931 message.

- EXTERN int **ooCreateQ931Message** (**Q931Message** **msg, int msgType)

This function is invoked to create an outgoing Q931 message.

- EXTERN ASN1USINT **ooGenerateCallReference** ()

This function is invoked to generate a unique call reference number.

- EXTERN int **ooGenerateCallIdentifier** (H225CallIdentifier *callid)

This function is used to generate a unique call identifier for the call.

- EXTERN int **ooFreeQ931Message** (**Q931Message** *q931Msg)

This function is invoked to release the memory used up by a Q931 message.

- EXTERN int **ooGetOutgoingQ931Msgbuf** (**ooCallData** *call, ASN1OCTET *msgbuf, int *len, int *msgType)

This function is invoked to retrive the outgoing message buffer for Q931 message.

- EXTERN int **ooSendReleaseComplete** (**ooCallData** *call)

This function is invoked to send a ReleaseComplete message for the currently active call.

- EXTERN int **ooSendCallProceeding** (**ooCallData** *call)

This function is invoked to send a call proceeding message in response to received setup message.

- EXTERN int **ooSendAlerting** (**ooCallData** *call)

This function is invoked to send alerting message in response to received setup message.

- EXTERN int **ooSendFacility** (**ooCallData** *call)

This function is invoked to send Facility message.

- EXTERN int **ooSendConnect** (**ooCallData** *call)

This function is invoked to send a Connect message in response to received setup message.

- EXTERN int **ooH323MakeCall** (char *destip, int port, char *callToken)

This function is used to send a SETUP message for outgoing call.

- EXTERN int **ooH323HangCall** (char *callToken)

This function is used to handup a currently active call.

- EXTERN int **ooAcceptCall** (**ooCallData** *call)

Function to accept a call by sending connect.

- EXTERN int **ooH323MakeCall_helper** (**ooCallData** *call)

An helper function to ooMakeCall.

Function Documentation

EXTERN int ooAcceptCall (ooCallData * *call*)

Function to accept a call by sending connect.

This function is used as a helper function to ooSendConnect.

Parameters:

call Pointer to the call for which connect has to be sent

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooCreateQ931Message (Q931Message ** *msg*, int *msgType*)

This function is invoked to create an outgoing Q931 message.

Parameters:

msg Reference to the pointer of type Q931 message.
msgType Type of Q931 message to be created

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooDecodeUUIE (Q931Message * *q931Msg*)

This function is used to decode the UUIE of the message from the list of ies.

It decodes the User-User ie and populates the userInfo field of the message.

Parameters:

q931Msg Pointer to the message whose User-User ie has to be decoded.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooEncodeUUIE (Q931Message * *q931msg*)

This function is used to encode the UUIE field of the Q931 message.

It encodes UUIE and adds the encoded data to the list of ies.

Parameters:

q931msg Pointer to the Q931 message whose UUIE field has to be encoded.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooFreeQ931Message (Q931Message * *q931Msg*)

This function is invoked to release the memory used up by a Q931 message.

Parameters:

q931Msg Pointer to a Q931 message which has to be freed.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooGenerateCallIdentifier (H225CallIdentifier * *callid*)

This function is used to generate a unique call identifier for the call.

Parameters:

callid Pointer to the callid structure, which will be populated with the generated callid.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN ASN1USINT ooGenerateCallReference ()

This function is invoked to generate a unique call reference number.

Parameters:

None

Returns:

- call reference number

EXTERN int ooGetOutgoingQ931Msgbuf (ooCallData * *call*, ASN1OCTET * *msgbuf*, int * *len*, int * *msgType*)

This function is invoked to retrive the outgoing message buffer for Q931 message.

Parameters:

call Pointer to call for which outgoing Q931 message has to be retrieved.
msgbuf Pointer to a buffer in which retrieved message will be returned.
len Pointer to int in which length of the buffer will be returned.
msgType Pointer to integer in which message type of the ougoing message is returned.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooH323HangCall (char * *callToken*)

This function is used to handup a currently active call.

It sets the call state to CLEARING and initiates closing of all logical channels.

Parameters:

callToken Unique token of the call to be hanged.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooH323MakeCall (char * *destip*, int *port*, char * *callToken*)

This function is used to send a SETUP message for outgoing call.

It first creates an H.225 TCP connection with the remote end point and then sends SETUP message over this connection.

Parameters:

destip Dotted IP address of the remote end point.
port Port at which remote endpoint is listening for calls.
callToken Unique token for the new call.

Returns:

OO_OK, on success. OO_FAILED, on failure

EXTERN int ooH323MakeCall_helper (ooCallData * *call*)

An helper function to ooMakeCall.

Parameters:

call Pointer to the new call.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooHandleH2250Message (ooCallData * *call*, Q931Message * *q931Msg*)

This function is used to handle received H.2250 messages.

It calls helper functions based on the type of message received.

Parameters:

call Pointer to the call for which a H.2250 message is received
q931Msg Pointer to the received q931Msg

Returns:

OO_OK, on success. OO_FAILED, on failure

EXTERN int ooHandleStartH245FacilityMessage (ooCallData * *call*, H225Facility_UUIE * *facility*)

This is a helper function used to handle an startH245 Facility message.

Parameters:

call Handle to the call
facility Pointer to the facility message.

EXTERN int ooHandleTunneledH245Messages (ooCallData * *call*, H225H323_UU_PDU * *pH323UUPdu*)

This function is used to process tunneled H245 messages.

Parameters:

call Handle to the call
pH323UUPdu Pointer to the pdu containing tunneled messages.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooOnReceivedFacility (ooCallData * *call*, Q931Message * *pQ931Msg*)

This function is used to process a received Facility message.

Parameters:

call Handle to the call for which message has been received.
pQ931Msg Pointer the the received Facility message.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooOnReceivedSetup (ooCallData * *call*, Q931Message * *q931Msg*)

This function is used to process a received SETUP message.

Parameters:

call Pointer to call for which SETUP message is received.
q931Msg Pointer to the received SETUP message.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooOnReceivedSignalConnect (ooCallData * *call*, Q931Message * *q931Msg*)

This function is used to process a received CONNECT message.

It creates H.245 negotiation channel, and starts TCS and MSD procedures.

Parameters:

call Pointer to call for which CONNECT message is received.
q931Msg Pointer to the received q931Msg

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooQ931Decode (Q931Message * *msg*, int *length*, ASN1OCTET * *data*)

This function is invoked to decode a Q931 message.

Parameters:

msg Pointer to the Q931 message
length Length of the encoded data
data Pointer to the data to be decoded

Returns:

Completion status - 0 on success, -1 on failure

EXTERN Q931InformationElement* ooQ931GetIE (const Q931Message * *q931msg*, int *ieCode*)

This function is invoked to retrieve an IE element from a Q931 message.

Parameters:

q931msg Pointer to the Q931 message
ieCode IE code for the IE element to be retrieved

Returns:

Pointer to a Q931InformationElement contating the IE element.

EXTERN void ooQ931Print (const Q931Message * *q931msg*)

This function is invoked to print a Q931 message.

Parameters:

q931msg Pointer to the Q931 message

Returns:

- none

EXTERN int ooRetrieveAliases (ooCallData * *call*, H225_SeqOfH225AliasAddress * *pAddresses*, int *remote*)

This function is used to retrieve the aliases from Sequence of alias addresses.

Parameters:

call Handle to the call.
pAddresses Pointer to the sequence of alias addresses.
remote Indicates whether aliases are for remote host.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooSendAlerting (ooCallData * *call*)

This function is invoked to send alerting message in response to received setup message.

Parameters:

call Pointer to the call for which Alerting message have to be sent.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooSendCallProceeding (ooCallData * *call*)

This function is invoked to send a call proceeding message in response to received setup message.

Parameters:

call Pointer to the call for which CallProceeding message have to be sent.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooSendConnect (ooCallData * *call*)

This function is invoked to send a Connect message in response to received setup message.

Parameters:

call Pointer to the call for which connect message has to be sent.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooSendFacility (ooCallData * *call*)

This function is invoked to send Facility message.

Parameters:

call Pointer to the call for which Facility message have to be sent.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooSendReleaseComplete (ooCallData * *call*)

This function is invoked to send a ReleaseComplete message for the currently active call.

Parameters:

call Pointer to the call for which ReleaseComplete message have to be sent.

Returns:

Completion status - 0 on success, -1 on failure

Media plug-in Interface definitions

Typedefs

- typedef int(* **MediaAPI_CreateTxRTPChan**)(int *channelId, char *destip, int port)

Signature for function to Create Tx RTP channel.

- typedef int(* **MediaAPI_CloseTxRTPChan**)(int)

Signature for function to Close Tx RTP channel.

- typedef int(* **MediaAPI_CreateRecvRTPChan**)(int *channelId, char *localip, int localport)

Signature for function to Create Rx RTP channel.

- typedef int(* **MediaAPI_CloseRecvRTPChan**)(int)

Signature for function to Close Rx RTP channel.

- typedef int(* **MediaAPI_StartTxWaveFile**)(int channelId, char *filename)

Signature for function to Start transmission of media file.

- typedef int(* **MediaAPI_StopTxWaveFile**)(int channelId)

Signature for function to Stop transmission of media file.

- typedef int(* **MediaAPI_StartTxMic**)(int channelId)

Signature for function to Start transmitting captured audio from microphone.

- typedef int(* **MediaAPI_StopTxMic**)(int channelId)

Signature for function to Stop transmitting microphone data.

- typedef int(* **MediaAPI_StartRecvAndPlayback**)(int channelId)

Signature for function to Start receiving rtp data and playback.

- typedef int(* **MediaAPI_StopRecvAndPlayback**)(int channelId)

Signature for function to stop receiving rtp data.

- typedef int(* **MediaAPI_InitializePlugin**)()

Signature for function to Initialize the media plug-in.

Media plugin support functions

Functions

- EXTERN int **ooLoadSndRTPPlugin** (char *name)

Loads the media plugin into the process space.

- EXTERN int **ooReleaseSndRTPPlugin** ()

Unloads the plug-in from process space.

- EXTERN int **ooCreateTransmitRTPChannel** (char *destip, int port)

Creates a transmit RTP channel.

- EXTERN int **ooCloseTransmitRTPChannel** ()

Closes a transmit RTP channel.

- EXTERN int **ooCreateReceiveRTPChannel** (char *localip, int localport)

Creates a receive RTP channel.

- EXTERN int **ooCloseReceiveRTPChannel** ()

Closes a receive RTP channel.

- EXTERN int **ooStartTransmitWaveFile** (char *filename)

Start transmitting a audio file.

- EXTERN int **ooStopTransmitWaveFile** ()

Stop transmission of a audio file.

- EXTERN int **ooStartTransmitMic** ()

Starts capturing audio data from mic and transmits it as rtp stream.

- EXTERN int **ooStopTransmitMic** ()

Stop transmission of mic audio data.

- EXTERN int **ooStartReceiveAudioAndPlayback** ()

Starts receiving rtp stream data and play it on the speakers.

- EXTERN int **ooStopReceiveAudioAndPlayback** ()

Stop receiving rtp stream data.This calls corresponding interface function of the plug-in library.

- EXTERN int **ooStartReceiveAudioAndRecord** ()

Not suuported currently.

- EXTERN int **ooStopReceiveAudioAndRecord** ()

Not supported currently.

- EXTERN int **ooSetLocalRTPAndRTCPAddrs** ()

Set local RTP and RTCP addresses for the session.

- EXTERN int **ooRTPShutDown** ()

Closes transmit and receive RTP channels, if open.

Variables

- void * **media**

Function Documentation

EXTERN int ooCloseReceiveRTPChannel ()

Closes a receive RTP channel.

Basically calls the corresponding function of the plug-in library.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooCloseTransmitRTPChannel ()

Closes a transmit RTP channel.

Basically calls the corresponding function of the plug-in library.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooCreateReceiveRTPChannel (char * *localip*, int *localport*)

Creates a receive RTP channel.

Basically calls the corresponding function of the plug-in library.

Parameters:

localip IP address of the endpoint where RTP data will be received.
localport Port number of the local endpoint

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooCreateTransmitRTPChannel (char * *destip*, int *port*)

Creates a transmit RTP channel.

Basically calls the corresponding function of the plug-in library.

Parameters:

destip IP address of the destination endpoint.
port Destination port number.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooLoadSndRTPPlugin (char * *name*)

Loads the media plugin into the process space.

Parameters:

name Name of the media plug-in library.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooReleaseSndRTPPlugin ()

Unloads the plug-in from process space.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooRTPShutDown ()

Closes transmit and receive RTP channels, if open.

This calls corresponding interface functions to close the channels.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooSetLocalRTPAndRTCPAddrs ()

Set local RTP and RTCP addresses for the session.

This function gets next available ports for RTP and RTCP communication.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooStartReceiveAudioAndPlayback ()

Starts receiving rtp stream data and play it on the speakers.

This calls corresponding interface function of the plug-in library.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooStartTransmitMic ()

Starts capturing audio data from mic and transmits it as rtp stream.

This calls corresponding interface function of the plug-in library.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooStartTransmitWaveFile (char * *filename*)

Start transmitting a audio file.

This calls corresponding function of the plug-in library.

Parameters:

filename Name of the file to be played.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooStopReceiveAudioAndPlayback ()

Stop receiving rtp stream data.This calls corresponding interface function of the plug-in library.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooStopTransmitMic ()

Stop transmission of mic audio data.

This calls corresponding interface function of the plug-in library.

Returns:

Completion status - 0 on success, -1 on failure

EXTERN int ooStopTransmitWaveFile ()

Stop transmission of a audio file.

This calls corresponding function of the plug-in library.

Returns:

Completion status - 0 on success, -1 on failure

Socket Layer

Defines

- #define **OOSOCKET_INVALID** ((**OOSOCKET**)-1)
- #define **OOIPADDR_ANY** ((**OOIPADDR**)0)
- #define **OOIPADDR_LOCAL** ((**OOIPADDR**)0x7f000001UL) /* 127.0.0.1 */

Typedefs

- typedef int **OOSOCKET**

Socket's handle.

- typedef unsigned long **OOIPADDR**

The IP address represented as unsigned long value.

Functions

- EXTERN int **ooSocketAccept** (**OOSOCKET** socket, **OOSOCKET** *pNewSocket, **OOIPADDR** *destAddr, int *destPort)

This function permits an incoming connection attempt on a socket.

- EXTERN int **ooSocketAddrToStr** (**OOIPADDR** ipAddr, char *pbuf, int bufsize)

This function converts an IP address to its string representation.

- EXTERN int **ooSocketBind** (**OOSOCKET** socket, **OOIPADDR** addr, int port)

This function associates a local address with a socket.

- EXTERN int **ooSocketClose** (**OOSOCKET** socket)

This function closes an existing socket.

- EXTERN int **ooSocketConnect** (**OOSOCKET** socket, const char *host, int port)

This function establishes a connection to a specified socket.

- EXTERN int **ooSocketCreate** (**OOSOCKET** *psocket)

This function creates a socket.

- EXTERN int **ooSocketCreateUDP** (**OOSOCKET** *psocket)

This function creates a UDP datagram socket.

- EXTERN int **ooSocketsInit** (void)

This function initiates use of sockets by an application.

- EXTERN int **ooSocketsCleanup** (void)

This function terminates use of sockets by an application.

- EXTERN int **ooSocketListen** (**OOSOCKET** socket, int maxConnection)

This function places a socket a state where it is listening for an incoming connection.

- EXTERN int **ooSocketRecv** (**OOSOCKET** socket, ASN1OCTET *pbuf, ASN1UINT bufsize)

This function receives data from a connected socket.

- EXTERN int **ooSocketRecvFrom** (**OOSOCKET** socket, ASN1OCTET *pbuf, ASN1UINT bufsize, char *remotehost, int *remoteport)

This function receives data from a connected/unconnected socket.

- EXTERN int **ooSocketSend** (**OOSOCKET** socket, const ASN1OCTET *pdata, ASN1UINT size)

This function sends data on a connected socket.

- EXTERN int **ooSocketSendTo** (**OOSOCKET** socket, const ASN1OCTET *pdata, ASN1UINT size, const char *remotehost, int remoteport)

This function sends data on a connected or unconnected socket.

- EXTERN int **ooSocketSelect** (int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct

timeval *timeout)

This function is used for synchronous monitoring of multiple sockets.

- EXTERN int **ooSocketStrToAddr** (const char *pIPAddrStr, **OOIPADDR** *pIPAddr)

This function converts the string with IP address to a double word representation.

- EXTERN int **ooGetLocalIPAddress** (char *pIPAddrs)

This function retrives the IP address of the local host.

- EXTERN long **ooHTONL** (long val)
- EXTERN short **ooHTONS** (short val)

Typedef Documentation

typedef unsigned long OOIPADDR

The IP address represented as unsigned long value.

The most significant 8 bits in this unsigned long value represent the first number of the IP address. The least significant 8 bits represent the last number of the IP address.

Definition at line 79 of file ooSocket.h.

Function Documentation

EXTERN int ooGetLocalIPAddress (char * *pIPAddrs*)

This function retrives the IP address of the local host.

Parameters:

pIPAddrs Pointer to a char buffer in which local IP address will be returned.

Returns:

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

EXTERN int ooSocketAccept (OOSOCKET *socket*, OOSOCKET * *pNewSocket*, OOIPADDR * *destAddr*, int * *destPort*)

This function permits an incoming connection attempt on a socket.

It extracts the first connection on the queue of pending connections on socket. It then creates a new socket and returns a handle to the new socket. The newly created socket is the socket that will handle the actual connection and has the same properties as original socket. See description of 'accept' socket

function for further details.

Parameters:

socket The socket's handle created by call to ::rtSocketCreate function.
pNewSocket The pointer to variable to receive the new socket's handle.
destAddr Optional pointer to a buffer that receives the IP address of the connecting entity. It may be NULL.
destPort Optional pointer to a buffer that receives the port of the connecting entity. It may be NULL.

Returns:

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

EXTERN int ooSocketAddrToStr (OOIPADDR *ipAddr*, char * *pbuf*, int *bufsize*)

This function converts an IP address to its string representation.

Parameters:

ipAddr The IP address to be converted.
pbuf Pointer to the buffer to receive a string with the IP address.
bufsize Size of the buffer.

Returns:

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

EXTERN int ooSocketBind (OOSOCKET *socket*, OOIPADDR *addr*, int *port*)

This function associates a local address with a socket.

It is used on an unconnected socket before subsequent calls to the ::rtSocketConnect or ::rtSocketListen functions. See description of 'bind' socket function for further details.

Parameters:

socket The socket's handle created by call to ::rtSocketCreate function.
addr The local IP address to assign to the socket.
port The local port number to assign to the socket.

Returns:

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

EXTERN int ooSocketClose (OOSOCKET *socket*)

This function closes an existing socket.

Parameters:

socket The socket's handle created by call to ::rtSocketCreate or ::rtSocketAccept function.

Returns:

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

EXTERN int ooSocketConnect (OOSOCKET *socket*, const char * *host*, int *port*)

This function establishes a connection to a specified socket.

It is used to create a connection to the specified destination. When the socket call completes successfully, the socket is ready to send and receive data. See description of 'connect' socket function for further details.

Parameters:

socket The socket's handle created by call to ::rtSocketCreate function.
host The null-terminated string with the IP address in the following format:
"NNN.NNN.NNN.NNN", where NNN is a number in the range (0..255).
port The destination port to connect.

Returns:

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

EXTERN int ooSocketCreate (OOSOCKET * *psocket*)

This function creates a socket.

The only streaming TCP/IP sockets are supported at the moment.

Parameters:

psocket The pointer to the socket's handle variable to receive the handle of new socket.

Returns:

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

EXTERN int ooSocketCreateUDP (OOSOCKET * *psocket*)

This function creates a UDP datagram socket.

Parameters:

psocket The pointer to the socket's handle variable to receive the handle of new socket.

Returns:

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

EXTERN int ooSocketListen (OOSOCKET *socket*, int *maxConnection*)

This function places a socket a state where it is listening for an incoming connection.

To accept connections, a socket is first created with the ::rtSocketCreate function and bound to a local address with the ::rtSocketBind function, a maxConnection for incoming connections is specified with ::rtSocketListen, and then the connections are accepted with the ::rtSocketAccept function. See description of 'listen' socket function for further details.

Parameters:

socket The socket's handle created by call to ::rtSocketCreate function.
maxConnection Maximum length of the queue of pending connections.

Returns:

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

EXTERN int ooSocketRecv (OOSOCKET *socket*, ASN1OCTET * *pbuf*, ASN1UINT *bufsize*)

This function receives data from a connected socket.

It is used to read incoming data on sockets. The socket must be connected before calling this function. See description of 'recv' socket function for further details.

Parameters:

socket The socket's handle created by call to ::rtSocketCreate or ::rtSocketAccept function.
pbuf Pointer to the buffer for the incoming data.
bufsize Length of the buffer.

Returns:

If no error occurs, returns the number of bytes received. Otherwise, the negative value is error code.

EXTERN int ooSocketRecvFrom (OOSOCKET *socket*, ASN1OCTET * *pbuf*, ASN1UINT *bufsize*, char * *remotehost*, int * *remoteport*)

This function receives data from a connected/unconnected socket.

It is used to read incoming data on sockets. It populates the remotehost and remoteport parameters with information of remote host. See description of 'recvfrom' socket function for further details.

Parameters:

socket The socket's handle created by call to ooSocketCreate

pbuf Pointer to the buffer for the incoming data.
bufsize Length of the buffer.
remotehost Pointer to a buffer in which remote ip address will be returned.
remoteport Pointer to an int in which remote port number will be returned.

Returns:

If no error occurs, returns the number of bytes received. Otherwise, the negative value is error code.

EXTERN int ooSocketsCleanup (void)

This function terminates use of sockets by an application.

This function must be called after done with sockets.

Returns:

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

EXTERN int ooSocketSelect (int *nfds*, fd_set * *readfds*, fd_set * *writefds*, fd_set * *exceptfds*, struct timeval * *timeout*)

This function is used for synchronous monitoring of multiple sockets.

For more information refer to documnetation of "select" system call.

Parameters:

nfds The highest numbered descriptor to be monitored plus one.
readfds The descriptors listed in readfds will be watched for whether read would block on them.
writefds The descriptors listed in writefds will be watched for whether write would block on them.
exceptfds The descriptors listed in exceptfds will be watched for exceptions.
timeout Upper bound on amout of time elapsed before select returns.

Returns:

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

EXTERN int ooSocketSend (OOSOCKET *socket*, const ASN1OCTET * *pdata*, ASN1UINT *size*)

This function sends data on a connected socket.

It is used to write outgoing data on a connected socket. See description of 'send' socket function for further details.

Parameters:

socket The socket's handle created by call to ::rtSocketCreate or ::rtSocketAccept function.
pdata Buffer containing the data to be transmitted.
size Length of the data in pdata.

Returns:

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

EXTERN int ooSocketSendTo (OOSOCKET *socket*, const ASN1OCTET * *pdata*, ASN1UINT *size*, const char * *remotehost*, int *remoteport*)

This function sends data on a connected or unconnected socket.

See description of 'sendto' socket function for further details.

Parameters:

socket The socket's handle created by call to ::rtSocketCreate or ::rtSocketAccept function.
pdata Buffer containing the data to be transmitted.
size Length of the data in pdata.
remotehost Remote host ip address to which data has to be sent.
remoteport Remote port ip address to which data has to be sent.

Returns:

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

EXTERN int ooSocketsInit (void)

This function initiates use of sockets by an application.

This function must be called first before use sockets.

Returns:

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

EXTERN int ooSocketStrToAddr (const char * *pIPAddrStr*, OOIPADDR * *pIPAddr*)

This function converts the string with IP address to a double word representation.

The converted address may be used with the ::rtSocketBind function.

Parameters:

pIPAddrStr The null-terminated string with the IP address in the following format: "NNN.NNN.NNN.NNN", where NNN is a number in the range (0..255).
pIPAddr Pointer to the converted IP address.

Returns:

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

Stack Control Commands

Functions

- EXTERN int **ooMakeCall** (char *destip, int port, char *callToken)

This function is used by an application to place a call.

- EXTERN int **ooHangCall** (char *callToken)

This function is used by an user application to hang a call.

- EXTERN int **ooStopMonitor** ()

This function is used by the user application to stop monitoring calls.

Function Documentation

EXTERN int ooHangCall (char * *callToken*)

This function is used by an user application to hang a call.

Parameters:

callToken The uinque token for the call.

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooMakeCall (char * *destip*, int *port*, char * *callToken*)

This function is used by an application to place a call.

Parameters:

destip Dotted IP address of the remote endpoint
port Port number at which remote endpoint is listening for calls.
callToken Pointer to a buffer in which callToken will be returned

Returns:

OO_OK, on success. OO_FAILED, on failure.

EXTERN int ooStopMonitor ()

This function is used by the user application to stop monitoring calls.

Parameters:

None

Returns:

OO_OK, on success. OO_FAILED, on failure.

Rtmem

Defines

- #define **memAlloc**(pctxt, nbytes) memHeapAlloc(&(pctxt)->pTypeMemHeap,nbytes)

Allocate memory.

- #define **memAllocZ**(pctxt, nbytes) memHeapAllocZ(&(pctxt)->pTypeMemHeap,nbytes)

Allocate and zero memory.

- #define **memRealloc**(pctxt, mem_p, nbytes) memHeapRealloc(&(pctxt)->pTypeMemHeap, (void*)mem_p, nbytes)

Reallocate memory.

- #define **memFreePtr**(pctxt, mem_p)

Free memory pointer.

- #define **memFree**(pctxt) memHeapFreeAll(&(pctxt)->pTypeMemHeap)

Free memory associated with a context.

- #define **memReset**(pctxt) memHeapReset(&(pctxt)->pTypeMemHeap)

Reset memory associated with a context.

- #define **OSCDECL**
- #define **INCRBITIDX**(pctxt)
- #define **DECODEBIT**(pctxt, pvalue)
- #define **decodeUnconsInteger**(pctxt, pvalue) decodeSemiConsInteger(pctxt, pvalue, ASN1INT_MIN)

This function will decode an unconstrained integer.

- #define **decodeUnconsUnsigned**(pctxt, pvalue) decodeSemiConsUnsigned(pctxt, pvalue, 0U)

This function will decode an unconstrained unsigned integer.

- #define **encodeUnconsInteger**(pctxt, value) encodeSemiConsInteger(pctxt,value,ASN1INT_MIN)

This function encodes an unconstrained integer.

Typedefs

- typedef void *OSCDECL * **OSMallocFunc** (size_t size)
- typedef void *OSCDECL * **OSReallocFunc** (void *ptr, size_t size)

Functions

- typedef **void** (OSCDECL *OSFreeFunc)(void *ptr)
- EXTERN void **memHeapAddRef** (void **ppvMemHeap)
- EXTERN void * **memHeapAlloc** (void **ppvMemHeap, int nbytes)
- EXTERN void * **memHeapAllocZ** (void **ppvMemHeap, int nbytes)
- EXTERN int **memHeapCheckPtr** (void **ppvMemHeap, void *mem_p)
- EXTERN int **memHeapCreate** (void **ppvMemHeap)
- EXTERN void **memHeapFreeAll** (void **ppvMemHeap)
- EXTERN void **memHeapFreePtr** (void **ppvMemHeap, void *mem_p)
- EXTERN void * **memHeapRealloc** (void **ppvMemHeap, void *mem_p, int nbytes_)
- EXTERN void **memHeapRelease** (void **ppvMemHeap)
- EXTERN void **memHeapReset** (void **ppvMemHeap)
- EXTERN void * **memHeapMarkSaved** (void **ppvMemHeap, const void *mem_p, ASN1BOOL saved)
- EXTERN void **memHeapSetProperty** (void **ppvMemHeap, ASN1UINT propId, void *pProp)
- EXTERN void **memSetAllocFuncs** (OSMallocFunc malloc_func, OSReallocFunc realloc_func, OSFreeFunc free_func)

This function sets the pointers to standard allocation functions.

- EXTERN void **memFreeOpenSeqExt** (OOCTXT *pctxt, DList *pElemList)
- EXTERN void **memHeapSetFlags** (OOCTXT *pctxt, ASN1UINT flags)
- EXTERN void **memHeapClearFlags** (OOCTXT *pctxt, ASN1UINT flags)
- EXTERN void **memHeapSetDefBlkSize** (OOCTXT *pctxt, ASN1UINT blkSize)

This function sets the pointer to standard allocation functions.

- EXTERN ASN1UINT **memHeapGetDefBlkSize** (OOCTXT *pctxt)

This function returns the actual granularity of memory blocks.

- EXTERN int **decodeBits** (OOCTXT *pctxt, ASN1UINT *pvalue, ASN1UINT nbits)

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

- EXTERN int **decodeBitString** (OOCTXT *pctxt, ASN1UINT *numbits_p, ASN1OCTET *buffer, ASN1UINT bufsiz)

This function will decode a value of the ASN.1 bit string type whose maximum size is is known in advance.

- EXTERN int **decodeBMPString** (OOCTXT *pctxt, ASN1BMPString *pvalue, Asn116BitCharSet *permCharSet)

This function will decode a variable of the ASN.1 BMP character string.

- EXTERN int **decodeByteAlign** (OOCTXT *pctxt)

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

- EXTERN int **decodeConsInteger** (OOCTXT *pctxt, ASN1INT *pvalue, ASN1INT lower, ASN1INT upper)

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

- EXTERN int **decodeConsUnsigned** (OOCTXT *pctxt, ASN1UINT *pvalue, ASN1UINT lower, ASN1UINT upper)

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

- EXTERN int **decodeConsUInt8** (OOCTXT *pctxt, ASN1UINT8 *pvalue, ASN1UINT lower, ASN1UINT upper)

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

- EXTERN int **decodeConsUInt16** (OOCTXT *pctxt, ASN1USINT *pvalue, ASN1UINT lower, ASN1UINT upper)

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

- EXTERN int **decodeConsWholeNumber** (OOCTXT *pctxt, ASN1UINT *padjusted_value, ASN1UINT range_value)

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

- EXTERN int **decodeConstrainedStringEx** (OOCTXT *pctxt, const char **string, const char *charSet, ASN1UINT abits, ASN1UINT ubits, ASN1UINT canSetBits)

This function decodes a constrained string value.

- EXTERN int **decodeDynBitString** (OOCTXT *pctxt, ASN1DynBitStr *pBitStr)

This function will decode a variable of thr ASN.1 BIT STRING type.

- EXTERN int **decodeDynOctetString** (OOCTXT *pctxt, ASN1DynOctStr *pOctStr)

This function will decode a value of the ASN.1 octet string type whose maximum size is known in advance.

- EXTERN int **decodeLength** (OOCTXT *pctxt, ASN1UINT *pvalue)

This function will decode a length determinant value.

- EXTERN int **moveBitCursor** (OOCTXT *pctxt, int bitOffset)
- EXTERN int **decodeObjectIdentifier** (OOCTXT *pctxt, ASN1OBJID *pvalue)

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

- EXTERN int **decodeOctetString** (OOCTXT *pctxt, ASN1UINT *numocts_p, ASN1OCTET *buffer, ASN1UINT bufsiz)

This function will decode a value of the ASN.1 octet string type whose maximun size is known in advance.

- EXTERN int **decodeOpenType** (OOCTXT *pctxt, const ASN1OCTET **object_p2, ASN1UINT *numocts_p)

This function will decode an ASN.1 open type.

- EXTERN int **decodeSmallNonNegWholeNumber** (OOCTXT *pctxt, ASN1UINT *pvalue)

This function will decode a small non-negative whole number as specified in Section 10.6 of the X.691 standard.

- EXTERN int **decodeSemiConsInteger** (OOCTXT *pctxt, ASN1INT *pvalue, ASN1INT lower)

This function will decode a semi-constrained integer.

- EXTERN int **decodeSemiConsUnsigned** (OOCTXT *pctxt, ASN1UINT *pvalue, ASN1UINT lower)

This function will decode a semi-constrained unsigned integer.

- EXTERN int **decodeVarWidthCharString** (OOCTXT *pctxt, const char **pvalue)
- EXTERN int **encodeBit** (OOCTXT *pctxt, ASN1BOOL value)

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

- EXTERN int **encodeBits** (OOCTXT *pctxt, ASN1UINT value, ASN1UINT nbits)

This function encodes multiple bits.

- EXTERN int **encodeBitString** (OOCTXT *pctxt, ASN1UINT numocts, const ASN1OCTET *data)

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

- EXTERN int **encodeBMPString** (OOCTXT *pctxt, ASN1BMPString value, Asn116BitCharSet *permCharSet)

This function will encode a variable of the ASN.1 BMP character string.

- EXTERN int **encodeByteAlign** (OOCTXT *pctxt)

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

- EXTERN int **encodeCheckBuffer** (OOCTXT *pctxt, ASN1UINT nbytes)

This function will determine if the given number of bytes will fit in the encode buffer.

- EXTERN int **encodeConstrainedStringEx** (OOCTXT *pctxt, const char *string, const char *charSet, ASN1UINT abits, ASN1UINT ubits, ASN1UINT canSetBits)

This function encodes a constrained string value.

- EXTERN int **encodeConsInteger** (OOCTXT *pctxt, ASN1INT value, ASN1INT lower, ASN1INT upper)

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

- EXTERN int **encodeConsUnsigned** (OOCTXT *pctxt, ASN1UINT value, ASN1UINT lower, ASN1UINT upper)

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

- EXTERN int **encodeConsWholeNumber** (OOCTXT *pctxt, ASN1UINT adjusted_value, ASN1UINT range_value)

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

- EXTERN int **encodeExpandBuffer** (OOCTXT *pctxt, ASN1UINT nbytes)

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

- EXTERN ASN1OCTET * **encodeGetMsgPtr** (OOCTXT *pctxt, int *pLength)

This function will return the message pointer and length of an encoded message.

- EXTERN int **encodeLength** (OOCTXT *pctxt, ASN1UINT value)

This function will encode a length determinant value.

- EXTERN int **encodeObjectIdentifier** (OOCTXT *pctxt, ASN1OBJID *pvalue)

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

- EXTERN int **encodebitsFromOctet** (OOCTXT *pctxt, ASN1OCTET value, ASN1UINT nbits)

This function encodes bits from a given octet to the output buffer.

- EXTERN int **encodeOctets** (OOCTXT *pctxt, const ASN1OCTET *pvalue, ASN1UINT nbits)

This fuction will encode an array of octets.

- EXTERN int **encodeOctetString** (OOCTXT *pctxt, ASN1UINT numocts, const ASN1OCTET *data)

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

- EXTERN int **encodeOpenType** (OOCTXT *pctxt, ASN1UINT numocts, const ASN1OCTET *data)

This function will encode an ASN.1 open type.

- EXTERN int **encodeOpenTypeExt** (OOCTXT *pctxt, DList *pElemList)

This function will encode an ASN.1 open type extension.

- EXTERN int **encodeOpenTypeExtBits** (OOCTXT *pctxt, DList *pElemList)
- EXTERN int **encodeSmallNonNegWholeNumber** (OOCTXT *pctxt, ASN1UINT value)

This function will endcode a small, non-negative whole number as specified in Section 10.6 of teh X.691 standard.

- EXTERN int **encodeSemiConsInteger** (OOCTXT *pctxt, ASN1INT value, ASN1INT lower)

This function encodes a semi-constrained integer.

- EXTERN int **encodeSemiConsUnsigned** (OOCTXT *pctxt, ASN1UINT value, ASN1UINT lower)

This function encodes an semi-constrained unsigned integer.

- EXTERN int **encodeVarWidthCharString** (OOCTXT *pctxt, const char *value)
- EXTERN int **addSizeConstraint** (OOCTXT *pctxt, Asn1SizeCnst *pSize)
- EXTERN ASN1BOOL **alignCharStr** (OOCTXT *pctxt, ASN1UINT len, ASN1UINT nbits, Asn1SizeCnst *pSize)
- EXTERN int **bitAndOctetStringAlignmentTest** (Asn1SizeCnst *pSizeList, ASN1UINT itemCount, ASN1BOOL bitStrFlag, ASN1BOOL *pAlignFlag)
- EXTERN int **getPERMsgLen** (OOCTXT *pctxt)
- EXTERN Asn1SizeCnst * **getSizeConstraint** (OOCTXT *pctxt, ASN1BOOL extbit)
- EXTERN int **checkSizeConstraint** (OOCTXT *pctxt, int size)
- EXTERN ASN1UINT **getUIntBitCount** (ASN1UINT value)
- EXTERN Asn1SizeCnst * **checkSize** (Asn1SizeCnst *pSizeList, ASN1UINT value, ASN1BOOL *pExtendable)
- EXTERN void **init16BitCharSet** (Asn116BitCharSet *pCharSet, ASN116BITCHAR first, ASN116BITCHAR last, ASN1UINT abits, ASN1UINT ubits)
- EXTERN ASN1BOOL **isExtendableSize** (Asn1SizeCnst *pSizeList)
- EXTERN void **set16BitCharSet** (OOCTXT *pctxt, Asn116BitCharSet *pCharSet, Asn116BitCharSet *pAlphabet)
- EXTERN const char * **rtBitStrToString** (ASN1UINT numbits, const ASN1OCTET *data, char *buffer, size_t bufsiz)
- EXTERN const char * **rtOctStrToString** (ASN1UINT numocts, const ASN1OCTET *data, char *buffer, size_t bufsiz)

Define Documentation

#define DECODEBIT(pctxt, pvalue)

Value:

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

Definition at line 1230 of file ooasn1.h.

#define decodeUnconsInteger(pctxt, pvalue) decodeSemiConsInteger(pctxt, pvalue, ASN1INT_MIN)

This function will decode an unconstrained integer.

Parameters:

pctxt 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.

Definition at line 1612 of file ooasn1.h.

#define decodeUnconsUnsigned(pctxt, pvalue) decodeSemiConsUnsigned(pctxt, pvalue, 0U)

This function will decode an unconstrained unsigned integer.

Parameters:

pctxt 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.

Definition at line 1625 of file ooasn1.h.

#define encodeUnconsInteger(pctxt, value) encodeSemiConsInteger(pctxt,value,ASN1INT_MIN)

This function encodes an unconstrained integer.

Parameters:

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

Returns:

Completion status of operation:

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

Definition at line 1999 of file ooasn1.h.

#define INCRBITIDX(pctxt)

Value:

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

Definition at line 1225 of file ooasn1.h.

#define memAlloc(pctxt, nbytes) memHeapAlloc(&(pctxt)->pTypeMemHeap,nbytes)

Allocate memory.

This macro allocates the given number of bytes. It is similar to the C `malloc` run-time function.

Parameters:

pctxt - Pointer to a context block
nbytes - Number of bytes of memory to allocate

Returns:

- Void pointer to allocated memory or NULL if insufficient memory was available to fulfill the request.

Definition at line 1044 of file ooasn1.h.

#define memAllocZ(pctxt, nbytes) memHeapAllocZ(&(pctxt)->pTypeMemHeap,nbytes)

Allocate and zero memory.

This macro allocates the given number of bytes and then initializes the memory block to zero.

Parameters:

pctxt - Pointer to a context block
nbytes - Number of bytes of memory to allocate

Returns:

- Void pointer to allocated memory or NULL if insufficient memory was available to fulfill the request.

Definition at line 1056 of file ooasn1.h.

#define memFree(pctxt) memHeapFreeAll(&(pctxt)->pTypeMemHeap)

Free memory associated with a context.

This macro frees all memory held within a context. This is all memory allocated using the memHeapAlloc (and similar macros) and the mem memory allocation functions using the given context variable.

Parameters:

pctxt - Pointer to a context block

Definition at line 1099 of file ooasn1.h.

#define memFreePtr(pctxt, mem_p)

Value:

```
if (memHeapCheckPtr (&(pctxt)->pTypeMemHeap, (void*)mem_p)) \
memHeapFreePtr(&(pctxt)->pTypeMemHeap, (void*)mem_p)
```

Free memory pointer. This macro frees memory at the given pointer. The memory must have been allocated using the memHeapAlloc (or similar) macros or the mem memory allocation macros. This macro is similar to the C `free` function.

Parameters:

pctxt - Pointer to a context block
mem_p - Pointer to memory block to free. This must have been allocated using the memHeapAlloc or memAlloc macro or the memHeapAlloc function.

Definition at line 1087 of file ooasn1.h.

#define memRealloc(pctxt, mem_p, nbytes) memHeapRealloc(&(pctxt)->pTypeMemHeap, (void*)mem_p, nbytes)

Reallocate memory.

This macro reallocates a memory block (either expands or contracts) to the given number of bytes. It is similar to the C `realloc` run-time function.

Parameters:

pctxt - Pointer to a context block
mem_p - Pointer to memory block to reallocate. This must have been allocated using the memHeapAlloc macro or the memHeapAlloc function.
nbytes - Number of bytes of memory to which the block is to be resized.

Returns:

- Void pointer to allocated memory or NULL if insufficient memory was available to fulfill the request. This may be the same as the pmem pointer that was passed in if the block did not need to be relocated.

Definition at line 1073 of file ooasn1.h.

#define memReset(pctxt) memHeapReset(&(pctxt)->pTypeMemHeap)

Reset memory associated with a context.

This macro resets all memory held within a context. This is all memory allocated using the memHeapAlloc (and similar macros) and the mem memory allocation functions using the given context variable.

The difference between this and the ASN1MEMFREE macro is that the memory blocks held within the context are not actually freed. Internal pointers are reset so the existing blocks can be reused. This can provide a performace improvement for repetitive tasks such as decoding messages in a loop.

Parameters:

pctxt - Pointer to a context block

Definition at line 1116 of file ooasn1.h.

Function Documentation

EXTERN int decodeBits (OOCTXT * *pctxt*, ASN1UINT * *pvalue*, ASN1UINT *nbits*)

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

Parameters:

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

EXTERN int decodeBitString (OOCTXT * *pctxt*, ASN1UINT * *numbits_p*, ASN1OCTET * *buffer*, ASN1UINT *bufsiz*)

This function will decode a value of the ASN.1 bit string type whose maximum size is is known in advance.

The ASN1C complier generates a call to this function to decode bit string productions or elements that contain a size constraint.

Parameters:

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

EXTERN int decodeBMPString (OOCTXT * *pctxt*, 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:

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

EXTERN int decodeByteAlign (OOCTXT * *pctxt*)

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

Parameters:

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

Returns:

Completion status of operation:

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

EXTERN int decodeConsInteger (OOCTXT * *pctxt*, ASN1INT * *pvalue*, ASN1INT *lower*, ASN1INT *upper*)

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

Parameters:

pctxt 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.

EXTERN int decodeConstrainedStringEx (OOCTXT * *pctxt*, const char ** *string*, const char * *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:

pctxt 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.

EXTERN int decodeConsUInt16 (OOCTXT * *pctxt*, 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:

pctxt 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.

EXTERN int decodeConsUInt8 (OOCTXT * *pctxt*, 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:

pctxt 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.

EXTERN int decodeConsUnsigned (OOCTXT * *pctxt*, ASN1UINT * *pvalue*, ASN1UINT *lower*, ASN1UINT *upper*)

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

Parameters:

pctxt 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.

EXTERN int decodeConsWholeNumber (OOCTXT * *pctxt*, 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:

pctxt Pointer to context block structure.
padjusted_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.

EXTERN int decodeDynBitString (OOCTXT * *pctxt*, ASN1DynBitStr * *pBitStr*)

This function will decode a variable of thr ASN.1 BIT STRING type.

This function allocates dynamic memory t store the decoded result. The ASN1C complier generates a call to this function to decode an unconstrained bit string production or element.

Parameters:

pctxt A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
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 memAlloc function. This memory is tracked within the context and released when the freeContext function is invoked.

Returns:

Completion status of operation:

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

EXTERN int decodeDynOctetString (OOCTXT * *pctxt*, ASN1DynOctStr * *pOctStr*)

This function will decode a value of the ASN.1 octet string type whose maximum size is known in

advance.

The ASN1C complier generates a call to this function to decode octet string productions or elements that contain a size constraint.

Parameters:

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

EXTERN int decodeLength (OOCTXT * *pctxt*, ASN1UINT * *pvalue*)

This function will decode a length determinant value.

Parameters:

pctxt A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
pvalue A pointer to an unsigned integer variable to receive the decoded length value.

Returns:

Completion status of operation:

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

EXTERN int decodeObjectIdentifier (OOCTXT * *pctxt*, ASN1OBJID * *pvalue*)

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

Parameters:

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

Returns:

Completion status of operation:

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

EXTERN int decodeOctetString (OOCTXT * *pctxt*, ASN1UINT * *numocts_p*, ASN1OCTET * *buffer*, ASN1UINT *bufsiz*)

This function will decode a value of the ASN.1 octet string type whose maximun 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:

pctxt 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.

EXTERN int decodeOpenType (OOCTXT * *pctxt*, const ASN1OCTET ** *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:

pctxt 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.

EXTERN int decodeSemiConsInteger (OOCTXT * *pctxt*, ASN1INT * *pvalue*, ASN1INT *lower*)

This function will decode a semi-constrained integer.

Parameters:

pctxt 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.

EXTERN int decodeSemiConsUnsigned (OOCTXT * *pctxt*, ASN1UINT * *pvalue*, ASN1UINT *lower*)

This function will decode a semi-constrained unsigned integer.

Parameters:

pctxt 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.

EXTERN int decodeSmallNonNegWholeNumber (OOCTXT * *pctxt*, 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:

pctxt 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 t receive decoded results.

Returns:

Completion status of operation:

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

EXTERN int encodeBit (OOCTXT * *pctxt*, ASN1BOOL *value*)

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

Parameters:

pctxt 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.

EXTERN int encodeBits (OOCTXT * *pctxt*, ASN1UINT *value*, ASN1UINT *nbits*)

This function encodes multiple bits.

Parameters:

pctxt 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.

EXTERN int encodebitsFromOctet (OOCTXT * *pctxt*, ASN1OCTET *value*, ASN1UINT *nbits*)

This function encodes bits from a given octet to the output buffer.

Parameters:

pctxt Pointer to ASN.1 PER context structure
value Value of bits to be encoded
nbits Number of bits to be encoded

Returns:

Status of operation

EXTERN int encodeBitString (OOCTXT * *pctxt*, ASN1UINT *numocts*, const ASN1OCTET * *data*)

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

Parameters:

pctxt A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
numocts The number of bits n 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.

EXTERN int encodeBMPString (OOCTXT * *pctxt*, 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:

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

EXTERN int encodeByteAlign (OOCTXT * *pctxt*)

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

Parameters:

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

Returns:

Completion status of operation:

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

EXTERN int encodeCheckBuffer (OOCTXT * *pctxt*, ASN1UINT *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:

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

EXTERN int encodeConsInteger (OOCTXT * *pctxt*, ASN1INT *value*, ASN1INT *lower*, ASN1INT *upper*)

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

Parameters:

pctxt 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.

EXTERN int encodeConstrainedStringEx (OOCTXT * *pctxt*, const char * *string*, const char * *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:

pctxt 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.

EXTERN int encodeConsUnsigned (OOCTXT * *pctxt*, 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 >= MAXINT

Parameters:

pctxt 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.

EXTERN int encodeConsWholeNumber (OOCTXT * *pctxt*, 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:

pctxt 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.

EXTERN int encodeExpandBuffer (OOCTXT * *pctxt*, ASN1UINT *nbytes*)

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

Parameters:

pctxt 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.

EXTERN ASN1OCTET* encodeGetMsgPtr (OOCTXT * *pctxt*, int * *pLength*)

This function will return the message pointer and length of an encoded message.

This function is called after a complier 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 adn the encodeGetMsgLen function can be used to obtain the lenght of the message.

Parameters:

pctxt 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.

EXTERN int encodeLength (OOCTXT * *pctxt*, ASN1UINT *value*)

This function will encode a length determinant value.

Parameters:

pctxt 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.

EXTERN int encodeObjectIdentifier (OOCTXT * *pctxt*, ASN1OBJID * *pvalue*)

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

Parameters:

pctxt 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.

EXTERN int encodeOctets (OOCTXT * *pctxt*, const ASN1OCTET * *pvalue*, ASN1UINT *nbits*)

This fuction will encode an array of octets.

The Octets will be encoded unaligned starting at the current bit offset within the encode buffer.

Parameters:

pctxt 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.

EXTERN int encodeOctetString (OOCTXT * *pctxt*, ASN1UINT *numocts*, const ASN1OCTET * *data*)

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

Parameters:

pctxt 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.

EXTERN int encodeOpenType (OOCTXT * *pctxt*, ASN1UINT *numocts*, const ASN1OCTET * *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:

pctxt 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.

EXTERN int encodeOpenTypeExt (OOCTXT * *pctxt*, DList * *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:

pctxt 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.

EXTERN int encodeSemiConsInteger (OOCTXT * *pctxt*, ASN1INT *value*, ASN1INT *lower*)

This function encodes a semi-constrained integer.

Parameters:

pctxt 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.

EXTERN int encodeSemiConsUnsigned (OOCTXT * *pctxt*, ASN1UINT *value*, ASN1UINT *lower*)

This function encodes an semi-constrained unsigned integer.

Parameters:

pctxt 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.

EXTERN int encodeSmallNonNegWholeNumber (OOCTXT * *pctxt*, ASN1UINT *value*)

This function will endcode a small, non-negative whole number as specified in Section 10.6 of teh 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:

pctxt 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.

EXTERN ASN1UINT memHeapGetDefBlkSize (OOCTXT * *pctxt*)

This function returns the actual granularity of memory blocks.

Parameters:

pctxt Pointer to a context block.

EXTERN void memHeapSetDefBlkSize (OOCTXT * *pctxt*, ASN1UINT *blkSize*)

This function sets the pointer to standard allocation functions.

These functions are used to allocate/reallocate/free the memory blocks. By default, standard C functions - malloc, realloc, and free - are used. But if some platforms do not support these functions or some other reasons exist) they can be overloaded. The functions being overloaded should have the same prototypes as standard ones.

Parameters:

pctxt Pointer to a context block.
blkSize The currently used minimum size and the granularity of memory blocks.

EXTERN void memSetAllocFuncs (OSMallocFunc *malloc_func*, OSReallocFunc *realloc_func*, OSFreeFunc *free_func*)

This function sets the pointers to standard allocation functions.

These functions are used to allocate/reallocate/free the memory blocks. By default, standard C functions - 'malloc', 'realloc' and 'free' - are used. But if some platforms do not support these functions (or some other reasons exist) they can be overloaded. The functions being overloaded should have the same prototypes as standard ones.

Parameters:

malloc_func Pointer to the memory allocation function ('malloc' by default).
realloc_func Pointer to the memory reallocation function ('realloc' by default).
free_func Pointer to the memory deallocation function ('free' by default).

EXTERN int moveBitCursor (OOCTXT * *pctxt*, int *bitOffset*)

Parameters:

pctxt A pointer to a context structure. This provides a storage area for the function to store all working variables that must be maintained between function calls.
bitOffset The bit offset inside the message buffer.

H323FrameworkStack Data Structure Documentation

Asn1NamedCEventHandler Struct Reference

This is a basic C based event handler structure, which can be used to define user-defined event handlers.

```
#include <asn1CEvtHndlr.h>
```

Data Fields

- **rtStartElement startElement**
- **rtEndElement endElement**
- **rtBoolValue boolValue**
- **rtIntValue intValue**
- **rtUIntValue uIntValue**
- **rtBitStrValue bitStrValue**
- **rtOctStrValue octStrValue**
- **rtCharStrValue charStrValue**
- **rtCharStrValue16Bit charStrValue16Bit**
- **rtCharStrValue32Bit charStrValue32Bit**
- **rtNullValue nullValue**
- **rtOidValue oidValue**
- **rtRealValue realValue**
- **rtEnumValue enumValue**
- **rtOpenTypeValue openTypeValue**

Detailed Description

This is a basic C based event handler structure, which can be used to define user-defined event handlers.

Definition at line 234 of file asn1CEvtHndlr.h.

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

- **asn1CEvtHndlr.h**

H245Message Struct Reference

Defines the H245 message structure.

```
#include <ootypes.h>
```

Data Fields

- OOCTXT * **pctxt**
- H245MultimediaSystemControlMessage **h245Msg**
- ASN1UINT **msgType**

Detailed Description

Defines the H245 message structure.

All request/response and command messages are represented using this structure.

Definition at line 340 of file ootypes.h.

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

- **ootypes.h**

ooCallData Struct Reference

Structure to store all the information related to a particular call.

`#include <ootypes.h>`

Data Fields

- OOCTXT * **pctxt**
- char **callToken** [20]
- char **callType** [10]
- ASN1USINT **callReference**
- H225CallIdentifier **callIdentifier**
- int **callState**
- int **callEndReason**
- int **h245SessionState**
- int **isTunnelingActive**
- int **isFastStartActive**
- ooMediaInfo * **mediaInfo**
- char **localIP** [20]
- **OOSOCKET * h225Channel**
- int * **h225ChanPort**
- **OOSOCKET * h245Channel**
- int * **h245ChanPort**
- **OOSOCKET * h245listener**
- int * **h245listenport**
- char **remoteIP** [20]
- int **remotePort**
- int **remoteH245Port**
- int **sendH225**
- int **sendH245**
- DList **outH225Queue**
- DList **outH245Queue**
- ASN1OCTET * **remoteDisplayName**
- ooAliases * **remoteAliases**
- ooAliases * **localAliases**
- int **masterSlaveState**
- ASN1UINT **statusDeterminationNumber**
- int **localTermCapState**
- int **remoteTermCapState**
- **H245Message * remoteTermCapSet**
- DList **remoteFastStartOLCs**
- ASN1UINT8 **remoteTermCapSeqNo**
- ASN1UINT8 **localTermCapSeqNo**
- **ooLogicalChannel * logicalChans**
- int **noOfLogicalChannels**
- int **logicalChanNoBase**
- int **logicalChanNoMax**
- int **logicalChanNoCur**
- int **isAudioActive**
- **ooCallData * next**
- **ooCallData * prev**

Detailed Description

Structure to store all the information related to a particular call.

Definition at line 389 of file ootypes.h.

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

- **ootypes.h**

ooCommand Struct Reference

Structure for stack commands.

`#include <ootypes.h>`

Data Fields

- int **type**
- void * **param1**
- void * **param2**
- void * **param3**

Detailed Description

Structure for stack commands.

Definition at line 302 of file ootypes.h.

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

- **ootypes.h**

ooEndPoint Struct Reference

Structure to store all the config information related to the endpoint created by an application.

`#include <ootypes.h>`

Data Fields

- OOCTXT **ctxt**
- FILE * **fptraceFile**
- **ooH323Ports tcpPorts**

Range of port numbers to be used for TCP connections.

- **ooH323Ports udpPorts**

Range of port numbers to be used for UDP connections.

- **ooH323Ports rtpPorts**

Range of port numbers to be used for RTP connections.

- int **h245Tunneling**
- int **fastStart**
- int **termType**
- int **t35CountryCode**
- int **t35Extension**
- int **manufacturerCode**
- char * **productID**
- char * **versionID**
- char * **callerid**
- char * **callername**
- **OOSOCKET * stackSocket**
- int **callType**
- DList **audioCaps**
- DList **dataApplicationCaps**
- DList **videoCaps**
- cb_OnIncomingCall **onIncomingCall**
- cb_OnOutgoingCall **onOutgoingCall**
- cb_OnCallEstablished **onCallEstablished**
- cb_OnCallCleared **onCallCleared**
- cb_OnStartLogicalChannel **onStartLogicalChannel**
- int **listenPort**
- **OOSOCKET * listener**
- **ooCallData * callList**
- int **callMode**

Detailed Description

Structure to store all the config information related to the endpoint created by an application.

Definition at line 480 of file ootypes.h.

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

- **ootypes.h**

ooH323EpCapability Struct Reference

Structure to store information related to end point capability.

`#include <ootypes.h>`

Data Fields

- int **t**
- void * **cap**
- **cb_StartReceiveChannel startReceiveChannel**
- **cb_StartTransmitChannel startTransmitChannel**
- **cb_StopReceiveChannel stopReceiveChannel**
- **cb_StopTransmitChannel stopTransmitChannel**

Detailed Description

Structure to store information related to end point capability.

Definition at line 460 of file ootypes.h.

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

- **ootypes.h**

ooH323Ports Struct Reference

This structure is used to define the port ranges to be used by the application.

`#include <ootypes.h>`

Data Fields

- int **start**
- int **max**
- int **current**

Detailed Description

This structure is used to define the port ranges to be used by the application.

Definition at line 312 of file ootypes.h.

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

- **ootypes.h**

ooLogicalChannel Struct Reference

Structure to store information of logical channels for call.

```
#include <ootypes.h>
```

Data Fields

- int **channelNo**
- int **sessionID**
- char **type** [10]
- char **dir** [10]
- char **remoteIP** [20]
- int **remoteRtpPort**
- int **remoteRtcpPort**
- int **localRtpPort**
- int **localRtcpPort**
- char **localIP** [20]
- int **state**
- **ooH323EpCapability * chanCap**
- **ooLogicalChannel * next**

Detailed Description

Structure to store information of logical channels for call.

Definition at line 363 of file ootypes.h.

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

- **ootypes.h**

Q931Message Struct Reference

Defines the Q931 message structure.

```
#include <ootypes.h>
```

Data Fields

- OOCTXT * **pctxt**
- unsigned **protocolDiscriminator**
- unsigned **callReference**
- ASN1BOOL **fromDestination**
- unsigned **messageType**
- DList **ies**
- H225H323_UserInformation * **userInfo**

Detailed Description

Defines the Q931 message structure.

Contains context for memory allocation, protocol Discriminator, call reference, meesage type and list of user user IEs.

Definition at line 324 of file ootypes.h.

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

- **ootypes.h**

H323FrameworkStack File Documentation

asn1CEvtHndlr.h File Reference

C event handler structure.

```
#include <stdio.h>
#include "ooasn1.h"
```

Data Structures

- struct **Asn1NamedCEventHandler**

This is a basic C based event handler structure, which can be used to define user-defined event handlers.

Typedefs

- typedef void(* **rtStartElement**)(const char *name, int index)

This is a function pointer for a callback function which is invoked from within a decode function when an element of a SEQUENCE, SET, SEQUENCE OF, SET OF, or CHOICE construct is parsed.

- typedef void(* **rtEndElement**)(const char *name, int index)

This is a function pointer for a callback function which is invoked from within a decode function when parsing is complete on an element of a SEQUENCE, SET, SEQUENCE OF, SET OF, or CHOICE construct.

- typedef void(* **rtBoolValue**)(ASN1BOOL value)

This is a function pointer for a callback function which is invoked from within a decode function when a value of the BOOLEAN ASN.1 type is parsed.

- typedef void(* **rtIntValue**)(ASN1INT value)

This is a function pointer for a callback function which is invoked from within a decode function when a value of the INTERGER ASN.1 type is parsed.

- typedef void(* **rtUIntValue**)(ASN1UINT value)

This is a function pointer for a callback function which is invoked from within a decode function when a value of the INTEGER ASN.1 type is parsed.

- typedef void(* **rtBitStrValue**)(ASN1UINT numbits, const ASN1OCTET *data)

This is a function pointer for a callback function which is invoked from within a decode function when a value of the BIT STRING ASN.1 type is parsed.

- typedef void(* **rtOctStrValue**)(ASN1UINT numocts, const ASN1OCTET *data)

This is a function pointer for a callback function which is invoked from within a decode function

when a value of one of the OCTET STRING ASN.1 type is parsed.

- typedef void(* **rtCharStrValue**)(const char *value)

This is a function pointer for a callback function which is invoked from within a decode function when a value of one of the 8-bit ASN.1 character string types is parsed.

- typedef void(* **rtCharStrValue16Bit**)(ASN1UINT nchars, ASN116BITCHAR *data)

This is a function pointer for a callback function which is invoked from within a decode function when a value of one of the 16-bit ASN.1 character string types is parsed.

- typedef void(* **rtCharStrValue32Bit**)(ASN1UINT nchars, ASN132BITCHAR *data)

This is a function pointer for a callback function which is invoked from within a decode function when a value of one of the 32-bit ASN.1 characer string types is parsed.

- typedef void(* **rtNullValue**)()

This is a function pointer for a callback function which is invoked from within a decode function when a value of the NULL ASN.1 type is parsed.

- typedef void(* **rtOidValue**)(ASN1UINT numSubIds, ASN1UINT *pSubIds)

This is a function pointer for a callback function which is invoked from within a decode function whn a value the OBJECT IDENTIFIER ASN.1 type is parsed.

- typedef void(* **rtRealValue**)(double value)

This is a function pointer for a callback function which is invoked from within a decode function when a value the REAL ASN.1 type is parsed.

- typedef void(* **rtEnumValue**)(ASN1UINT value)

This is a function pointer for a callback function which is invoked from within a decode function when a value of the ENUMERATED ASN.1 type is parsed.

- typedef void(* **rtOpenTypeValue**)(ASN1UINT numocts, const ASN1OCTET *data)

This is a function pointer for a callback function which is invoked from within a decode function when an ASN.1 open type is parsed.

- typedef **Asn1NamedCEventHandler Asn1NamedCEventHandler**

This is a basic C based event handler structure, which can be used to define user-defined event handlers.

Functions

- EXTERN void **rtAddEventHandler** (OOCTXT *pCtxt, **Asn1NamedCEventHandler** *pHandler)

This function is called to add a new event handler to the context event handler list.

- EXTERN void **rtRemoveEventHandler** (OOCTXT *pCtxt, **Asn1NamedCEventHandler** *pHandler)

This function is called to remove an event handler from the context event handler list.

- EXTERN void **rtInvokeStartElement** (OOCTXT *pCtxt, const char *name, int index)

The following functions are invoked from within the generated code to call the various user-defined event handler methods ..

- EXTERN void **rtInvokeEndElement** (OOCTXT *pCtxt, const char *name, int index)
- EXTERN void **rtInvokeBoolValue** (OOCTXT *pCtxt, ASN1BOOL value)
- EXTERN void **rtInvokeIntValue** (OOCTXT *pCtxt, ASN1INT value)
- EXTERN void **rtInvokeUIntValue** (OOCTXT *pCtxt, ASN1UINT value)
- EXTERN void **rtInvokeBitStrValue** (OOCTXT *pCtxt, ASN1UINT numbits, const ASN1OCTET *data)
- EXTERN void **rtInvokeOctStrValue** (OOCTXT *pCtxt, ASN1UINT numocts, const ASN1OCTET *data)
- EXTERN void **rtInvokeCharStrValue** (OOCTXT *pCtxt, const char *value)
- EXTERN void **rtInvokeCharStr16BitValue** (OOCTXT *pCtxt, ASN1UINT nchars, ASN116BITCHAR *data)
- EXTERN void **rtInvokeCharStr32BitValue** (OOCTXT *pCtxt, ASN1UINT nchars, ASN132BITCHAR *data)
- EXTERN void **rtInvokeNullValue** (OOCTXT *pCtxt)
- EXTERN void **rtInvokeOidValue** (OOCTXT *pCtxt, ASN1UINT numSubIds, ASN1UINT *pSubIds)
- EXTERN void **rtInvokeRealValue** (OOCTXT *pCtxt, double value)
- EXTERN void **rtInvokeEnumValue** (OOCTXT *pCtxt, ASN1UINT value)
- EXTERN void **rtInvokeOpenTypeValue** (OOCTXT *pCtxt, ASN1UINT numocts, const ASN1OCTET *data)

Detailed Description

C event handler structure.
The ASN1CEventHandler is a structure type which can be used to define event handlers by the user.
Definition in file **asn1CEvtHndlr.h**.

oo.h File Reference

This file defines the trace functionality.

```
#include "ootypes.h"
```

Functions

- EXTERN void **ooTrace** (const char *fmtspec,...)

This function is used to write the messages to the trace file.

Detailed Description

This file defines the trace functionality.

Definition in file **oo.h**.

Function Documentation

EXTERN void ooTrace (const char * *fmtspec*, ...)

This function is used to write the messages to the trace file.

Parameters:

fmtspec Printf style format spec.
... Printf style variable list of arguments

Returns:

- none

ooasn1.h File Reference

Common ASN.1 runtime constants, data structure definitions, and run-time functions to support BER/DER/PER as defined in the ITU-T standards.

```
#include <limits.h>
#include <stdio.h>
#include <string.h>
#include <stddef.h>
```

Data Structures

- struct **ASN1OBJID**
- struct **ASN1OctStr**
- struct **ASN1DynOctStr**
- struct **ASN1DynBitStr**
- struct **ASN1SeqOf**
- struct **ASN1SeqOfOctStr**
- struct **ASN1OpenType**
- struct **Asn116BitCharString**
- struct **Asn132BitCharString**
- struct **Asn1CharArray**
- struct **Asn1CharSet**
- struct **Asn116BitCharSet**
- struct **_SListNode**
- struct **_SList**
- struct **_DListNode**
- struct **_DList**
- struct **_Asn1SizeCnst**
- struct **ASN1BUFFER**
- struct **ASN1BUFSAVE**
- struct **ASN1ErrLocn**
- struct **ASN1ErrInfo**
- struct **OOCTXT**

Defines

- #define **TV_UNIV** 0 /* universal */
- #define **TV_APPL** 1 /* application-wide */
- #define **TV_CTXT** 2 /* context-specific */
- #define **TV_PRIV** 3 /* private-use */
- #define **TV_PRIM** 0 /* primitive */
- #define **TV_CONS** 1 /* constructor */
- #define **TM_UNIV** 0x00000000 /* universal class */
- #define **TM_APPL** 0x40000000 /* application-wide class */
- #define **TM_CTXT** 0x80000000 /* context-specific class */
- #define **TM_PRIV** 0xC0000000 /* private-use class */
- #define **TM_PRIM** 0x00000000 /* primitive form */
- #define **TM_CONS** 0x20000000 /* constructor form */
- #define **TM_IDCODE** 0x1FFFFFFF /* ID code mask */
- #define **ASN_K_BADTAG** 0xFFFFFFFF /* invalid tag code */
- #define **ASN_K_NOTAG** 0xFFFFFFFF /* no tag input parameter */

- #define **TM_CLASS** 0xC0 /* class mask */
- #define **TM_FORM** 0x20 /* form mask */
- #define **TM_CLASS_FORM** 0xE0 /* class/form mask */
- #define **TM_B_IDCODE** 0x1F /* id code mask (byte) */
- #define **MINMSGLEN** 8 /* minimum message length */
- #define **ASN_OK** 0 /* normal completion status */
- #define **ASN_OK_FRAG** 2 /* message fragment detected */
- #define **ASN_E_BUFOVFLW** -1 /* encode buffer overflow */
- #define **ASN_E_ENDOFBUF** -2 /* unexpected end of buffer on decode */
- #define **ASN_E_IDNOTFOU** -3 /* identifer not found */
- #define **ASN_E_INVOBJID** -4 /* invalid object identifier */
- #define **ASN_E_INVLEN** -5 /* invalid field length */
- #define **ASN_E_INVENUM** -6 /* enumerated value not in defined set */
- #define **ASN_E_SETDUPL** -7 /* duplicate element in set */
- #define **ASN_E_SETMISRQ** -8 /* missing required element in set */
- #define **ASN_E_NOTINSET** -9 /* element not part of set */
- #define **ASN_E_SEQOVFLW** -10 /* sequence of field overflow */
- #define **ASN_E_INVOPT** -11 /* invalid option encountered in choice */
- #define **ASN_E_NOMEM** -12 /* no dynamic memory available */
- #define **ASN_E_INVHEXS** -14 /* invalid hex string */
- #define **ASN_E_INVBINS** -15 /* invalid binary string */
- #define **ASN_E_INVREAL** -16 /* invalid real value */
- #define **ASN_E_STROVFLW** -17 /* octet or bit string field overflow */
- #define **ASN_E_BADVALUE** -18 /* invalid value specification */
- #define **ASN_E_UNDEFVAL** -19 /* no def found for ref'd defined value */
- #define **ASN_E_UNDEFTYP** -20 /* no def found for ref'd defined type */
- #define **ASN_E_BADTAG** -21 /* invalid tag value */
- #define **ASN_E_TOODEEP** -22 /* nesting level is too deep */
- #define **ASN_E_CONSVIO** -23 /* value constraint violation */
- #define **ASN_E_RANGERR** -24 /* invalid range (lower > upper) */
- #define **ASN_E_ENDOFFILE** -25 /* end of file on file decode */
- #define **ASN_E_INVUTF8** -26 /* invalid UTF-8 encoding */
- #define **ASN_E_CONCMODF** -27 /* Concurrent list modification */
- #define **ASN_E_ILLSTATE** -28 /* Illegal state error */
- #define **ASN_E_OUTOFBND** -29 /* out of bounds (of array, etc) */
- #define **ASN_E_INVPARAM** -30 /* invalid parameter */
- #define **ASN_E_INVFORMAT** -31 /* invalid time string format */
- #define **ASN_E_NOTINIT** -32 /* not initialized */
- #define **ASN_E_TOOBIG** -33 /* value is too big for given data type */
- #define **ASN_E_INVCHAR** -34 /* invalid character (not in char set) */
- #define **ASN_E_XMLSTATE** -35 /* XML state error */
- #define **ASN_E_XMLPARSE** -36 /* XML parse error */
- #define **ASN_E_SEQORDER** -37 /* SEQUENCE elements not in order */
- #define **ASN_E_INVINDEX** -38 /* invalid index for TC id */
- #define **ASN_E_INVTCVAL** -39 /* invalid value for TC field */
- #define **ASN_E_FILNOTFOU** -40 /* file not found */
- #define **ASN_E_FILEREAD** -41 /* error occurred reading file */
- #define **ASN_E_FILEWRITE** -42 /* error occurred writing file */
- #define **ASN_E_INVBASE64** -43 /* invalid base64 encoding */
- #define **ASN_E_INVSOCKET** -44 /* invalid socket operation */
- #define **ASN_E_XMLLIBNFOU** -45 /* XML library is not found */
- #define **ASN_E_XMLLIBINV** -46 /* XML library is invalid */
- #define **ASN_E_NOTSUPP** -99 /* non-supported ASN construct */
- #define **ASN_K_INDEFLEN** -9999 /* indefinite length message indicator */
- #define **ASN_ID_EOC** 0 /* end of contents */

- #define **ASN_ID_BOOL** 1 /* boolean */
- #define **ASN_ID_INT** 2 /* integer */
- #define **ASN_ID_BITSTR** 3 /* bit string */
- #define **ASN_ID_OCTSTR** 4 /* byte (octet) string */
- #define **ASN_ID_NULL** 5 /* null */
- #define **ASN_ID_OBJID** 6 /* object ID */
- #define **ASN_ID_OBJDSC** 7 /* object descriptor */
- #define **ASN_ID_EXTERN** 8 /* external type */
- #define **ASN_ID_REAL** 9 /* real */
- #define **ASN_ID_ENUM** 10 /* enumerated value */
- #define **ASN_ID_EPDV** 11 /* EmbeddedPDV type */
- #define **ASN_ID_RELOID** 13 /* relative object ID */
- #define **ASN_ID_SEQ** 16 /* sequence, sequence of */
- #define **ASN_ID_SET** 17 /* set, set of */
- #define **ASN_SEQ_TAG** 0x30 /* SEQUENCE universal tag byte */
- #define **ASN_SET_TAG** 0x31 /* SET universal tag byte */
- #define **ASN_ID_NumericString** 18
- #define **ASN_ID_PrintableString** 19
- #define **ASN_ID_TeletexString** 20
- #define **ASN_ID_T61String** ASN_ID_TeletexString
- #define **ASN_ID_VideotexString** 21
- #define **ASN_ID_IA5String** 22
- #define **ASN_ID_UTCTime** 23
- #define **ASN_ID_GeneralTime** 24
- #define **ASN_ID_GraphicString** 25
- #define **ASN_ID_VisibleString** 26
- #define **ASN_ID_GeneralString** 27
- #define **ASN_ID_UniversalString** 28
- #define **ASN_ID_BMPString** 30
- #define **XM_SEEK** 0x01 /* seek match until found or end-of-buf */
- #define **XM_ADVANCE** 0x02 /* advance pointer to contents on match */
- #define **XM_DYNAMIC** 0x04 /* alloc dyn mem for decoded variable */
- #define **XM_SKIP** 0x08 /* skip to next field after parsing tag */
- #define **ASN_K_MAXDEPTH** 32 /* maximum nesting depth for messages */
- #define **ASN_K_MAXSUBIDS** 128 /* maximum sub-id's in an object ID */
- #define **ASN_K_MAXENUM** 100 /* maximum enum values in an enum type */
- #define **ASN_K_MAXERRP** 5 /* maximum error parameters */
- #define **ASN_K_MAXERRSTK** 8 /* maximum levels on error ctxt stack */
- #define **ASN_K_ENCBUFSIZ** 16*1024 /* dynamic encode buffer extent size */
- #define **ASN_K_MEMBUFSEG** 1024 /* memory buffer extent size */
- #define **NUM_ABITS** 4
- #define **NUM_UBITS** 4
- #define **NUM_CANSET** " 0123456789"
- #define **PRN_ABITS** 8
- #define **PRN_UBITS** 7
- #define **PRN_CANSET** "
  '()+,-./0123456789:=?ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
- #define **VIS_ABITS** 8
- #define **VIS_UBITS** 7
- #define **VIS_CANSET**
- #define **T61_ABITS** 8
- #define **T61_UBITS** 7
- #define **T61_CANSET**
- #define **IA5_ABITS** 8
- #define **IA5_UBITS** 7

- #define **IA5_CANSET**
- #define **IA5_RANGE1_LOWER** 0
- #define **IA5_RANGE2_LOWER** 0x5f
- #define **GEN_ABITS** 8
- #define **GEN_UBITS** 7
- #define **GEN_CANSET**
- #define **BMP_ABITS** 16
- #define **BMP_UBITS** 16
- #define **BMP_FIRST** 0
- #define **BMP_LAST** 0xffff
- #define **UCS_ABITS** 32
- #define **UCS_UBITS** 32
- #define **UCS_FIRST** 0
- #define **UCS_LAST** 0xffffffful
- #define **ASN1TAG_LSHIFT** 24
- #define **ASN1UINT_MAX** 4294967295U
- #define **ASN1INT_MAX** ((ASN1INT)2147483647L)
- #define **ASN1INT_MIN** ((ASN1INT)(-ASN1INT_MAX-1))
- #define **ASN1INT64** long
- #define **FALSE** 0
- #define **TRUE** 1
- #define **XM_K_MEMBLKSIZ** (4*1024)
- #define **ASN1DYNCTXT** 0x8000
- #define **ASN1INDEFLEN** 0x4000
- #define **ASN1TRACE** 0x2000
- #define **ASN1LASTEOC** 0x1000
- #define **ASN1FASTCOPY** 0x0800 /* turns on the "fast copy" mode */
- #define **ASN1CONSTAG** 0x0400 /* form of last parsed tag */
- #define **ASN1CANXER** 0x0200 /* canonical XER */
- #define **ASN1SAVEBUF** 0x0100 /* do not free dynamic encode buffer */
- #define **ASN1OPENTYPE** 0x0080 /* item is an open type field */
- #define **ASN1MAX**(a, b) (((a)>(b))?(a):(b))
- #define **ASN1MIN**(a, b) (((a)<(b))?(a):(b))
- #define **ALLOC_ASN1ARRAY**(pctxt, pseqof, type)

Allocate a dynamic array.

- #define **ALLOC_ASN1ELEM**(pctxt, type) (type*) memHeapAllocZ (&(pctxt)->pTypeMemHeap, sizeof(type))

Allocate and zero an ASN.1 element.

- #define **ALLOC_ASN1ELEMDNODE**(pctxt, type)
- #define **ASN1MALLOC**(pctxt, nbytes) memHeapAlloc(&(pctxt)->pTypeMemHeap, nbytes)

Allocate memory.

- #define **ASN1MEMFREE**(pctxt) memHeapFreeAll(&(pctxt)->pTypeMemHeap)

Free memory associated with a context.

- #define **ASN1MEMFREEPTR**(pctxt, pmem) memHeapFreePtr(&(pctxt)->pTypeMemHeap, (void*)pmem)

Free memory pointer.

- #define **ASN1BUFCUR**(cp) (cp)->buffer.data[(cp)->buffer.byteIndex]
- #define **ASN1BUFPTR**(cp) &(cp)->buffer.data[(cp)->buffer.byteIndex]
- #define **EXTERN**
- #define **ASN1CRTMALLOC0**(nbytes) malloc(nbytes)
- #define **ASN1CRTFREE0**(ptr) free(ptr)
- #define **ASN1CRTMALLOC** memHeapAlloc
- #define **ASN1CRTFREE** ASN1MEMFREEPTR
- #define **DE_INCRBITIDX**(pctxt)
- #define **DE_BIT**(pctxt, pvalue)
- #define **encodeIA5String**(pctxt, value, permCharSet) encodeConstrainedStringEx(pctxt, value, permCharSet, 8, 7, 7)
- #define **encodeGeneralizedTime** encodeIA5String
- #define **decodeIA5String**(pctxt, pvalue, permCharSet) decodeConstrainedStringEx(pctxt, pvalue, permCharSet, 8, 7, 7)
- #define **decodeGeneralizedTime** decodeIA5String
- #define **ZEROCONTEXT**(pctxt) memset(pctxt,0,sizeof(OOCTXT))
- #define **LOG_ASN1ERR**(ctxt, stat) errSetData(&(ctxt)->errInfo,stat,__FILE__,__LINE__)
- #define **LOG_ASN1ERR_AND_FREE**(pctxt, stat, lctxt) freeContext ((lctxt)), LOG_ASN1ERR(pctxt, stat)
- #define **RT_MH_DONTKEEPFREE** 0x1
- #define **OSRTMH_PROPID_DEFBLKSIZE** 1
- #define **OSRTMH_PROPID_SETFLAGS** 2
- #define **OSRTMH_PROPID_CLEARFLAGS** 3
- #define **OSRTMH_PROPID_USER** 10
- #define **memAlloc**(pctxt, nbytes) memHeapAlloc(&(pctxt)->pTypeMemHeap,nbytes)

Allocate memory.

- #define **memAllocZ**(pctxt, nbytes) memHeapAllocZ(&(pctxt)->pTypeMemHeap,nbytes)

Allocate and zero memory.

- #define **memRealloc**(pctxt, mem_p, nbytes) memHeapRealloc(&(pctxt)->pTypeMemHeap, (void*)mem_p, nbytes)

Reallocate memory.

- #define **memFreePtr**(pctxt, mem_p)

Free memory pointer.

- #define **memFree**(pctxt) memHeapFreeAll(&(pctxt)->pTypeMemHeap)

Free memory associated with a context.

- #define **memReset**(pctxt) memHeapReset(&(pctxt)->pTypeMemHeap)

Reset memory associated with a context.

- #define **OSCDECL**
- #define **INCRBITIDX**(pctxt)
- #define **DECODEBIT**(pctxt, pvalue)
- #define **decodeUnconsInteger**(pctxt, pvalue) decodeSemiConsInteger(pctxt, pvalue, ASN1INT_MIN)

This function will decode an unconstrained integer.

- #define **decodeUnconsUnsigned**(pctxt, pvalue) decodeSemiConsUnsigned(pctxt, pvalue, 0U)

This function will decode an unconstrained unsigned integer.

- #define **encodeUnconsInteger**(pctxt, value) encodeSemiConsInteger(pctxt,value,ASN1INT_MIN)

This function encodes an unconstrained integer.

Typedefs

- typedef char **ASN1CHAR**
- typedef unsigned char **ASN1OCTET**
- typedef ASN1OCTET **ASN1BOOL**
- typedef signed char **ASN1INT8**
- typedef unsigned char **ASN1UINT8**
- typedef int **ASN1INT**
- typedef unsigned int **ASN1UINT**
- typedef ASN1INT **ASN1ENUM**
- typedef double **ASN1REAL**
- typedef short **ASN1SINT**
- typedef unsigned short **ASN1USINT**
- typedef ASN1UINT **ASN1TAG**
- typedef ASN1USINT **ASN116BITCHAR**
- typedef ASN1UINT **ASN132BITCHAR**
- typedef void * **ASN1ANY**
- typedef const char * **ASN1GeneralizedTime**
- typedef const char * **ASN1GeneralString**
- typedef const char * **ASN1GraphicString**
- typedef const char * **ASN1IA5String**
- typedef const char * **ASN1ISO646String**
- typedef const char * **ASN1NumericString**
- typedef const char * **ASN1ObjectDescriptor**
- typedef const char * **ASN1PrintableString**
- typedef const char * **ASN1TeletexString**
- typedef const char * **ASN1T61String**
- typedef const char * **ASN1UTCTime**
- typedef const char * **ASN1UTF8String**
- typedef const char * **ASN1VideotexString**
- typedef const char * **ASN1VisibleString**
- typedef Asn116BitCharString **ASN1BMPString**
- typedef Asn132BitCharString **ASN1UniversalString**
- typedef _SListNode **SListNode**
- typedef _SList **SList**
- typedef _DListNode **DListNode**
- typedef _DList **DList**
- typedef _Asn1SizeCnst **Asn1SizeCnst**
- typedef OOCTXT **OOCTXT**
- typedef void *OSCDECL * **OSMallocFunc** (size_t size)
- typedef void *OSCDECL * **OSReallocFunc** (void *ptr, size_t size)

Functions

- EXTERN int **initContextBuffer** (OOCTXT *pctxt, const ASN1OCTET *bufaddr, ASN1UINT bufsiz)

This function assigns a buffer to a context block.

- EXTERN int **initContext** (OOCTXT *pctxt)

This function initializes a context block.

- EXTERN void **freeContext** (OOCTXT *pctxt)

This function frees all dynamic memory associated with a context.

- EXTERN OOCTXT * **newContext** ()

This function allocates a new OOCTXT block and initializes it.

- EXTERN void **copyContext** (OOCTXT *pdest, OOCTXT *psrc)
- EXTERN int **initSubContext** (OOCTXT *pctxt, OOCTXT *psrc)
- EXTERN void **setCtxtFlag** (OOCTXT *pctxt, ASN1USINT mask)
- EXTERN void **clearCtxtFlag** (OOCTXT *pctxt, ASN1USINT mask)
- EXTERN int **setPERBuffer** (OOCTXT *pctxt, ASN1OCTET *bufaddr, ASN1UINT bufsiz, ASN1BOOL aligned)
- EXTERN int **setPERBufferUsingCtxt** (OOCTXT *pTarget, OOCTXT *pSource)
- EXTERN DListNode * **dListAppend** (OOCTXT *pctxt, DList *pList, void *pData)

This function appends an item to the linked list structure.

- EXTERN DListNode * **dListAppendNode** (OOCTXT *pctxt, DList *pList, void *pData)
- EXTERN DListNode * **dListFindByIndex** (DList *pList, int index)
- EXTERN void **dListInit** (DList *pList)

This function initializes a doubly linked list structure.

- EXTERN void **dListFreeNodes** (OOCTXT *pctxt, DList *pList)

This function removes all nodes from the linked list and releases the memory that was allocated for storing the node structures (DListNode).

- EXTERN void **dListFreeAll** (OOCTXT *pctxt, DList *pList)

This function removes all nodes from the linked list structure and releases the memory that was allocated for storing the node structures (DListNode) and for data.

- EXTERN void **dListRemove** (DList *pList, DListNode *node)

This function removes a node from the linked list structure.

- EXTERN void **sListInit** (SList *pList)

This function is used to initialize a singly-linked list.

- EXTERN void **sListInitEx** (OOCTXT *pctxt, SList *pList)

This function is used to initialize a singly-linked list and assigns a context to be used for the list.

- EXTERN void **sListFree** (SList *pList)

This function is used to free-up all the nodes in the singly-linked list.

- EXTERN SList * **sListCreate** ()

This function is used to create a new singly-linked list.

- EXTERN SList * **sListCreateEx** (OOCTXT *pctxt)

This function is used to create a singly-linked list.

- EXTERN SListNode * **sListAppend** (SList *pList, void *pData)

This function is used to append a new data member to the list.

- EXTERN ASN1BOOL **sListFind** (SList *pList, void *pData)

This function is used to search for a particular data in the list.

- EXTERN void **sListRemove** (SList *pList, void *pData)

This function is used to remove a particular data member from the list.

- EXTERN int **errAddIntParm** (ASN1ErrInfo *pErrInfo, int errParm)

This function adds an integer parameter to an error information structure.

- EXTERN int **errAddStrParm** (ASN1ErrInfo *pErrInfo, const char *errprm_p)

This function adds an string parameter to an error information structure.

- EXTERN int **errAddUIntParm** (ASN1ErrInfo *pErrInfo, unsigned int errParm)

This function adds an unsigned integer parameter to an error information structure.

- EXTERN int **errCopyData** (ASN1ErrInfo *pSrcErrInfo, ASN1ErrInfo *pDestErrInfo)
- EXTERN void **errFreeParms** (ASN1ErrInfo *pErrInfo)

This function frees memory associated with the storage of parameters associated with an error message.

- EXTERN char * **errFmtMsg** (ASN1ErrInfo *pErrInfo, char *bufp)
- EXTERN char * **errGetText** (OOCTXT *pctxt)

This function gets the text of the error.

- EXTERN void **errPrint** (ASN1ErrInfo *pErrInfo)

This function prints error information to the standard output device.

- EXTERN int **errReset** (ASN1ErrInfo *pErrInfo)

This function resets the error information in the error information sturcture.

- EXTERN int **errSetData** (ASN1ErrInfo *pErrInfo, int status, const char *module, int lno)

This function sets error information in an error information structure.

- typedef **void** (OSCDECL *OSFreeFunc)(void *ptr)
- EXTERN void **memHeapAddRef** (void **ppvMemHeap)
- EXTERN void * **memHeapAlloc** (void **ppvMemHeap, int nbytes)

- EXTERN void * **memHeapAllocZ** (void **ppvMemHeap, int nbytes)
- EXTERN int **memHeapCheckPtr** (void **ppvMemHeap, void *mem_p)
- EXTERN int **memHeapCreate** (void **ppvMemHeap)
- EXTERN void **memHeapFreeAll** (void **ppvMemHeap)
- EXTERN void **memHeapFreePtr** (void **ppvMemHeap, void *mem_p)
- EXTERN void * **memHeapRealloc** (void **ppvMemHeap, void *mem_p, int nbytes_)
- EXTERN void **memHeapRelease** (void **ppvMemHeap)
- EXTERN void **memHeapReset** (void **ppvMemHeap)
- EXTERN void * **memHeapMarkSaved** (void **ppvMemHeap, const void *mem_p, ASN1BOOL saved)
- EXTERN void **memHeapSetProperty** (void **ppvMemHeap, ASN1UINT propId, void *pProp)
- EXTERN void **memSetAllocFuncs** (OSMallocFunc malloc_func, OSReallocFunc realloc_func, OSFreeFunc free_func)

This function sets the pointers to standard allocation functions.

- EXTERN void **memFreeOpenSeqExt** (OOCTXT *pctxt, DList *pElemList)
- EXTERN void **memHeapSetFlags** (OOCTXT *pctxt, ASN1UINT flags)
- EXTERN void **memHeapClearFlags** (OOCTXT *pctxt, ASN1UINT flags)
- EXTERN void **memHeapSetDefBlkSize** (OOCTXT *pctxt, ASN1UINT blkSize)

This function sets the pointer to standard allocation functions.

- EXTERN ASN1UINT **memHeapGetDefBlkSize** (OOCTXT *pctxt)

This function returns the actual granularity of memory blocks.

- EXTERN int **decodeBits** (OOCTXT *pctxt, ASN1UINT *pvalue, ASN1UINT nbits)

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

- EXTERN int **decodeBitString** (OOCTXT *pctxt, ASN1UINT *numbits_p, ASN1OCTET *buffer, ASN1UINT bufsiz)

This function will decode a value of the ASN.1 bit string type whose maximum size is is known in advance.

- EXTERN int **decodeBMPString** (OOCTXT *pctxt, ASN1BMPString *pvalue, Asn116BitCharSet *permCharSet)

This function will decode a variable of the ASN.1 BMP character string.

- EXTERN int **decodeByteAlign** (OOCTXT *pctxt)

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

- EXTERN int **decodeConsInteger** (OOCTXT *pctxt, ASN1INT *pvalue, ASN1INT lower, ASN1INT upper)

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

- EXTERN int **decodeConsUnsigned** (OOCTXT *pctxt, ASN1UINT *pvalue, ASN1UINT lower, ASN1UINT upper)

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

- EXTERN int **decodeConsUInt8** (OOCTXT *pctxt, ASN1UINT8 *pvalue, ASN1UINT lower, ASN1UINT upper)

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

- EXTERN int **decodeConsUInt16** (OOCTXT *pctxt, ASN1USINT *pvalue, ASN1UINT lower, ASN1UINT upper)

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

- EXTERN int **decodeConsWholeNumber** (OOCTXT *pctxt, ASN1UINT *padjusted_value, ASN1UINT range_value)

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

- EXTERN int **decodeConstrainedStringEx** (OOCTXT *pctxt, const char **string, const char *charSet, ASN1UINT abits, ASN1UINT ubits, ASN1UINT canSetBits)

This function decodes a constrained string value.

- EXTERN int **decodeDynBitString** (OOCTXT *pctxt, ASN1DynBitStr *pBitStr)

This function will decode a variable of thr ASN.1 BIT STRING type.

- EXTERN int **decodeDynOctetString** (OOCTXT *pctxt, ASN1DynOctStr *pOctStr)

This function will decode a value of the ASN.1 octet string type whose maximum size is known in advance.

- EXTERN int **decodeLength** (OOCTXT *pctxt, ASN1UINT *pvalue)

This function will decode a length determinant value.

- EXTERN int **moveBitCursor** (OOCTXT *pctxt, int bitOffset)
- EXTERN int **decodeObjectIdentifier** (OOCTXT *pctxt, ASN1OBJID *pvalue)

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

- EXTERN int **decodeOctetString** (OOCTXT *pctxt, ASN1UINT *numocts_p, ASN1OCTET *buffer, ASN1UINT bufsiz)

This function will decode a value of the ASN.1 octet string type whose maximun size is known in advance.

- EXTERN int **decodeOpenType** (OOCTXT *pctxt, const ASN1OCTET **object_p2, ASN1UINT *numocts_p)

This function will decode an ASN.1 open type.

- EXTERN int **decodeSmallNonNegWholeNumber** (OOCTXT *pctxt, ASN1UINT *pvalue)

This function will decode a small non-negative whole number as specified in Section 10.6 of the X.691 standard.

- EXTERN int **decodeSemiConsInteger** (OOCTXT *pctxt, ASN1INT *pvalue, ASN1INT lower)

This function will decode a semi-constrained integer.

- EXTERN int **decodeSemiConsUnsigned** (OOCTXT *pctxt, ASN1UINT *pvalue, ASN1UINT lower)

This function will decode a semi-constrained unsigned integer.

- EXTERN int **decodeVarWidthCharString** (OOCTXT *pctxt, const char **pvalue)
- EXTERN int **encodeBit** (OOCTXT *pctxt, ASN1BOOL value)

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

- EXTERN int **encodeBits** (OOCTXT *pctxt, ASN1UINT value, ASN1UINT nbits)

This function encodes multiple bits.

- EXTERN int **encodeBitString** (OOCTXT *pctxt, ASN1UINT numocts, const ASN1OCTET *data)

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

- EXTERN int **encodeBMPString** (OOCTXT *pctxt, ASN1BMPString value, Asn116BitCharSet *permCharSet)

This function will encode a variable of the ASN.1 BMP character string.

- EXTERN int **encodeByteAlign** (OOCTXT *pctxt)

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

- EXTERN int **encodeCheckBuffer** (OOCTXT *pctxt, ASN1UINT nbytes)

This function will determine if the given number of bytes will fit in the encode buffer.

- EXTERN int **encodeConstrainedStringEx** (OOCTXT *pctxt, const char *string, const char *charSet, ASN1UINT abits, ASN1UINT ubits, ASN1UINT canSetBits)

This function encodes a constrained string value.

- EXTERN int **encodeConsInteger** (OOCTXT *pctxt, ASN1INT value, ASN1INT lower, ASN1INT upper)

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

- EXTERN int **encodeConsUnsigned** (OOCTXT *pctxt, ASN1UINT value, ASN1UINT lower, ASN1UINT upper)

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

- EXTERN int **encodeConsWholeNumber** (OOCTXT *pctxt, ASN1UINT adjusted_value, ASN1UINT range_value)

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

- EXTERN int **encodeExpandBuffer** (OOCTXT *pctxt, ASN1UINT nbytes)

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

- EXTERN ASN1OCTET * **encodeGetMsgPtr** (OOCTXT *pctxt, int *pLength)

This function will return the message pointer and length of an encoded message.

- EXTERN int **encodeLength** (OOCTXT *pctxt, ASN1UINT value)

This function will encode a length determinant value.

- EXTERN int **encodeObjectIdentifier** (OOCTXT *pctxt, ASN1OBJID *pvalue)

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

- EXTERN int **encodebitsFromOctet** (OOCTXT *pctxt, ASN1OCTET value, ASN1UINT nbits)

This function encodes bits from a given octet to the output buffer.

- EXTERN int **encodeOctets** (OOCTXT *pctxt, const ASN1OCTET *pvalue, ASN1UINT nbits)

This fuction will encode an array of octets.

- EXTERN int **encodeOctetString** (OOCTXT *pctxt, ASN1UINT numocts, const ASN1OCTET *data)

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

- EXTERN int **encodeOpenType** (OOCTXT *pctxt, ASN1UINT numocts, const ASN1OCTET *data)

This function will encode an ASN.1 open type.

- EXTERN int **encodeOpenTypeExt** (OOCTXT *pctxt, DList *pElemList)

This function will encode an ASN.1 open type extension.

- EXTERN int **encodeOpenTypeExtBits** (OOCTXT *pctxt, DList *pElemList)
- EXTERN int **encodeSmallNonNegWholeNumber** (OOCTXT *pctxt, ASN1UINT value)

This function will endcode a small, non-negative whole number as specified in Section 10.6 of teh X.691 standard.

- EXTERN int **encodeSemiConsInteger** (OOCTXT *pctxt, ASN1INT value, ASN1INT lower)

This function encodes a semi-constrained integer.

- EXTERN int **encodeSemiConsUnsigned** (OOCTXT *pctxt, ASN1UINT value, ASN1UINT lower)

This function encodes an semi-constrained unsigned integer.

- EXTERN int **encodeVarWidthCharString** (OOCTXT *pctxt, const char *value)
- EXTERN int **addSizeConstraint** (OOCTXT *pctxt, Asn1SizeCnst *pSize)
- EXTERN ASN1BOOL **alignCharStr** (OOCTXT *pctxt, ASN1UINT len, ASN1UINT nbits, Asn1SizeCnst *pSize)
- EXTERN int **bitAndOctetStringAlignmentTest** (Asn1SizeCnst *pSizeList, ASN1UINT itemCount, ASN1BOOL bitStrFlag, ASN1BOOL *pAlignFlag)
- EXTERN int **getPERMsgLen** (OOCTXT *pctxt)
- EXTERN Asn1SizeCnst * **getSizeConstraint** (OOCTXT *pctxt, ASN1BOOL extbit)

- EXTERN int **checkSizeConstraint** (OOCTXT *pctxt, int size)
- EXTERN ASN1UINT **getUIntBitCount** (ASN1UINT value)
- EXTERN Asn1SizeCnst * **checkSize** (Asn1SizeCnst *pSizeList, ASN1UINT value, ASN1BOOL *pExtendable)
- EXTERN void **init16BitCharSet** (Asn116BitCharSet *pCharSet, ASN116BITCHAR first, ASN116BITCHAR last, ASN1UINT abits, ASN1UINT ubits)
- EXTERN ASN1BOOL **isExtendableSize** (Asn1SizeCnst *pSizeList)
- EXTERN void **set16BitCharSet** (OOCTXT *pctxt, Asn116BitCharSet *pCharSet, Asn116BitCharSet *pAlphabet)
- EXTERN const char * **rtBitStrToString** (ASN1UINT numbits, const ASN1OCTET *data, char *buffer, size_t bufsiz)
- EXTERN const char * **rtOctStrToString** (ASN1UINT numocts, const ASN1OCTET *data, char *buffer, size_t bufsiz)

Detailed Description

Common ASN.1 runtime constants, data structure definitions, and run-time functions to support BER/DER/PER as defined in the ITU-T standards.

Definition in file **ooasn1.h**.

ooCalls.h File Reference

This file contains Call management functions.

```
#include "ootypes.h"
```

Functions

- EXTERN **ooCallData** * **ooCreateCall** (char *type, char *callToken)

This function is used to create a new call entry.

- EXTERN int **ooAddCallToList** (**ooEndPoint** *h323ep, **ooCallData** *call)

This function is used to add a call to the list of existing calls.

- EXTERN **ooCallData** * **ooFindCallByToken** (char *callToken)

This function is used to find a call by using the unique token for the call.

- EXTERN int **ooEndCall** (**ooCallData** *call)

This function is used to clear a call.

- EXTERN int **ooRemoveCallFromList** (**ooEndPoint** *h323ep, **ooCallData** *call)

This function is used to remove a call from the list of existing calls.

- EXTERN int **ooCleanCall** (**ooCallData** *call)

This function is used to clean a call.

- EXTERN **ooLogicalChannel** * **ooAddNewLogicalChannel** (**ooCallData** *call, int channelNo, int sessionID, char *type, char *dir, **ooH323EpCapability** *epCap)

This function is used to add a new logical channel entry into the list of currently active logical channels.

- EXTERN **ooLogicalChannel** * **ooFindLogicalChannelByLogicalChannelNo** (**ooCallData** *call, int channelNo)

This function is used to find a logical channel by logical channel number.

- EXTERN int **ooOnLogicalChannelEstablished** (**ooCallData** *call, **ooLogicalChannel** *pChannel)

This function is called when a new logical channel is established.

- EXTERN ASN1BOOL **ooIsSessionEstablished** (**ooCallData** *call, int sessionID, char *dir)

This fuction is used to check whether a specified session in specified direction is active for the call.

- EXTERN **ooLogicalChannel** * **ooGetLogicalChannel** (**ooCallData** *call, int sessionID)

This function is used to retrieve a logical channel with particular sessionID.

- EXTERN int **ooRemoveLogicalChannel** (**ooCallData** *call, int ChannelNo)

This function is used to remove a logical channel from the list of logical channels.

- EXTERN int **ooClearLogicalChannel** (**ooCallData** *call, int channelNo)

This function is used to cleanup a logical channel.

- EXTERN int **ooClearAllLogicalChannels** (**ooCallData** *call)

This function is used to cleanup all the logical channels associated with the call.

- EXTERN int **ooAddMediaInfo** (**ooCallData** *call, ooMediaInfo mediaInfo)

This function can be used by an application to specify media endpoint information for different types of media.

Detailed Description

This file contains Call management functions.

Definition in file **ooCalls.h**.

ooCapability.h File Reference

This file contains Capability management functions.

```
#include "ootypes.h"
#include "ooasn1.h"
```

Functions

- EXTERN int **ooAddAudioCapability** (H245AudioCapability audioCap, int dir, **cb_StartReceiveChannel** startReceiveChannel, **cb_StartTransmitChannel** startTransmitChannel, **cb_StopReceiveChannel** stopReceiveChannel, **cb_StopTransmitChannel** stopTransmitChannel)

Function to add audio capabilities to the endpoint.

- EXTERN H245AudioCapability * **ooCreateDupAudioCap** (**ooCallData** *call, H245AudioCapability *audioCap, OOCTXT *pctxt)

This function is used to create a duplicate audio capability from the one supplied as parameter.

- EXTERN H245GSMAudioCapability * **ooCreateDupGSMAudioCap** (**ooCallData** *call, H245GSMAudioCapability *gsmCap, OOCTXT *pctxt)

This function is used to create a duplicate GSM audio capability.

- EXTERN **ooH323EpCapability** * **ooIsAudioCapSupported** (**ooCallData** *call, H245AudioCapability *audioCap, int dir)

This function is used to check whether local endpoint supports a particular type of audio capability.

- EXTERN int **ooCompareAudioCaps** (**ooCallData** *call, H245AudioCapability *cap1, H245AudioCapability *cap2)

This function is used to compare two audio capabilities for a match.

- EXTERN int **ooCompareG711Caps** (H245AudioCapability *cap1, H245AudioCapability *cap2)

This function is used to compare two G711 audio capabilities for a match.

- EXTERN int **ooCompareGSMCaps** (H245AudioCapability *cap1, H245AudioCapability *cap2)

This function is used to compare two GSM audio capabilities for a match.

- EXTERN **ooH323EpCapability** * **ooIsDataTypeSupported** (**ooCallData** *call, H245DataType *data, int dir)

Checks whether a particular datatype is supported by the end point.

- EXTERN **ooH323EpCapability** * **ooIsAudioDataTypeSupported** (**ooCallData** *call, H245AudioCapability *audioData, int dir)

Checks whether a particular audio datatype is supported by the end point.

- EXTERN **ooH323EpCapability** * **ooIsNonStandardDataTypeSupported** (**ooCallData** *call, H245NonStandardParameter *nonStandard, int dir)

- EXTERN **ooH323EpCapability * ooIsVideoDataTypeSupported** (**ooCallData** *call, H245VideoCapability *videoData, int dir)
- EXTERN **ooH323EpCapability * ooIsApplicationDataTypeSupported** (**ooCallData** *call, H245DataApplicationCapability *data, int dir)
- EXTERN **ooH323EpCapability * ooIsEncryptedDataTypeSupported** (**ooCallData** *call, H245EncryptionMode *encryptionData, int dir)
- EXTERN **ooH323EpCapability * ooIsH235ControlDataTypeSupported** (**ooCallData** *call, H245NonStandardParameter *h235Control, int dir)
- EXTERN **ooH323EpCapability * ooIsH235MediaDataTypeSupported** (**ooCallData** *call, H245H235Media *h235Media, int dir)
- EXTERN **ooH323EpCapability * ooIsMultiplexedStreamDataTypeSupported** (**ooCallData** *call, H245MultiplexedStreamParameter *multiplexedStream, int dir)

Detailed Description

This file contains Capability management functions.

Definition in file **ooCapability.h**.

oochannels.h File Reference

This file contains functions to create and use channels.

```
#include "H323-MESSAGES.h"
#include "MULTIMEDIA-SYSTEM-CONTROL.h"
#include "ootypes.h"
#include "ooSocket.h"
```

Defines

- #define **OORECEIVER** 1
- #define **OOTRANSMITTER** 2
- #define **OODUPLEX** 3

Functions

- EXTERN int **ooCreateH323Listener** ()

This function is used to create a listener for incoming calls.

- EXTERN int **ooCreateH245Listener** (**ooCallData** *call)

This function is used to create a listener for incoming H.245 connections.

- EXTERN int **ooAcceptH225Connection** ()

This function is used to accept incoming H.225 connections.

- EXTERN int **ooAcceptH245Connection** (**ooCallData** *call)

This function is used to accept an incoming H.245 connection.

- EXTERN int **ooCreateH225Connection** (**ooCallData** *call)

This function is used to create an H.225 connection to the remote end point.

- EXTERN int **ooCreateH245Connection** (**ooCallData** *call)

This function is used to setup an H.245 connection with the remote endpoint for control negotiations.

- EXTERN int **ooCloseH225Connection** (**ooCallData** *call)

This function is used to close an H.225 connection.

- EXTERN int **ooCloseH245Connection** (**ooCallData** *call)

This function is used to close an H.245 connection for a call.

- EXTERN int **ooMonitorChannels** ()

This function is used to start monitoring channels for the calls.

- EXTERN int **ooStopMonitorCalls** ()

This function is called to stop the monitor channels thread.

- EXTERN int **ooH2250Receive** (**ooCallData** *call)

This function is used to receive an H.2250 message received on a calls H.225 channel.

- EXTERN int **ooH245Receive** (**ooCallData** *call)

This function is used to receive an H.245 message received on a calls H.245 channel.

- EXTERN int **ooSendH225Msg** (**ooCallData** *call, **Q931Message** *msg)

This function is used to enqueue an H.225 message into an outgoing queue for the call.

- EXTERN int **ooSendH245Msg** (**ooCallData** *call, **H245Message** *msg)

This function is used to enqueue an H.245 message into an outgoing queue for the call.

- EXTERN int **ooSendMsg** (**ooCallData** *call, int type)

This function is used to Send a message on the channel, when channel is available for write.

- EXTERN int **ooOnSendMsg** (**ooCallData** *call, int msgType)

This function is called after a message is sent on the call's channel.

Detailed Description

This file contains functions to create and use channels.

Definition in file **oochannels.h**.

ooh245.h File Reference

This file contains functions to support H245 negotiations.

```
#include "ooasn1.h"
#include "ootypes.h"
#include "oochannels.h"
#include "oo.h"
#include "oosndrtp.h"
#include "ooq931.h"
#include "MULTIMEDIA-SYSTEM-CONTROL.h"
```

Functions

- EXTERN int **ooCreateH245Message** (**H245Message** **msg, int type)

Creates an outgoing H245 message of the type specified by the type argument for the Application context.

- EXTERN int **ooFreeH245Message** (**H245Message** *pmsg)

Frees up the memory used by the H245 message.

- EXTERN int **ooGetOutgoingH245Msgbuf** (**ooCallData** *call, ASN1OCTET *msgbuf, int *len, int *msgType)

This function is used to retrieve an H.245 message enqueued in the outgoing queue.

- EXTERN int **ooSendTermCapMsg** (**ooCallData** *call)

This function is used to send out a treminal capability set message.

- EXTERN ASN1UINT **ooGenerateStatusDeterminationNumber** ()

This function is used to generate a random status determination number for MSD procedure.

- EXTERN int **ooHandleMasterSlave** (**ooCallData** *call, void *pmsg, int msgType)

This fuction is used to handle received MasterSlaveDetermination procedure messages.

- EXTERN int **ooSendMasterSlaveDetermination** (**ooCallData** *call)

This function is used to send MSD message.

- EXTERN int **ooSendMasterSlaveDeterminationAck** (**ooCallData** *call, char *status)

This function is used to send a MasterSlaveDeterminationAck message.

- EXTERN int **ooHandleOpenLogicalChannel** (**ooCallData** *call, H245OpenLogicalChannel *olc)

This function is used to handle received OpenLogicalChannel message.

- EXTERN int **ooHandleOpenLogicalAudioChannel** (**ooCallData** *call, H245OpenLogicalChannel *olc)

This function is used to handle a received OpenLogicalChannel message which is trying to open a audio channel.

- EXTERN int **ooOnReceivedOpenLogicalChannelAck** (**ooCallData** *call, H245OpenLogicalChannelAck *olcAck)

This function is used to handle a received OpenLogicalChannelAck message.

- EXTERN int **ooSendEndSessionCommand** (**ooCallData** *call)

This message is used to send an EndSession command.

- EXTERN int **ooHandleH245Command** (**ooCallData** *call, H245CommandMessage *command)

This function is used to handle a received H245Command message.

- EXTERN int **ooOnReceivedTerminalCapabilitySetAck** (**ooCallData** *call)

This function is called on receiving a TreminalCapabilitySetAck message.

- EXTERN int **ooCloseAllLogicalChannels** (**ooCallData** *call)

This function is called to close all the open logical channels.

- EXTERN int **ooSendCloseLogicalChannel** (**ooCallData** *call, **ooLogicalChannel** *logicalChan)

This function is used to send out a CloseLogicalChannel message for a particular logical channel.

- EXTERN int **ooOnReceivedCloseLogicalChannel** (**ooCallData** *call, H245CloseLogicalChannel *clc)

This function is used to process a received closeLogicalChannel request.

- EXTERN int **ooOnReceivedCloseChannelAck** (**ooCallData** *call, H245CloseLogicalChannelAck *clcAck)

This function is used to process a received CloseLogicalChannelAck message.

- EXTERN int **ooHandleH245Message** (**ooCallData** *call, **H245Message** *pmsg)

This function is used to handle received H245 message.

- EXTERN int **ooOnReceivedTerminalCapabilitySet** (**ooCallData** *call, **H245Message** *pmsg)

This function is used to process received TCS message.

- EXTERN int **ooH245AcknowledgeTerminalCapabilitySet** (**ooCallData** *call)

This function is used to send a TCSAck message to remote endpoint.

- EXTERN int **ooOpenLogicalChannels** (**ooCallData** *call)

This function is used to start OpenLogicalChannel procedure for all the channels to be opened for the call.

- EXTERN int **ooOpenLogicalAudioChannel** (**ooCallData** *call)

This function is used to send OpenLogicalChannel message for audio channel.

- EXTERN int **ooOpenG711ULaw64KChannel** (**ooCallData** *call, **ooH323EpCapability** *epCap)

This function is used to build a OpenLogicalChannel message using G711ULaw64K capability.

- EXTERN int **ooSendRequestCloseLogicalChannel** (**ooCallData** *call, **ooLogicalChannel** *logicalChan)

This function is used to request a remote end point to close a logical channel.

- EXTERN int **ooOnReceivedRequestChannelClose** (**ooCallData** *call, H245RequestChannelClose *rclc)

This function handles the received RequestChannelClose message, verifies that the requested channel is forward channel.

- EXTERN int **ooBuildOpenLogicalChannelAudio** (**ooCallData** *call, H245OpenLogicalChannel *olc, **ooH323EpCapability** *epCap, OOCTXT *pctxt)

Builds an OLC with an audio capability passed as parameter.

- EXTERN int **ooSendAsTunneledMessage** (**ooCallData** *call, ASN1OCTET *msgbuf, int len, int msgType)

This function sends an encoded H.245 message buffer as a tunneled H.245 message.

Detailed Description

This file contains functions to support H245 negotiations.

Definition in file **ooh245.h**.

ooh323.h File Reference

This file contains functions to support H.225 messages.

```
#include "ooasn1.h"
#include "ootypes.h"
#include "oo.h"
#include "ooq931.h"
#include "MULTIMEDIA-SYSTEM-CONTROL.h"
```

Functions

- EXTERN int **ooOnReceivedSetup** (**ooCallData** *call, **Q931Message** *q931Msg)

This function is used to process a received SETUP message.

- EXTERN int **ooOnReceivedSignalConnect** (**ooCallData** *call, **Q931Message** *q931Msg)

This function is used to process a received CONNECT message.

- EXTERN int **ooHandleH2250Message** (**ooCallData** *call, **Q931Message** *q931Msg)

This function is used to handle received H.2250 messages.

- EXTERN int **ooOnReceivedFacility** (**ooCallData** *call, **Q931Message** *pQ931Msg)

This function is used to process a received Facility message.

- EXTERN int **ooHandleTunneledH245Messages** (**ooCallData** *call, H225H323_UU_PDU *pH323UUPdu)

This function is used to process tunneled H245 messages.

- EXTERN int **ooHandleStartH245FacilityMessage** (**ooCallData** *call, H225Facility_UUIE *facility)

This is a helper function used to handle an startH245 Facility message.

- EXTERN int **ooRetrieveAliases** (**ooCallData** *call, H225_SeqOfH225AliasAddress *pAddresses, int remote)

This function is used to retrieve the aliases from Sequence of alias addresses.

Detailed Description

This file contains functions to support H.225 messages.

Definition in file **ooh323.h**.

ooh323ep.h File Reference

This file contains H323 endpoint related functions.

```
#include "ootypes.h"
#include "ooasn1.h"
```

Functions

- EXTERN int **ooInitializeH323Ep** (const char *tracefile, int h245Tunneling, int fastStart, int termType, int t35CountryCode, int t35Extension, int manufacturer, char *productID, char *versionID, int callType, int listenport, char *callerid, char *callername, int callMode)

This function is the first function to be invoked before using stack.

- EXTERN int **ooH323EpRegisterCallbacks** (cb_OnIncomingCall onIncomingCall, cb_OnOutgoingCall onOutgoingCall, cb_OnCallEstablished onCallEstablished, cb_OnCallCleared onCallCleared, cb_OnStartLogicalChannel onStartLogicalChannel)

This function is used to register the H323 Endpoint callback functions.

- EXTERN int **ooDestroyH323Ep** ()

This function is the last function to be invoked after done using the stack.

Detailed Description

This file contains H323 endpoint related functions.

Definition in file **ooh323ep.h**.

ooports.h File Reference

This file contains functions to manage ports used by the stack.

```
#include "ootypes.h"
```

Defines

- #define **OOTCP** 1
- #define **OOUDP** 2
- #define **OORTP** 3

Functions

- EXTERN int **ooSetTCPPorts** (int start, int max)

Sets the range of ports that can be potentially used for TCP connections.

- EXTERN int **ooSetUDPPorts** (int start, int max)

Sets the range of ports that can be potentially used for UDP transport.

- EXTERN int **ooSetRTPPorts** (int start, int max)

Sets the range of ports that can be potentially used for RTP RTCP transport.

- EXTERN int **ooGetNextPort** (**ooEndPoint** *ep, int type)

Get the next port of type TCP/UDP/RTP from the corresponding range.

- EXTERN int **ooBindPort** (**ooEndPoint** *ep, int type, **OOSOCKET** socket)

Bind socket to a port within the port range specified by the application at the startup.

Detailed Description

This file contains functions to manage ports used by the stack.

Definition in file **ooports.h**.

Function Documentation

EXTERN int ooBindPort (ooEndPoint * *ep*, int *type*, OOSOCKET *socket*)

Bind socket to a port within the port range specified by the application at the startup.

Parameters:

ep Reference to H323 Endpoint structure.
type Type of the port required for the socket.

socket The socket to be bound.

Returns:

In case of success returns the port number to which socket is bound and in case of failure just returns a negative value.

EXTERN int ooGetNextPort (ooEndPoint * *ep*, int *type*)

Get the next port of type TCP/UDP/RTP from the corresponding range.

When max value for the range is reached, it starts again from the first port number of the range.

Parameters:

ep Reference to the H323 Endpoint structure.
type Type of the port to be retrieved(OOTCP/OOUDP/OORTP).

Returns:

The next port number for the specified type is returned.

EXTERN int ooSetRTPPorts (int *start*, int *max*)

Sets the range of ports that can be potentially used for RTP RTCP transport.

Parameters:

start Starting port number for the range.
max Ending port number for the range

Returns:

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

EXTERN int ooSetTCPPorts (int *start*, int *max*)

Sets the range of ports that can be potentially used for TCP connections.

Parameters:

start Starting port number for the range.
max Ending port number for the range

Returns:

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

EXTERN int ooSetUDPPorts (int *start*, int *max*)

Sets the range of ports that can be potentially used for UDP transport.

Parameters:

start Starting port number for the range.
max Ending port number for the range

Returns:

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

ooq931.h File Reference

This file contains functions to support call signalling.

```
#include "ooasn1.h"
#include "ootypes.h"
#include "H323-MESSAGES.h"
```

Data Structures

- struct **Q931InformationElement**

Defines

- #define **Q931_E_TOOSHORT** (-1001)
- #define **Q931_E_INVCALLREF** (-1002)
- #define **Q931_E_INVLENGTH** (-1003)

Typedefs

- typedef Q931InformationElement **Q931InformationElement**

Enumerations

- enum **Q931MsgTypes** { **Q931NationalEscapeMsg** = 0x00, **Q931AlertingMsg** = 0x01, **Q931CallProceedingMsg** = 0x02, **Q931ConnectMsg** = 0x07, **Q931ConnectAckMsg** = 0x0f, **Q931ProgressMsg** = 0x03, **Q931SetupMsg** = 0x05, **Q931SetupAckMsg** = 0x0d, **Q931ResumeMsg** = 0x26, **Q931ResumeAckMsg** = 0x2e, **Q931ResumeRejectMsg** = 0x22, **Q931SuspendMsg** = 0x25, **Q931SuspendAckMsg** = 0x2d, **Q931SuspendRejectMsg** = 0x21, **Q931UserInformationMsg** = 0x20, **Q931DisconnectMsg** = 0x45, **Q931ReleaseMsg** = 0x4d, **Q931ReleaseCompleteMsg** = 0x5a, **Q931RestartMsg** = 0x46, **Q931RestartAckMsg** = 0x4e, **Q931SegmentMsg** = 0x60, **Q931CongestionCtrlMsg** = 0x79, **Q931InformationMsg** = 0x7b, **Q931NotifyMsg** = 0x6e, **Q931StatusMsg** = 0x7d, **Q931StatusEnquiryMsg** = 0x75, **Q931FacilityMsg** = 0x62 }
- enum **Q931IECodes** { **Q931BearerCapabilityIE** = 0x04, **Q931CauseIE** = 0x08, **Q931FacilityIE** = 0x1c, **Q931ProgressIndicatorIE** = 0x1e, **Q931CallStateIE** = 0x14, **Q931DisplayIE** = 0x28, **Q931SignalIE** = 0x34, **Q931CallingPartyNumberIE** = 0x6c, **Q931CalledPartyNumberIE** = 0x70, **Q931RedirectingNumberIE** = 0x74, **Q931UserUserIE** = 0x7e }
- enum **Q931InformationTransferCapability** { **Q931TransferSpeech**, **Q931TransferUnrestrictedDigital** = 8, **Q931TransferRestrictedDigital** = 9, **Q931Transfer3_1kHzAudio** = 16, **Q931TrasnferUnrestrictedDigitalWithTones** = 17, **Q931TransferVideo** = 24 }
- enum **Q931CauseValues** { **Q931NoRouteToNetwork** = 0x02, **Q931NoRouteToDestination** = 0x03, **Q931ChannelUnacceptable** = 0x06, **Q931NormalCallClearing** = 0x10, **Q931UserBusy** = 0x11, **Q931NoResponse** = 0x12, **Q931NoAnswer** = 0x13, **Q931SubscriberAbsent** = 0x14, **Q931CallRejected** = 0x15, **Q931NumberChanged** = 0x16, **Q931Redirection** = 0x17, **Q931DestinationOutOfOrder** = 0x1b, **Q931InvalidNumberFormat** = 0x1c, **Q931StatusEnquiryResponse** = 0x1e, **Q931NoCircuitChannelAvailable** = 0x22, **Q931Congestion** = 0x2a, **Q931InvalidCallReference** = 0x51, **Q931ErrorInCauseIE** = 0 }
- enum **Q931SignalInfo** { **Q931SignalDialToneOn**, **Q931SignalRingBackToneOn**, **Q931SignalInterceptToneOn**, **Q931SignalNetworkCongestionToneOn**,

Q931SignalBusyToneOn, **Q931SignalConfirmToneOn**, **Q931SignalAnswerToneOn**, **Q931SignalCallWaitingTone**, **Q931SignalOffhookWarningTone**, **Q931SignalPreemptionToneOn**, **Q931SignalTonesOff** = 0x3f, **Q931SignalAlertingPattern0** = 0x40, **Q931SignalAlertingPattern1**, **Q931SignalAlertingPattern2**, **Q931SignalAlertingPattern3**, **Q931SignalAlertingPattern4**, **Q931SignalAlertingPattern5**, **Q931SignalAlertingPattern6**, **Q931SignalAlertingPattern7**, **Q931SignalAlretingOff** = 0x4f, **Q931SignalErrorInIE** = 0x100 }

- enum **Q931NumberingPlanCodes** { **Q931UnknownPlan** = 0x00, **Q931ISDNPlan** = 0x01, **Q931DataPlan** = 0x03, **Q931TelexPlan** = 0x04, **Q931NationalStandardPlan** = 0x08, **Q931PrivatePlan** = 0x09, **Q931ReservedPlan** = 0x0f }
- enum **Q931TypeOfNumberCodes** { **Q931UnknownType** = 0x00, **Q931InternationalType** = 0x01, **Q931NationalType** = 0x02, **Q931NetworkSpecificType** = 0x03, **Q931SubscriberType** = 0x04, **Q931AbbreviatedType** = 0x06, **Q931ReservedType** = 0x07 }

Functions

- EXTERN int **ooQ931Decode** (**Q931Message** *msg, int length, ASN1OCTET *data)

This function is invoked to decode a Q931 message.

- EXTERN int **ooDecodeUUIE** (**Q931Message** *q931Msg)

This function is used to decode the UUIE of the message from the list of ies.

- EXTERN int **ooEncodeUUIE** (**Q931Message** *q931msg)

This function is used to encode the UUIE field of the Q931 message.

- EXTERN Q931InformationElement * **ooQ931GetIE** (const **Q931Message** *q931msg, int ieCode)

This function is invoked to retrieve an IE element from a Q931 message.

- EXTERN void **ooQ931Print** (const **Q931Message** *q931msg)

This function is invoked to print a Q931 message.

- EXTERN int **ooCreateQ931Message** (**Q931Message** **msg, int msgType)

This function is invoked to create an outgoing Q931 message.

- EXTERN ASN1USINT **ooGenerateCallReference** ()

This function is invoked to generate a unique call reference number.

- EXTERN int **ooGenerateCallIdentifier** (H225CallIdentifier *callid)

This function is used to generate a unique call identifier for the call.

- EXTERN int **ooFreeQ931Message** (**Q931Message** *q931Msg)

This function is invoked to release the memory used up by a Q931 message.

- EXTERN int **ooGetOutgoingQ931Msgbuf** (**ooCallData** *call, ASN1OCTET *msgbuf, int *len, int *msgType)

This function is invoked to retrive the outgoing message buffer for Q931 message.

- EXTERN int **ooSendReleaseComplete** (**ooCallData** *call)

This function is invoked to send a ReleaseComplete message for the currently active call.

- EXTERN int **ooSendCallProceeding** (**ooCallData** *call)

This function is invoked to send a call proceeding message in response to received setup message.

- EXTERN int **ooSendAlerting** (**ooCallData** *call)

This function is invoked to send alerting message in response to received setup message.

- EXTERN int **ooSendFacility** (**ooCallData** *call)

This function is invoked to send Facility message.

- EXTERN int **ooSendConnect** (**ooCallData** *call)

This function is invoked to send a Connect message in response to received setup message.

- EXTERN int **ooH323MakeCall** (char *destip, int port, char *callToken)

This function is used to send a SETUP message for outgoing call.

- EXTERN int **ooH323HangCall** (char *callToken)

This function is used to handup a currently active call.

- EXTERN int **ooAcceptCall** (**ooCallData** *call)

Function to accept a call by sending connect.

- EXTERN int **ooH323MakeCall_helper** (**ooCallData** *call)

An helper function to ooMakeCall.

Detailed Description

This file contains functions to support call signalling.

Definition in file **ooq931.h**.

oosndrtp.h File Reference

This file contains functions to read from sound device and playback.

```
#include <stdlib.h>
#include "ootypes.h"
#include <dlfcn.h>
```

Typedefs

- typedef int(* **MediaAPI_CreateTxRTPChan**)(int *channelId, char *destip, int port)

Signature for function to Create Tx RTP channel.

- typedef int(* **MediaAPI_CloseTxRTPChan**)(int)

Signature for function to Close Tx RTP channel.

- typedef int(* **MediaAPI_CreateRecvRTPChan**)(int *channelId, char *localip, int localport)

Signature for function to Create Rx RTP channel.

- typedef int(* **MediaAPI_CloseRecvRTPChan**)(int)

Signature for function to Close Rx RTP channel.

- typedef int(* **MediaAPI_StartTxWaveFile**)(int channelId, char *filename)

Signature for function to Start transmission of media file.

- typedef int(* **MediaAPI_StopTxWaveFile**)(int channelId)

Signature for function to Stop transmission of media file.

- typedef int(* **MediaAPI_StartTxMic**)(int channelId)

Signature for function to Start transmitting captured audio from microphone.

- typedef int(* **MediaAPI_StopTxMic**)(int channelId)

Signature for function to Stop transmitting microphone data.

- typedef int(* **MediaAPI_StartRecvAndPlayback**)(int channelId)

Signature for function to Start receiving rtp data and playback.

- typedef int(* **MediaAPI_StopRecvAndPlayback**)(int channelId)

Signature for function to stop receiving rtp data.

- typedef int(* **MediaAPI_InitializePlugin**)()

Signature for function to Initialize the media plug-in.

Functions

- EXTERN int **ooLoadSndRTPPlugin** (char *name)

Loads the media plugin into the process space.

- EXTERN int **ooReleaseSndRTPPlugin** ()

Unloads the plug-in from process space.

- EXTERN int **ooCreateTransmitRTPChannel** (char *destip, int port)

Creates a transmit RTP channel.

- EXTERN int **ooCloseTransmitRTPChannel** ()

Closes a transmit RTP channel.

- EXTERN int **ooCreateReceiveRTPChannel** (char *localip, int localport)

Creates a receive RTP channel.

- EXTERN int **ooCloseReceiveRTPChannel** ()

Closes a receive RTP channel.

- EXTERN int **ooStartTransmitWaveFile** (char *filename)

Start transmitting a audio file.

- EXTERN int **ooStopTransmitWaveFile** ()

Stop transmission of a audio file.

- EXTERN int **ooStartTransmitMic** ()

Starts capturing audio data from mic and transmits it as rtp stream.

- EXTERN int **ooStopTransmitMic** ()

Stop transmission of mic audio data.

- EXTERN int **ooStartReceiveAudioAndPlayback** ()

Starts receiving rtp stream data and play it on the speakers.

- EXTERN int **ooStopReceiveAudioAndPlayback** ()

Stop receiving rtp stream data.This calls corresponding interface function of the plug-in library.

- EXTERN int **ooStartReceiveAudioAndRecord** ()

Not suuported currently.

- EXTERN int **ooStopReceiveAudioAndRecord** ()

Not supported currently.

- EXTERN int **ooSetLocalRTPAndRTCPAddrs** ()

Set local RTP and RTCP addresses for the session.

- EXTERN int **ooRTPShutDown** ()

Closes transmit and receive RTP channels, if open.

Variables

- void * **media**

Detailed Description

This file contains functions to read from sound device and playback.
It also provides a wrapper for oRTP function calls and creates threads for receiving and sending rtp packets.
Definition in file **oosndrtp.h**.

ooSocket.h File Reference

Common runtime constants, data structure definitions, and run-time functions to support the sockets' operations.

```
#include <sys/types.h>
#include "sys/time.h"
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <arpa/inet.h>
#include "ooasn1.h"
```

Defines

- #define **OOSOCKET_INVALID** ((**OOSOCKET**)-1)
- #define **OOIPADDR_ANY** ((**OOIPADDR**)0)
- #define **OOIPADDR_LOCAL** ((**OOIPADDR**)0x7f000001UL) /* 127.0.0.1 */

Typedefs

- typedef int **OOSOCKET**

Socket's handle.

- typedef unsigned long **OOIPADDR**

The IP address represented as unsigned long value.

Functions

- EXTERN int **ooSocketAccept** (**OOSOCKET** socket, **OOSOCKET** *pNewSocket, **OOIPADDR** *destAddr, int *destPort)

This function permits an incoming connection attempt on a socket.

- EXTERN int **ooSocketAddrToStr** (**OOIPADDR** ipAddr, char *pbuf, int bufsize)

This function converts an IP address to its string representation.

- EXTERN int **ooSocketBind** (**OOSOCKET** socket, **OOIPADDR** addr, int port)

This function associates a local address with a socket.

- EXTERN int **ooSocketClose** (**OOSOCKET** socket)

This function closes an existing socket.

- EXTERN int **ooSocketConnect** (**OOSOCKET** socket, const char *host, int port)

This function establishes a connection to a specified socket.

- EXTERN int **ooSocketCreate** (**OOSOCKET** *psocket)

This function creates a socket.

- EXTERN int **ooSocketCreateUDP** (**OOSOCKET** *psocket)

This function creates a UDP datagram socket.

- EXTERN int **ooSocketsInit** (void)

This function initiates use of sockets by an application.

- EXTERN int **ooSocketsCleanup** (void)

This function terminates use of sockets by an application.

- EXTERN int **ooSocketListen** (**OOSOCKET** socket, int maxConnection)

This function places a socket a state where it is listening for an incoming connection.

- EXTERN int **ooSocketRecv** (**OOSOCKET** socket, ASN1OCTET *pbuf, ASN1UINT bufsize)

This function receives data from a connected socket.

- EXTERN int **ooSocketRecvFrom** (**OOSOCKET** socket, ASN1OCTET *pbuf, ASN1UINT bufsize, char *remotehost, int *remoteport)

This function receives data from a connected/unconnected socket.

- EXTERN int **ooSocketSend** (**OOSOCKET** socket, const ASN1OCTET *pdata, ASN1UINT size)

This function sends data on a connected socket.

- EXTERN int **ooSocketSendTo** (**OOSOCKET** socket, const ASN1OCTET *pdata, ASN1UINT size, const char *remotehost, int remoteport)

This function sends data on a connected or unconnected socket.

- EXTERN int **ooSocketSelect** (int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout)

This function is used for synchronous monitoring of multiple sockets.

- EXTERN int **ooSocketStrToAddr** (const char *pIPAddrStr, **OOIPADDR** *pIPAddr)

This function converts the string with IP address to a double word representation.

- EXTERN int **ooGetLocalIPAddress** (char *pIPAddrs)

This function retrives the IP address of the local host.

- EXTERN long **ooHTONL** (long val)
- EXTERN short **ooHTONS** (short val)

Detailed Description

Common runtime constants, data structure definitions, and run-time functions to support the sockets' operations.

Definition in file **ooSocket.h**.

ooStackCmds.h File Reference

This file contains stack commands which an user application can use to make call, hang call etc.

```
#include "ootypes.h"
```

Functions

- EXTERN int **ooMakeCall** (char *destip, int port, char *callToken)

This function is used by an application to place a call.

- EXTERN int **ooHangCall** (char *callToken)

This function is used by an user application to hang a call.

- EXTERN int **ooStopMonitor** ()

This function is used by the user application to stop monitoring calls.

Detailed Description

This file contains stack commands which an user application can use to make call, hang call etc.

Definition in file **ooStackCmds.h**.

ooTimer.h File Reference

Timer structures and functions.

```
#include "ooSocket.h"
```

Data Structures

- struct **_OOTimer**

Typedefs

- typedef int(* **OOTimerCbFunc**)(struct _OOTimer *pTimer, void *data)
- typedef _OOTimer **OOTimer**

Functions

- EXTERN void **ooTimerComputeExpireTime** (OOTimer *pTimer)

This function computes the relative expiration time from the current time for the given timer object.

- EXTERN OOTimer * **ooTimerCreate** (OOTimerCbFunc cb, OOUINT32 deltaSecs, void *data, OOBOOL reRegister)

This function creates and initializes a new timer object.

- EXTERN void **ooTimerDelete** (OOTimer *pTimer)

This function deletes the given timer object.

- EXTERN OOBOOL **ooTimerExpired** (OOTimer *pTimer)

This function checks a timer to determine if it is expired.

- EXTERN void **ooTimerFireExpired** (void)

This function loops through the global timer list and fires all expired timers by calling the registered callback functions.

- EXTERN int **ooTimerInsertEntry** (OOTimer *pTimer)

This function inserts the given timer object into the correct chronological position in the global timer list.

- EXTERN struct timeval * **ooTimerNextTimeout** (struct timeval *ptimeout)

This function calculates the relative time from the current time that the first timer in global timer list will expire.

- EXTERN void **ooTimerReset** (OOTimer *pTimer)

This function resets the given timer object if its reregister flag is set.

Detailed Description

Timer structures and functions.

Definition in file **ooTimer.h**.

Function Documentation

EXTERN void ooTimerComputeExpireTime (OOTimer * *pTimer*)

This function computes the relative expiration time from the current time for the given timer object.

Parameters:

pTimer Pointer to timer object.

EXTERN OOTimer* ooTimerCreate (OOTimerCbFunc *cb*, OOUINT32 *deltaSecs*, void * *data*, OOBOOL *reRegister*)

This function creates and initializes a new timer object.

Parameters:

cb Timer callback function.
deltaSecs Time in seconds to timer expiration.
data Callback user data argument.
reRegister Should timer be re-registered after it expires?

Returns:

Pointer to created timer object.

EXTERN void ooTimerDelete (OOTimer * *pTimer*)

This function deletes the given timer object.

Parameters:

pTimer Pointer to timer object.

EXTERN OOBOOL ooTimerExpired (OOTimer * *pTimer*)

This function checks a timer to determine if it is expired.

Parameters:

pTimer Pointer to timer object.

Returns:

True if timer expired, false if not.

EXTERN int ooTimerInsertEntry (OOTimer * *pTimer*)

This function inserts the given timer object into the correct chronological position in the global timer list.

Parameters:

pTimer Pointer to timer object.

Returns:

Index to position where inserted in list.

EXTERN struct timeval* ooTimerNextTimeout (struct timeval * *ptimeout*)

This function calculates the relative time from the current time that the first timer in global timer list will expire.

Parameters:

ptimeout timeval structure to receive timeout value.

Returns:

ptimeout

EXTERN void ooTimerReset (OOTimer * *pTimer*)

This function resets the given timer object if its reregister flag is set.

Otherwise, it is deleted.

Parameters:

pTimer Pointer to timer object.

ootypes.h File Reference

This file contains the definitions of common constants and data structures.

```
#include "ooSocket.h"
#include "MULTIMEDIA-SYSTEM-CONTROL.h"
#include "H323-MESSAGES.h"
#include "ooasn1.h"
```

Data Structures

- struct **ooCommand**

Structure for stack commands.

- struct **ooH323Ports**

This structure is used to define the port ranges to be used by the application.

- struct **Q931Message**

Defines the Q931 message structure.

- struct **H245Message**

Defines the H245 message structure.

- struct **ooMediaInfo**
- struct **ooLogicalChannel**

Structure to store information of logical channels for call.

- struct **ooAliases**
- struct **ooCallData**

Structure to store all the information related to a particular call.

- struct **ooH323EpCapability**

Structure to store information related to end point capability.

- struct **ooEndPoint**

Structure to store all the config information related to the endpoint created by an application.

Defines

- #define **TRACELVL** 1
- #define **OOTRACEERR1**(a) ooTrace(a)
- #define **OOTRACEERR2**(a, b) ooTrace(a,b)
- #define **OOTRACEERR3**(a, b, c) ooTrace(a,b,c)
- #define **OOTRACEERR4**(a, b, c, d) ooTrace(a,b,c,d)
- #define **OOTRACEWARN1**(a) if(TRACELVL > 1) ooTrace(a)

- #define **OOTRACEWARN2**(a, b) if(TRACELVL > 1) ooTrace(a,b)
- #define **OOTRACEWARN3**(a, b, c) if(TRACELVL > 1) ooTrace(a,b,c)
- #define **OOTRACEWARN4**(a, b, c, d) if(TRACELVL > 1) ooTrace(a,b,c,d)
- #define **OOTRACEINFO1**(a) if(TRACELVL > 2) ooTrace(a)
- #define **OOTRACEINFO2**(a, b) if(TRACELVL > 2) ooTrace(a,b)
- #define **OOTRACEINFO3**(a, b, c) if(TRACELVL > 2) ooTrace(a,b,c)
- #define **OOTRACEINFO4**(a, b, c, d) if(TRACELVL > 2) ooTrace(a,b,c,d)
- #define **OOTRACEINFO5**(a, b, c, d, e) if(TRACELVL > 2) ooTrace(a,b,c,d,e)
- #define **OOTRACEDBGA1**(a) if(TRACELVL > 3) ooTrace(a)
- #define **OOTRACEDBGA2**(a, b) if(TRACELVL > 3) ooTrace(a,b)
- #define **OOTRACEDBGA3**(a, b, c) if(TRACELVL > 3) ooTrace(a,b,c)
- #define **OOTRACEDBGA4**(a, b, c, d) if(TRACELVL > 3) ooTrace(a,b,c,d)
- #define **OOTRACEDBGB1**(a) if(TRACELVL > 4) ooTrace(a)
- #define **OOTRACEDBGB2**(a, b) if(TRACELVL > 4) ooTrace(a,b)
- #define **OOTRACEDBGB3**(a, b, c) if(TRACELVL > 4) ooTrace(a,b,c)
- #define **OOTRACEDBGB4**(a, b, c, d) if(TRACELVL > 4) ooTrace(a,b,c,d)
- #define **OOTRACEDBGC1**(a) if(TRACELVL > 5) ooTrace(a)
- #define **OOTRACEDBGC2**(a, b) if(TRACELVL > 5) ooTrace(a,b)
- #define **OOTRACEDBGC3**(a, b, c) if(TRACELVL > 5) ooTrace(a,b,c)
- #define **OOTRACEDBGC4**(a, b, c, d) if(TRACELVL > 5) ooTrace(a,b,c,d)
- #define **OO_FAILED** -1
- #define **OO_OK** 1
- #define **OO_MasterSlave_Idle** 2

Various states of master slave determination prcedure.

- #define **OO_MasterSlave_DetermineSent** 3
- #define **OO_MasterSlave_AckReceived** 4
- #define **OO_MasterSlave_Master** 5
- #define **OO_MasterSlave_Slave** 6
- #define **OO_LocalTermCapExchange_Idle** 9

States for Capability Exchange Procedure.

- #define **OO_LocalTermCapSetSent** 10
- #define **OO_LocalTermCapSetAckRecvd** 11
- #define **OO_RemoteTermCapExchange_Idle** 12
- #define **OO_RemoteTermCapSetRecvd** 13
- #define **OO_RemoteTermCapSetAckSent** 14
- #define **OO_FASTSTART_SENT** 15
- #define **OO_FASTSTART_RECEIVED** 16
- #define **OO_FASTSTART_ACCEPTED** 17
- #define **OO_FASTSTART_REFUSED** 18
- #define **OO_UNKNOWN** 20

Call Clear Reasons.

- #define **OO_REMOTE_CLOSED_CONNECTION** 21
- #define **OO_REMOTE_CLOSED_H245_CONNECTION** 22
- #define **OO_REMOTE_CLEARED** 23
- #define **OO_HOST_CLEARED** 24
- #define **OO_NORMAL** 25
- #define **OO_TotalMessages** 5
- #define **OO_CALL_CREATED** 50

call states

- #define **OO_CALL_CONNECTING** 51
- #define **OO_CALL_CONNECTED** 52
- #define **OO_CALL_CLEAR** 53 /* call marked for clearing */
- #define **OO_CALL_CLEAR_CLOLCS** 54 /* Logical Channels closed*/
- #define **OO_CALL_CLEAR_CLELCS** 55 /* Logical Channels cleared*/
- #define **OO_CALL_CLEAR_ENDSESSION** 56 /* EndSession command sent*/
- #define **OO_CALL_CLEAR_CLOSEH245** 57 /* H245 sockets closed*/
- #define **OO_CALL_CLEAR_RELEASE** 58 /* Release Sent */
- #define **OO_CALL_CLEARED** 59 /* Call Cleared */
- #define **CALL_STATE_BASE** 50
- #define **OO_H245SESSION_INACTIVE** 61

H245 Session state.

- #define **OO_H245SESSION_ACTIVE** 62
- #define **OO_LOGICALCHAN_IDLE** 70

Logical Channel states.

- #define **OO_LOGICALCHAN_PROPOSED** 71
- #define **OO_LOGICALCHAN_ESTABLISHED** 72
- #define **OOTERMTYPE** 60

Terminal type of the endpoint.

- #define **MAXLOGMSGLEN** 1024
- #define **OOQ931MSG** 101

Various message types for H225 and H245 messages.

- #define **OOH245MSG** 102
- #define **OOSetup** 103
- #define **OOCallProceeding** 104
- #define **OOAlert** 105
- #define **OOConnect** 106
- #define **OOReleaseComplete** 107
- #define **OOFacility** 108
- #define **OOMasterSlaveDetermination** 109
- #define **OOMasterSlaveAck** 110
- #define **OOMasterSlaveReject** 111
- #define **OOMasterSlaveRelease** 112
- #define **OOTerminalCapabilitySet** 113
- #define **OOTerminalCapabilitySetAck** 114
- #define **OOTerminalCapabilitySetReject** 115
- #define **OOOpenLogicalChannel** 116
- #define **OOOpenLogicalChannelAck** 117
- #define **OOOpenLogicalChannelReject** 118
- #define **OOOpenLogicalChannelRelease** 119
- #define **OOEndSessionCommand** 120
- #define **OOCloseLogicalChannel** 121
- #define **OOCloseLogicalChannelAck** 122
- #define **OORequestChannelClose** 123
- #define **OORequestChannelCloseAck** 124
- #define **OO_MSGTYPE_BASE** 101
- #define **TCPPORTSSTART** 12030

Default port ranges used.

- #define **TCPPORTSEND** 12230
- #define **UDPPORTSSTART** 13030
- #define **UDPPORTSEND** 13230
- #define **RTPPORTSSTART** 14030
- #define **RTPPORTSEND** 14230
- #define **MAXMSGLEN** 4096

Maximum length for received messages.

- #define **OO_CMD_MAKECALL** 201
- #define **OO_CMD_HANGCALL** 202
- #define **OO_CMD_STOPMONITOR** 203
- #define **OO_CALLMODE_AUDIOCALL** 301

Endpoint call modes.

- #define **OO_CALLMODE_AUDIORX** 302
- #define **OO_CALLMODE_AUDIOTX** 303
- #define **OO_CALLMODE_VIDEOCALL** 304
- #define **OO_CALLMODE_FAX** 305

Typedefs

- typedef int(* **ChannelCallback**)(void *)

Type of callback functions to be registered at the time of channel creation.

- typedef int(* **CommandCallback**)()

Type of callback function registered at initialization for handling commands.

- typedef **ooCommand ooCommand**

Structure for stack commands.

- typedef **Q931Message Q931Message**

Defines the Q931 message structure.

- typedef **H245Message H245Message**

Defines the H245 message structure.

- typedef ooMediaInfo **ooMediaInfo**
- typedef **ooLogicalChannel ooLogicalChannel**

Structure to store information of logical channels for call.

- typedef ooAliases **ooAliases**
- typedef **ooCallData ooCallData**

Structure to store all the information related to a particular call.

- typedef int(* **cb_StartReceiveChannel**)(**ooCallData** *call, **ooLogicalChannel** *pChannel)

Call back for starting media receive channel.

- typedef int(* **cb_StartTransmitChannel**)(**ooCallData** *call, **ooLogicalChannel** *pChannel)

callback for starting media transmit channel

- typedef int(* **cb_StopReceiveChannel**)(**ooCallData** *call, **ooLogicalChannel** *pChannel)

callback to stop media receive channel

- typedef int(* **cb_StopTransmitChannel**)(**ooCallData** *call, **ooLogicalChannel** *pChannel)

callback to stop media transmit channel

- typedef **ooH323EpCapability ooH323EpCapability**

Structure to store information related to end point capability.

- typedef int(* **cb_OnIncomingCall**)(**ooCallData** *call)
- typedef int(* **cb_OnOutgoingCall**)(**ooCallData** *call)
- typedef int(* **cb_OnCallAnswered**)(**ooCallData** *call)
- typedef int(* **cb_OnAlerting**)(**ooCallData** *call)
- typedef int(* **cb_OnCallCleared**)(**ooCallData** *call)
- typedef int(* **cb_OnCallEstablished**)(**ooCallData** *call)
- typedef int(* **cb_OnStartLogicalChannel**)(**ooCallData** *call)
- typedef **ooEndPoint ooEndPoint**

Structure to store all the config information related to the endpoint created by an application.

Variables

- int **gCallTokenBase**

Stores base value for generating new call token.

- int **gCallTokenMax**

Stores Max value for call token, at which token is reset.

- int **gCurCallToken**

Stores current value for call token generation.

- int **gMonitor**
- DList **gCmdList**

List of stack commands issued by application which have to be processed.

- OOCTXT **gCtxt**

Context for stack commands list.

- pthread_mutex_t **gCmdMutex**

Mutex to protect access to stack commands list.

- **ooEndPoint gH323ep**

Global endpoint structure.

Detailed Description

This file contains the definitions of common constants and data structures.

Definition in file **ootypes.h**.

Define Documentation

#define OO_CALLMODE_AUDIOCALL 301

Endpoint call modes.

The call mode of the endpoint dictates what type of channels are created for the calls placed by the endpoint or received by the endpoint.

Definition at line 260 of file ootypes.h.

#define OOTERMTYPE 60

Terminal type of the endpoint.

Default is 60.

Definition at line 176 of file ootypes.h.

Typedef Documentation

typedef struct H245Message H245Message

Defines the H245 message structure.

All request/response and command messages are represented using this structure.

typedef struct Q931Message Q931Message

Defines the Q931 message structure.

Contains context for memory allocation, protocol Discriminator, call reference, meesage type and list of user user IEs.

printHandler.h File Reference

This is an implementation of a simple print handler.

```
#include "asn1CEvtHndlr.h"
```

Functions

- void **initializePrintHandler** (**Asn1NamedCEventHandler** *printHandler, char *varname)
- void **finishPrint** ()
- void **indent** ()
- void **printStartElement** (const char *name, int index)
- void **printEndElement** (const char *name, int index)
- void **printBoolValue** (ASN1BOOL value)
- void **printIntValue** (ASN1INT value)
- void **printuIntValue** (ASN1UINT value)
- void **printBitStrValue** (ASN1UINT numbits, const ASN1OCTET *data)
- void **printOctStrValue** (ASN1UINT numocts, const ASN1OCTET *data)
- void **printCharStrValue** (const char *value)
- void **printCharStr16BitValue** (ASN1UINT nchars, ASN116BITCHAR *data)
- void **printCharStr32BitValue** (ASN1UINT nchars, ASN132BITCHAR *data)
- void **printNullValue** ()
- void **printOidValue** (ASN1UINT numSubIds, ASN1UINT *pSubIds)
- void **printRealValue** (double value)
- void **printEnumValue** (ASN1UINT value)
- void **printOpenTypeValue** (ASN1UINT numocts, const ASN1OCTET *data)

Variables

- **Asn1NamedCEventHandler printHandler**
- const char * **pVarName**
- int **gIndentSpaces**

Detailed Description

This is an implementation of a simple print handler.
It outputs the fields of an encoded PER message to stdout in a structured output format..
Definition in file **printHandler.h**.

rtctype.h File Reference

```
#include "ooasn1.h"
```

Defines

- #define **OS_CTYPE_UPPER** 0x1
- #define **OS_CTYPE_LOWER** 0x2
- #define **OS_CTYPE_NUMBER** 0x4
- #define **OS_CTYPE_SPACE** 0x8
- #define **OS_CTYPE_PUNCT** 0x10
- #define **OS_CTYPE_CTRL** 0x20
- #define **OS_CTYPE_HEX** 0x40
- #define **OS_CTYPE_BLANK** 0x80
- #define **OS_ISALPHA**(c) (rtCtypeTable[(unsigned)(c)]&(OS_CTYPE_UPPER|OS_CTYPE_LOWER))
- #define **OS_ISUPPER**(c) (rtCtypeTable[(unsigned)(c)]&OS_CTYPE_UPPER)
- #define **OS_ISLOWER**(c) (rtCtypeTable[(unsigned)(c)]&OS_CTYPE_LOWER)
- #define **OS_ISDIGIT**(c) (rtCtypeTable[(unsigned)(c)]&OS_CTYPE_NUMBER)
- #define **OS_ISXDIGIT**(c) (rtCtypeTable[(unsigned)(c)]&(OS_CTYPE_HEX|OS_CTYPE_NUMBER))
- #define **OS_ISSPACE**(c) (rtCtypeTable[(unsigned)(c)]&OS_CTYPE_SPACE)
- #define **OS_ISPUNCT**(c) (rtCtypeTable[(unsigned)(c)]&OS_CTYPE_PUNCT)
- #define **OS_ISALNUM**(c) (rtCtypeTable[(unsigned)(c)]&(OS_CTYPE_UPPER|OS_CTYPE_LOWER|OS_CTYPE_NUMBER))
- #define **OS_ISPRINT**(c)
- #define **OS_ISGRAPH**(c)
- #define **OS_ISCNTRL**(c) (rtCtypeTable[(unsigned)(c)]&OS_CTYPE_CTRL)
- #define **OS_TOLOWER**(c) (OS_ISUPPER(c) ? (c) - 'A' + 'a' : (c))
- #define **OS_TOUPPER**(c) (OS_ISLOWER(c) ? (c) - 'a' + 'A' : (c))

Variables

- EXTERN const ASN1OCTET **rtCtypeTable** [256]

Detailed Description

Definition in file **rtctype.h**.

Define Documentation

#define OS_ISGRAPH(c)

Value:

```
(rtCtypeTable[(unsigned)(c)]& \
(OS_CTYPE_PUNCT|OS_CTYPE_UPPER|OS_CTYPE_LOWER|OS_CTYPE_NUMBER))
```

Definition at line 57 of file rtctype.h.

#define OS_ISPRINT(c)

Value:

```
(rtCtypeTable[(unsigned)(c)]& \
(OS_CTYPE_PUNCT|OS_CTYPE_UPPER|OS_CTYPE_LOWER|OS_CTYPE_NUMBER|OS_CTYPE_BLANK))
```

Definition at line 54 of file rtctype.h.

SList.h File Reference

Singly Linked list header file.

```
#include "ooasn1.h"
```

Defines

- #define **OSMSGMALLOC**(pctxt, nbytes) memHeapAlloc(&(pctxt)->pMsgMemHeap, nbytes)
- #define **OSMSGREALLOC**(pctxt, pmem, nbytes) memHeapRealloc(&(pctxt)->pMsgMemHeap, pmem, nbytes)
- #define **OSMSGREALLOCARRAY**(pctxt, pseqof, type)
- #define **OSMSGMEMFREE**(pctxt) memHeapFreeAll(&(pctxt)->pMsgMemHeap)
- #define **OSMSGMEMFREEPTR**(pctxt, pmem) memHeapFreePtr(&(pctxt)->pMsgMemHeap, pmem)
- #define **OSMSGMEMRESET**(pctxt) memHeapReset(&(pctxt)->pMsgMemHeap)

Detailed Description

Singly Linked list header file.

Definition in file **SList.h**.

Define Documentation

#define OSMSGREALLOCARRAY(pctxt, pseqof, type)

Value:

```
do {\
if (sizeof(type)*(pseqof)->n < (pseqof)->n) return ASN_E_NOMEM; \
if (((pseqof)->elem = (type*) memHeapRealloc \
(&(pctxt)->pMsgMemHeap, (pseqof)->elem, sizeof(type)*(pseqof)->n)) == 0) \
return ASN_E_NOMEM; \
} while (0)
```

Definition at line 32 of file SList.h.

Index