# ASN1C

ASN.1 Compiler
Version 6.7
C Runtime
Reference Manual

*Objective Systems, Inc. version 6.7 — May 2014*

**Author's Contact Information**

Comments, suggestions, and inquiries regarding ASN1C may be submitted via electronic mail to info@obj-sys.com.

# Contents

# Chapter 1

# C/C++ Common Runtime Classes and Library Functions

The **ASN.1 C++ run-time classes** are wrapper classes that provide an object-oriented interface to the ASN.1 C run-time library functions. The categories of classes provided are as follows:

- Context Management classes manage the OSCTXT structure used to keep track of the working variables required to encode or decode ASN.1 messages.
- Message Buffer classes are used to manage message buffers.
- ASN1C Control Base classes are wrapper classes that are used as the base for compiler-generated ASN1C_ classes, including Date and Time Run-time classes.
- ASN.1 Type (ASN1T_) Base classes are used as the base for compiler- generated ASN1T_ C++ data structures.
- ASN.1 Stream classes are used to read and write ASN.1 messages from and to files, sockets, memory buffer, etc.
- TCP/IP or UDP Socket classes provide utility methods for doing socket I/O.
- Asn1NamedEventHandler classes include the base classes for user-defined error handler and event handler classes.

The **C run-time common library** contains common C functions used by the encoding rules (BER/DER, PER, and XER) low-level encode/decode functions. These functions are identified by their *rt* prefixes. The categories of functions provided are as follows:

- Memory Allocation macros and functions handle memory management for the ASN1C run-time.
- Context Management functions handle the allocation, initialization, and destruction of context variables (variables of type OSCTXT) that handle the working data used during encoding or decoding a message.
- Diagnostic Trace functions allow the output of trace messages to standard output that trace the execution of complier generated functions.
- Error Formatting and Print functions allow information about the encode/decode errors to be added to a context block structure and printed out.
- Memory Buffer Management functions handle the allocation, expansion, and de-allocation of dynamic memory buffers used by some encode/decode functions.

- Object Identifier Helper functions provide assistance in working with the object identifier ASN.1 type.
- Linked List and Stack Utility functions are used to maintain linked lists and stacks used within the ASN.1 run-time library functions.
- REAL Helper functions - REAL helper functions provide assistance in working with the REAL ASN.1 type. Two functions are provided to obtain the plus and minus infinity special values.
- Formatted Printing functions allow raw ASN.1 data fields to be formatted and printed to standard output and other output devices.
- Binary Coded Decimal (BCD) helper functions provide assistance in working with BCD numbers.
- Character String Conversion functions convert between standard null-terminated C strings and different ASN.1 string types.
- Big Integer Helper functions are arbitrary-precision integer manipulating functions used to maintain big integers used within the ASN.1 run-time functions.
- Comparison functions allow comparison of the values of primitive ASN.1 types. They make it possible to compare complex structures and determine what elements within those structures are different.
- Comparison to Standard Output functions do the same actions as the other comparison functions, but print the comparison results to standard output instead of to the buffer.
- Copy functions - This group of functions allows copying values of primitive ASN.1 types.
- Print Values to Standard Output functions print the output in a "name=value" format, where the value format is obtained by calling one of the ToString functions with the given value.

# Chapter 2

# Module Documentation

## 2.1 C Runtime Common Functions

### Classes

- struct ASN1OctStr
- struct ASN1DynBitStr
- struct ASN1BitStr32
- struct ASN1SeqOf
- struct ASN1SeqOfOctStr
- struct ASN1OpenType
- struct Asn1Object
- struct Asn116BitCharString
- struct Asn132BitCharString
- struct Asn1CharArray
- struct Asn1CharSet
- struct Asn116BitCharSet
- struct Asn132BitCharSet
- struct ASN1BigInt
- struct ASN1CCB

### Modules

- Object Identifier Helper Functions
- Time Helper Functions
- Character String Conversion Functions
- Comparison Functions
- Comparison to Standard Output Functions
- Copy Functions

### Defines

- #define XM_SEEK 0x01
- #define XM_ADVANCE 0x02
- #define XM_DYNAMIC 0x04

- #define XM_SKIP 0x08
- #define XM_OPTIONAL 0x10
- #define ASN_K_MAXDEPTH 32
- #define ASN_K_MAXENUM 100
- #define ASN_K_MAXERRP 5
- #define ASN_K_MAXERRSTK 8
- #define ASN_K_ENCBUFSIZ 16∗1024
- #define ASN_K_MEMBUFSEG 1024
- #define OSRTINDENTSPACES 3
- #define ASN1_K_PLUS_INFINITY 0x40
- #define ASN1_K_MINUS_INFINITY 0x41
- #define **REAL_BINARY** 0x80
- #define **REAL_SIGN** 0x40
- #define **REAL_EXPLEN_MASK** 0x03
- #define **REAL_EXPLEN_1** 0x00
- #define **REAL_EXPLEN_2** 0x01
- #define **REAL_EXPLEN_3** 0x02
- #define **REAL_EXPLEN_LONG** 0x03
- #define **REAL_FACTOR_MASK** 0x0c
- #define **REAL_BASE_MASK** 0x30
- #define **REAL_BASE_2** 0x00
- #define **REAL_BASE_8** 0x10
- #define **REAL_BASE_16** 0x20
- #define **REAL_ISO6093_MASK** 0x3F
- #define **ASN1REALMAX** (OSREAL)DBL_MAX
- #define **ASN1REALMIN** (OSREAL)-DBL_MAX
- #define ASN1DynOctStr OSDynOctStr
- #define OSSETBIT(bitStr, bitIndex) rtxSetBit (bitStr.data, bitStr.numbits, bitIndex)
- #define OSSETBITP(pBitStr, bitIndex) rtxSetBit ((pBitStr)->data, (pBitStr)->numbits, bitIndex)
- #define OSCLEARBIT(bitStr, bitIndex) rtxClearBit (bitStr.data, bitStr.numbits, bitIndex)
- #define OSCLEARBITP(pBitStr, bitIndex) rtxClearBit ((pBitStr)->data, (pBitStr)->numbits, bitIndex)
- #define OSTESTBIT(bitStr, bitIndex) rtxTestBit (bitStr.data, bitStr.numbits, bitIndex)
- #define OSTESTBITP(pBitStr, bitIndex) rtxTestBit ((pBitStr)->data, (pBitStr)->numbits, bitIndex)
- #define ASN1_K_CCBMaskSize 32
- #define ASN1_K_NumBitsPerMask 16
- #define ASN1_K_MaxSetElements (ASN1_K_CCBMaskSize∗ASN1_K_NumBitsPerMask)
- #define **ASN1NUMOCTS**(nbits) ((nbits>0)?(((nbits-1)/8)+1):0)

## Typedefs

- typedef void ∗ **ASN1ANY**
- typedef Asn1Object **ASN1Object**
- typedef OSUNICHAR **ASN116BITCHAR**
- 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 * **ASN1VideotexString**
- typedef const char * **ASN1VisibleString**
- typedef const OSUTF8CHAR * **ASN1UTF8String**
- typedef Asn116BitCharString **ASN1BMPString**
- typedef Asn132BitCharString **ASN1UniversalString**
- typedef struct ASN1BigInt ASN1BigInt
- typedef int(* ASN1DumpCbFunc )(const char *text_p, void *cbArg_p)

## Enumerations

- enum ASN1StrType { **ASN1HEX**, **ASN1BIN**, **ASN1CHR** }
- enum ASN1ActionType { **ASN1ENCODE**, **ASN1DECODE** }

- #define ALLOC_ASN1ARRAY(pctxt, pseqof, type)
- #define ALLOC_ASN1ARRAY1(pctxt, pseqof, type)

### 2.1.1 Detailed Description

The **C run-time common library** contains common C functions used by the low-level encode/decode functions. These functions are identified by their *rt* and *rtx* prefixes.

The categories of functions provided are as follows:

- Context management functions handle the allocation, initialization, and destruction of context variables (variables of type OSCTXT) that handle the working data used during the encoding or decoding of a message.
- Memory allocation macros and functions provide an optimized memory management interface.
- Doubly linked list (DList) functions are used to manipulate linked list structures that are used to model repeating XSD types and elements.
- UTF-8 and Unicode character string functions provide support for conversions to and from these formats in C or C++.
- Date/time conversion functions provide utilities for converting system and structured numeric date/time values to XML schema standard string format.
- Pattern matching function compare strings against patterns specified using regular expressions (regexp's).
- Diagnostic trace functions allow the output of trace messages to standard output.
- Error formatting and print functions allow information about encode/decode errors to be added to a context block structure and printed out.
- Memory buffer management functions handle the allocation, expansion, and de-allocation of dynamic memory buffers used by some encode/decode functions.
- Formatted print functions allow binary data to be formatted and printed to standard output and other output devices.
- Big Integer helper functions are arbitrary-precision integer manipulating functions used to maintain big integers.

## 2.1.2 Define Documentation

### 2.1.2.1 #define ALLOC_ASN1ARRAY(pctxt, pseqof, type)

**Value:**

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

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. Allocate a dynamic array. This macro allocates a dynamic array of records of the given type. This version of the macro will return the RTERR_-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

### 2.1.2.2 #define ALLOC_ASN1ARRAY1(pctxt, pseqof, type)

**Value:**

```
do {\
if (sizeof(type)*(pseqof)->n < (pseqof)->n) (pseqof)->elem = 0; \
else (pseqof)->elem = (type*) rtxMemHeapAllocZ \
(&(pctxt)->pMemHeap, sizeof(type)*(pseqof)->n); \
} while (0)
```

Allocate a dynamic array. This macro allocates a dynamic array of records of the given type. This version of the macro will set the internal parameters of the SEQUENCE OF structure to NULL 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

### 2.1.2.3 #define ASN1_K_CCBMaskSize 32

The maximum size for the context control block mask.

### 2.1.2.4 #define ASN1_K_MaxSetElements (ASN1_K_CCBMaskSize∗ASN1_K_NumBitsPerMask)

The maximum number of set elements that can be handled by the CCB.

#### 2.1.2.5 #define ASN1_K_MINUS_INFINITY 0x41

The ASN.1 Real value Minus Infinity.

#### 2.1.2.6 #define ASN1_K_NumBitsPerMask 16

The number of bits that can be set per mask.

#### 2.1.2.7 #define ASN1_K_PLUS_INFINITY 0x40

The ASN.1 Real value Plus Infinity.

#### 2.1.2.8 #define ASN1DynOctStr OSDynOctStr

We define ASN1DynOctStr to be the common generic OSDynOctStr type.

#### 2.1.2.9 #define ASN_K_ENCBUFSIZ 16*1024

dynamic encode buffer extent size

#### 2.1.2.10 #define ASN_K_MAXDEPTH 32

maximum nesting depth for messages

#### 2.1.2.11 #define ASN_K_MAXENUM 100

maximum enum values in an enum type

#### 2.1.2.12 #define ASN_K_MAXERRP 5

maximum error parameters

#### 2.1.2.13 #define ASN_K_MAXERRSTK 8

maximum levels on error ctxt stack

#### 2.1.2.14 #define ASN_K_MEMBUFSEG 1024

memory buffer extent size

#### 2.1.2.15 #define OSCLEARBIT(bitStr, bitIndex) rtxClearBit (bitStr.data, bitStr.numbits, bitIndex)

This macro clears the given bit in the given static bit string.

**Parameters**

***bitStr*** The bit string to manipulate.

***bitIndex*** The index to clear.

#### 2.1.2.16 #define OSCLEARBITP(pBitStr, bitIndex) rtxClearBit ((pBitStr)->data, (pBitStr)->numbits, bitIndex)

This macro clears the given bit in the given dynamic bit string.

**Parameters**

***pBitStr*** The pointer-to-bit string to manipulate.

***bitIndex*** The index to clear.

#### 2.1.2.17 #define OSRTINDENTSPACES 3

number of spaces for indent

#### 2.1.2.18 #define OSSETBIT(bitStr, bitIndex) rtxSetBit (bitStr.data, bitStr.numbits, bitIndex)

This macro sets the given bit in the given static bit string.

**Parameters**

***bitStr*** The bit string to manipulate.

***bitIndex*** The index to set.

#### 2.1.2.19 #define OSSETBITP(pBitStr, bitIndex) rtxSetBit ((pBitStr)->data, (pBitStr)->numbits, bitIndex)

This macro sets the given bit in the given dynamic bit string.

**Parameters**

***pBitStr*** A pointer-to-bit string to manipulate.

***bitIndex*** The index to set.

#### 2.1.2.20 #define OSTESTBIT(bitStr, bitIndex) rtxTestBit (bitStr.data, bitStr.numbits, bitIndex)

This macro tests the given bit in the given static bit string.

**Parameters**

***bitStr*** The bit string to manipulate.

***bitIndex*** The index to test.

**Returns**

TRUE if the bit is on; FALSE if the bit is off.

#### 2.1.2.21 #define OSTESTBITP(pBitStr, bitIndex) rtxTestBit ((pBitStr)->data, (pBitStr)->numbits, bitIndex)

This macro tests the given bit in the given dynamic bit string.

**Parameters**

***pBitStr*** The pointer-to-bit string to manipulate.

***bitIndex*** The index to set.

#### 2.1.2.22 #define XM_ADVANCE 0x02

advance pointer to contents on match

#### 2.1.2.23 #define XM_DYNAMIC 0x04

alloc dyn mem for decoded variable

#### 2.1.2.24 #define XM_OPTIONAL 0x10

tag test is for optional element

#### 2.1.2.25 #define XM_SEEK 0x01

seek match until found or end-of-buf

#### 2.1.2.26 #define XM_SKIP 0x08

skip to next field after parsing tag

### 2.1.3 Typedef Documentation

#### 2.1.3.1 typedef struct ASN1BigInt ASN1BigInt

A structure used to define an ASN.1 big integer. This structure is rarely, if ever, used by client code, and will instead be used by generated code to facilitate encoding and decoding integer values that cannot fit in normal C/C++ integer types.

#### 2.1.3.2 typedef int(∗ ASN1DumpCbFunc)(const char ∗text_p, void ∗cbArg_p)

ASN.1 dump utility callback function definition

### 2.1.4 Enumeration Type Documentation

#### 2.1.4.1 enum ASN1ActionType

An enumerated list of ASN.1 actions: encode or decode.

#### 2.1.4.2 enum ASN1StrType

An enumerated list of the various string types: hexadecimal, binary, and character strings.

## 2.2 Object Identifier Helper Functions

### Classes

- struct ASN1OBJID
- struct ASN1OID64

### Defines

- #define **ASN_K_MAXSUBIDS** 128

### Functions

- void rtSetOID (ASN1OBJID *ptarget, ASN1OBJID *psource)
- void rtAddOID (ASN1OBJID *ptarget, ASN1OBJID *psource)
- OSBOOL rtOIDsEqual (const ASN1OBJID *pOID1, const ASN1OBJID *pOID2)
- int rtOIDParseDottedNumberString (const char *oidstr, OSSIZE oidstrlen, ASN1OBJID *pvalue)
- OSBOOL rtOIDIsValid (const ASN1OBJID *pvalue)

### 2.2.1 Detailed Description

Object identifier helper functions provide assistance in working with the object identifier ASN.1 type.

### 2.2.2 Function Documentation

#### 2.2.2.1 void rtAddOID (ASN1OBJID * *ptarget*, ASN1OBJID * *psource*)

This function appends one object identifier to another one. It copies the data from a source variable to the end of a target variable. Typically, the source variable is a compiler-generated object identifier constant that resulted from an object identifier value specification within an ASN.1 specification.

**Parameters**

***ptarget*** A pointer to a target object identifier variable to receive object identifier data. Typically, this is a variable within a compiler-generated C structure.

***psource*** A pointer to a source object identifier variable to copy to a target. Typically, this is a compiler-generated variable corresponding to an ASN.1 value specification in the ASN.1 source file.

#### 2.2.2.2 OSBOOL rtOIDIsValid (const ASN1OBJID * *pvalue*)

This function determine if an OID value is valid according to ASN.1 rules. In particular it checks a) if number of subidentifiers is greater than or equal to 2, b) if the first subidentifier value is less than or equal to 2, and c) if the first subidentifier is 2 that the second subidentifier is less than 40.

**Parameters**

***pvalue*** Pointer to OID value to validate.

**Returns**

True if OID value is valid.

#### 2.2.2.3 int rtOIDParseDottedNumberString (const char * *oidstr*, OSSIZE *oidstrlen*, ASN1OBJID * *pvalue*)

This function parses an OID dotted number string (n.n.n) which is the from of OID XML content. Data must be in the form of numbers and dots only (i.e. OID components in other forms such as names or named number will cause a parse failure). Embedded whitespace will be ignored.

**Parameters**

***oidstr*** OID string containing data to be parsed.

***oidstrlen*** Length of the string.

***pvalue*** Pointer to OID value to receive parsed OID.

**Returns**

Status of operation: 0 = success, negative value is failure.

#### 2.2.2.4 OSBOOL rtOIDsEqual (const ASN1OBJID * *pOID1*, const ASN1OBJID * *pOID2*)

This function compares two OID values for equality.

**Parameters**

***pOID1*** Pointer to first OID value to compare.

***pOID2*** Pointer to second OID value to compare.

**Returns**

True if OID's are equal.

#### 2.2.2.5 void rtSetOID (ASN1OBJID * *ptarget*, ASN1OBJID * *psource*)

This function populates an object identifier variable with data. It copies data from a source variable to a target variable. Typically, the source variable is a compiler-generated object identifier constant that resulted from a object identifier value specification within an ASN.1 specification.

**Parameters**

***ptarget*** A pointer to a target object identifier variable to receive object * identifier data. Typically, this is a variable within a compiler-generated C structure.

***psource*** A pointer to a source object identifier variable to copy to a target. Typically, this is a compiler-generated variable corresponding to an ASN.1 value specification in the ASN.1 source file.

## 2.3 Time Helper Functions

### Functions

- int rtMakeGeneralizedTime (OSCTXT *pctxt, const OSNumDateTime *dateTime, char **outdata, size_t outdataSize)
- int rtMakeUTCTime (OSCTXT *pctxt, const OSNumDateTime *dateTime, char **outdata, size_t outdataSize)
- int rtParseGeneralizedTime (OSCTXT *pctxt, const char *value, OSNumDateTime *dateTime)
- int rtParseUTCTime (OSCTXT *pctxt, const char *value, OSNumDateTime *dateTime)
- void normalizeTimeZone (OSNumDateTime *pvalue)

### 2.3.1 Detailed Description

Utility functions for working with time strings in different formats. rtMake* functions create time strings, rtParse* functions parse time strings into the C OSNumDateTime structure defined in osSysTypes.h. Implementations of these functions exist for the ASN.1 GeneralizedTime and UTCTime formats.

### 2.3.2 Function Documentation

#### 2.3.2.1 void normalizeTimeZone (OSNumDateTime * *pvalue*)

This function normalizes the time zone for the given datetime structure.

**Parameters**

***pvalue*** A pointer-to an OSNumDateTime structure.

#### 2.3.2.2 int rtMakeGeneralizedTime (OSCTXT * *pctxt*, const OSNumDateTime * *dateTime*, char ** *outdata*, size_t *outdataSize*)

This function creates a time string in ASN.1 GeneralizedTime format as specified in the X.680 ITU-T standard.

**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.

***dateTime*** A pointer to a date/time structure that contains components of the date and time.

***outdata*** A pointer to a pointer to a destination string. If outdataSize is non-zero, it should be a pointer to a pointer to an actual array. Otherwise, the memory will be allocated and the pointer will be stored in the `outdata`.

***outdataSize*** A size of outdata (in octets). If zero, the memory for the `outdata` will be allocated. If not, the `outdata` 's size should be big enough to receive the generated time string. Otherwise, error code will be returned.

**Returns**

Completion status of operation:

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

#### 2.3.2.3 int rtMakeUTCTime (OSCTXT * *pctxt*, const OSNumDateTime * *dateTime*, char ** *outdata*, size_t *outdataSize*)

This function creates a time string in ASN.1 UTCTime format as specified in the X.680 ITU-T standard.

**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.

***dateTime*** A pointer to a date/time structure that contains components of the date and time.

***outdata*** A pointer to a pointer to a destination string. If outdataSize is non-zero, it should be a pointer to a pointer to an actual array. Otherwise, the memory will be allocated and the pointer will be stored in the `outdata`.

***outdataSize*** A size of outdata (in octets). If zero, the memory for the `outdata` will be allocated. If not, the `outdata` 's size should be big enough to receive the generated time string. Otherwise, error code will be returned.

**Returns**

Completion status of operation:

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

#### 2.3.2.4 int rtParseGeneralizedTime (OSCTXT * *pctxt*, const char * *value*, OSNumDateTime * *dateTime*)

This function parses a time string that is represented in ASN.1 GeneralizedTime format as specified in the X.680 ITU-T standard. It stores the parsed result in a numeric date/time C structure.

**Parameters**

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

***value*** A pointer to the time string to be parsed.

***dateTime*** A pointer to the destination structure.

**Returns**

Completion status of operation:

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

#### 2.3.2.5 int rtParseUTCTime (OSCTXT * *pctxt*, const char * *value*, OSNumDateTime * *dateTime*)

This function parses a time string that is represented in ASN.1 UTCTime format as specified in the X.680 ITU-T standard. It stores the parsed result in a numeric date/time C structure.

**Parameters**

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

***value*** A pointer to the time string to be parsed.

***dateTime*** A pointer to the destination date/time structure.

**Returns**

Completion status of operation:

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

## 2.4 Character String Conversion Functions

### Functions

- int rtValidateStr (ASN1TAG tag, const char *pdata)
- int rtValidateChars (ASN1TAG tag, const char *pdata, size_t nchars)
- const char * rtBMPToCString (ASN1BMPString *pBMPString, char *cstring, OSUINT32 cstrsize)
- const char * rtBMPToNewCString (ASN1BMPString *pBMPString)
- const char * rtBMPToNewCStringEx (OSCTXT *pctxt, ASN1BMPString *pBMPString)
- ASN1BMPString * rtCToBMPString (OSCTXT *pctxt, const char *cstring, ASN1BMPString *pBMPString, Asn116BitCharSet *pCharSet)
- OSBOOL rtIsIn16BitCharSet (OSUNICHAR ch, Asn116BitCharSet *pCharSet)
- const char * rtUCSToCString (ASN1UniversalString *pUCSString, char *cstring, OSUINT32 cstrsize)
- const char * rtUCSToNewCString (ASN1UniversalString *pUCSString)
- const char * rtUCSToNewCStringEx (OSCTXT *pctxt, ASN1UniversalString *pUCSString)
- ASN1UniversalString * rtCToUCSString (OSCTXT *pctxt, const char *cstring, ASN1UniversalString *pUCSString, Asn132BitCharSet *pCharSet)
- OSBOOL rtIsIn32BitCharSet (OS32BITCHAR ch, Asn132BitCharSet *pCharSet)
- wchar_t * rtUCSToWCSString (ASN1UniversalString *pUCSString, wchar_t *wcstring, OSUINT32 wcstrsize)
- ASN1UniversalString * rtWCSToUCSString (OSCTXT *pctxt, wchar_t *wcstring, ASN1UniversalString *pUCSString, Asn132BitCharSet *pCharSet)
- int rtUnivStrToUTF8 (OSCTXT *pctxt, const ASN1UniversalString *pUnivStr, OSOCTET *outbuf, size_t outbufsiz)
- int rtUTF8StrToASN1DynBitStr (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, ASN1DynBitStr *pvalue)
- int rtUTF8StrnToASN1DynBitStr (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, size_t nbytes, ASN1DynBitStr *pvalue)

### 2.4.1 Detailed Description

Common utility functions are provided to convert between standard null-terminated C strings and different ASN.1 string types.

### 2.4.2 Function Documentation

#### 2.4.2.1 const char* rtBMPToCString (ASN1BMPString * *pBMPString*, char * *cstring*, OSUINT32 *cstrsize*)

This function converts a BMP string into a null-terminated C string. Any characters that are not 8-bit characters are discarded.

**Parameters**

***pBMPString*** A pointer to a BMP string structure to be converted.

***cstring*** A pointer to a buffer to receive the converted string.

***cstrsize*** The size of the buffer to receive the converted string.

**Returns**

A pointer to the returned string structure. This is the cstring argument parameter value.

#### 2.4.2.2 const char* rtBMPToNewCString (ASN1BMPString * *pBMPString*)

This function converts a BMP string into a null-terminated C string. Any characters that are not 8-bit characters are discarded. This function allocates dynamic memory to hold the converted string using the standard C run-time malloc function. The user is responsible for freeing this memory.

**Parameters**

***pBMPString*** A pointer to a BMP string structure to be converted.

**Returns**

A pointer to the returned string structure. This is the cstring argument parameter value.

#### 2.4.2.3 const char* rtBMPToNewCStringEx (OSCTXT * *pctxt*, ASN1BMPString * *pBMPString*)

This function converts a BMP string into a null-terminated C string. Any characters that are not 8-bit characters are discarded. In contrast to rtBMPToNewCString, this function allocates dynamic memory to hold the converted string using the rtMemAlloc function. The rtMemFreePtr should be called to release the allocated memory or the rtmemFree function should be called to release all memory allocated using the specified context block.

**Parameters**

***pctxt*** A pointer to a context structure.

***pBMPString*** A pointer to a BMP string structure to be converted.

**Returns**

A pointer to the returned string structure. This is the cstring argument parameter value.

#### 2.4.2.4 ASN1BMPString* rtCToBMPString (OSCTXT * *pctxt*, const char * *cstring*, ASN1BMPString * *pBMPString*, Asn116BitCharSet * *pCharSet*)

This function converts a null-terminated C string into a 16-bit BMP string structure.

**Parameters**

***pctxt*** A pointer to a context string.

***cstring*** A pointer to a null-terminated C string to be converted into a BMP string.

***pBMPString*** A pointer to a BMP string structure to receive the converted string.

***pCharSet*** A pointer to a character set structure describing the character set currently associated with the BMP character string type.

**Returns**

A pointer to BMP string structure. This is the pBMPString argument parameter value.

#### 2.4.2.5 ASN1UniversalString* rtCToUCSString (OSCTXT * *pctxt*, const char * *cstring*, ASN1UniversalString * *pUCSString*, Asn132BitCharSet * *pCharSet*)

This function converts a null-terminated C string into a 32-bit UCS-4 (Universal Character Set, 4 byes) string structure.

**Parameters**

***pctxt*** A pointer to a context structure.

***cstring*** A pointer to a null-terminated C string to be converted into a Universal string.

***pUCSString*** A pointer to a Universal string structure to receive the converted string

***pCharSet*** A pointer to a character structure describing the character set currently associated with the Universal character string type.

**Returns**

A pointer to a Universal string structure. This is the pUCSString argument parameter value.

#### 2.4.2.6 OSBOOL rtIsIn16BitCharSet (OSUNICHAR *ch*, Asn116BitCharSet * *pCharSet*)

This function tests whether the given character is in the given 16-bit character set.

**Parameters**

***ch*** A 16-bit character.

***pCharSet*** A pointer-to Asn116BitCharSet that contains the set of valid character.

**Returns**

TRUE if the character is in the set, FALSE otherwise.

#### 2.4.2.7 OSBOOL rtIsIn32BitCharSet (OS32BITCHAR *ch*, Asn132BitCharSet * *pCharSet*)

This function tests whether the given character is in the given 32-bit character set.

**Parameters**

***ch*** A 32-bit character.

***pCharSet*** A pointer-to Asn132BitCharSet that contains the set of valid character.

**Returns**

TRUE if the character is in the set, FALSE otherwise.

#### 2.4.2.8 const char* rtUCSToCString (ASN1UniversalString * *pUCSString*, char * *cstring*, OSUINT32 *cstrsize*)

This function converts a Universal 32-bit string into a null-terminated C string. Any characters that are not 8-bit characters are discarded.

**Parameters**

***pUCSString*** A pointer to a Universal string structure to be converted.

***cstring*** A pointer to a buffer to receive a converted string.

***cstrsize*** The size of the buffer to receive the converted string.

**Returns**

The pointer to the returned string. This is the cstring argument parameter value.

#### 2.4.2.9 const char* rtUCSToNewCString (ASN1UniversalString * *pUCSString*)

This function converts a Universal 32-bit string into a null-terminated C string. Any characters that are not 8-bit characters are discarded. This function allocates dynamic memory to hold the converted string using the standard C run-time malloc function. The user is responsible for freeing this memory.

**Parameters**

***pUCSString*** A pointer to a Universal 32-bit string structure to be converted.

**Returns**

A pointer to allocated null-terminated string. The user is responsible for freeing this memory.

#### 2.4.2.10 const char* rtUCSToNewCStringEx (OSCTXT * *pctxt*, ASN1UniversalString * *pUCSString*)

This function converts a Universal 32-bit string into a null-terminated C string. Any characters that are not 8-bit characters are discarded. In contrast to rtUSCToNewCString this function allocates dynamic memory to hold the converted string using the rtMemAlloc function. The rtMemFreePtr should be called to release the allocated memory or the rtMemFree function should be called to release all memory allocated using the specified context block.

**Parameters**

***pctxt*** A pointer to a context block.

***pUCSString*** A pointer to a Universal 32-bit string structure to be converted.

**Returns**

A pointer to allocated null-terminated string. The user is responsible for freeing this memory.

#### 2.4.2.11 wchar_t* rtUCSToWCSString (ASN1UniversalString * *pUCSString*, wchar_t * *wcstring*, OSUINT32 *wcstrsize*)

This function converts a 32-bits encoded string to a wide character string.

**Parameters**

***pUCSString*** A pointer to a Universal string structure.

***wcstring*** The pointer to the buffer to receive the converted string.

***wcstrsize*** The number of wide characters (wchar_t) the outbuffer can hold.

**Returns**

A character count or error status. This will be negative if the conversion fails. If the result is positive, the number of characters was written to scstrsize.

#### 2.4.2.12 int rtUnivStrToUTF8 (OSCTXT * *pctxt*, const ASN1UniversalString * *pUnivStr*, OSOCTET * *outbuf*, size_t *outbufsiz*)

This function converts an ASN.1 Universal String type (32-bit characters) to UTF-8.

**Parameters**

***pctxt*** A pointer to a context structure.

***pUnivStr*** Pointer to universal string to be converted.

***outbuf*** Output buffer to receive UTF-8 characters.

***outbufsiz*** Output buffer size in bytes.

**Returns**

Zero if conversion was successful, a negative status code if failed.

#### 2.4.2.13 int rtUTF8StrnToASN1DynBitStr (OSCTXT * *pctxt*, const OSUTF8CHAR * *utf8str*, size_t *nbytes*, ASN1DynBitStr * *pvalue*)

This function converts the given part of UTF-8 string to a bit string value. The string consists of a series of '1' and '0' characters. This is the dynamic version in which memory is allocated for the returned binary string variable. Bits are stored from MSB to LSB order.

**Parameters**

***pctxt*** Pointer to context block structure.

***utf8str*** UTF-8 string to convert. Not necessary to be null-terminated.

***nbytes*** Size in bytes of utf8Str.

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

**Returns**

Completion status of operation:

- 0 = success,
- negative return value is error.

#### 2.4.2.14 int rtUTF8StrToASN1DynBitStr (OSCTXT * *pctxt*, const OSUTF8CHAR * *utf8str*, ASN1DynBitStr * *pvalue*)

This function converts the given null-terminated UTF-8 string to a bit string value. The string consists of a series of '1' and '0' characters. This is the dynamic version in which memory is allocated for the returned binary string variable. Bits are stored from MSB to LSB order.

**Parameters**

***pctxt*** Pointer to context block structure.

***utf8str*** Null-terminated UTF-8 string to convert

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

**Returns**

Completion status of operation:

- 0 = success,
- negative return value is error.

#### 2.4.2.15 int rtValidateChars (ASN1TAG *tag*, const char * *pdata*, size_t *nchars*)

This function ensures that a given string does not contain invalid characters.

**Parameters**

***tag*** The ASN.1 Tag that identifies the string.

***pdata*** A pointer to the character string to be examined.

***nchars*** The number of characters in pdata.

**Returns**

This function returns 0 if the string validates or the tag is not associated with a string; it otherwise returns the integer value of the character that invalidates the string.

#### 2.4.2.16 int rtValidateStr (ASN1TAG *tag*, const char * *pdata*)

This function ensures that a given string does not contain invalid characters.

**Parameters**

***tag*** The ASN.1 Tag that identifies the string.

***pdata*** A pointer to the character string to be examined.

**Returns**

This function returns 0 if the string validates or the tag is not associated with a string; it otherwise returns the integer value of the character that invalidates the string.

#### 2.4.2.17 ASN1UniversalString* rtWCSToUCSString (OSCTXT * *pctxt*, wchar_t * *wcstring*, ASN1UniversalString * *pUCSString*, Asn132BitCharSet * *pCharSet*)

This function converts a wide-character string to a Universal 32-bits encoded string.

**Parameters**

***pctxt*** A pointer to a context structure.

***wcstring*** The pointer to the wide-character (Unicode) string to convert

***pUCSString*** The pointer to the Universal String structure to receive the converted string.

***pCharSet*** The pointer to the character set structure describing the character set currently associated with the Universal character string type.

**Returns**

If the conversion of the WCS to the UTF-8 was successful, the number of bytes in the converted string is returned. If the encoding fails, a negative status value is returned.

## 2.5 Binary Coded Decimal (BCD) Helper Functions

### Defines

- #define rtQ825TBCDToString(numocts, data, buffer, bufsiz) rtxQ825TBCDToString(numocts, data, buffer, bufsiz)
- #define rtDecQ825TBCDString(pctxt, numocts, buffer, bufsiz) rtxDecQ825TBCDString(pctxt, numocts, buffer, bufsiz)
- #define rtEncQ825TBCDString(pctxt, str) rtxEncQ825TBCDString(pctxt, str)
- #define rtTBCDBinToChar(bcdDigit, pdigit) rtxTBCDBinToChar(bcdDigit, pdigit)
- #define rtTBCDCharToBin(digit, pbyte) rtxTBCDCharToBin(digit, pbyte)

### Functions

- const char * rtBCDToString (OSUINT32 numocts, const OSOCTET *data, char *buffer, size_t bufsiz, OSBOOL isTBCD)
- int rtStringToBCD (const char *str, OSOCTET *bcdStr, size_t bufsiz, OSBOOL isTBCD)
- int rtStringToDynBCD (OSCTXT *pctxt, const char *str, ASN1DynOctStr *poctstr)
- int rtStringToTBCD (const char *str, OSOCTET *bcdStr, size_t bufsiz)
- const char * rtTBCDToString (OSUINT32 numocts, const OSOCTET *data, char *buffer, size_t bufsiz)

### 2.5.1 Detailed Description

Binary Coded Decimal (BCD) helper functions provide assistance in working with BCD numbers. Functions are provided to convert to a BCD values to and from string form.

### 2.5.2 Define Documentation

#### 2.5.2.1 #define rtDecQ825TBCDString(pctxt, numocts, buffer, bufsiz) rtxDecQ825TBCDString(pctxt, numocts, buffer, bufsiz)

This function decodes a Q.825 TBCD value to a standard null-terminated string. TBCD digits are read from the decode buffer/stream and converted to their character equivalents. See 'rtQ825TBCDToString' for a description of the Q.825 TBCD format.

**Parameters**

***pctxt*** Pointer to a context structure block.

***numocts*** The number of octets in the BCD value to be read from input stream.

***buffer*** The destination buffer. Should not be less than bufsiz argument is.

***bufsiz*** The size of the destination buffer (in octets). The buffer size should be at least ((numocts * 2) + 1) to hold the BCD to String conversion.

**Returns**

Status of conversion: 0 = success, negative = error.

**Since**

6.6

#### 2.5.2.2 #define rtEncQ825TBCDString(pctxt, str) rtxEncQ825TBCDString(pctxt, str)

This function encodes a null-terminated string Q.825 TBCD string. TBCD digits are converted and written to the encode buffer/stream. See 'rtQ825TBCDToString' for a description of the Q.825 TBCD format.

**Parameters**

***pctxt*** Pointer to a context structure block.

***str*** Null-terminate string to be encoded. This string may only contain valid Q.825 TBCD characters.

**Returns**

Status of operation: 0 = success, negative = error.

**Since**

6.6

#### 2.5.2.3 #define rtQ825TBCDToString(numocts, data, buffer, bufsiz) rtxQ825TBCDToString(numocts, data, buffer, bufsiz)

This function converts a Q.825 TBCD value to a standard null-terminated string. Octet values can contain the filler digit to represent the odd number of BCD digits.

The encoding is as follows per Q.825:

This type (Telephony Binary Coded Decimal String) is used to represent digits from 0 through 9, $*$, #, a, b, c, two digits per octet, each digit encoded 0000 to 1001 (0 to 9), 1010 ($*$) 1011(#), 1100 (a), 1101 (b) or 1110 (c); 1111 (end of pulsing signal-ST); 0000 is used as a filler when there is an odd number of digits.

**Parameters**

***numocts*** The number of octets in the BCD value.

***data*** The pointer to the BCD value.

***buffer*** The destination buffer. Should not be less than bufsiz argument is.

***bufsiz*** The size of the destination buffer (in octets). The buffer size should be at least ((numocts $*$ 2) + 1) to hold the BCD to String conversion.

**Returns**

Status of conversion: 0 = success, negative = error.

**Since**

6.6

#### 2.5.2.4 #define rtTBCDBinToChar(bcdDigit, pdigit) rtxTBCDBinToChar(bcdDigit, pdigit)

This function converts a TBCD binary character into its ASCII equivalent.

**Parameters**

***bcdDigit*** TBCD digit

***pdigit*** Pointer to character to receive converted character

**Returns**

0 if conversion successful, or negative error code

**Since**

6.6

#### 2.5.2.5 #define rtTBCDCharToBin(digit, pbyte) rtxTBCDCharToBin(digit, pbyte)

This function converts a TBCD character ('0'-'9',"∗#abc") into its binary equivalent.

**Parameters**

***digit*** TBCD digit character ('0'-'9',"∗#abc")

***pbyte*** Pointer to byte to receive binary result.

**Returns**

0 if conversion successful, or negative error code

**Since**

6.6

### 2.5.3 Function Documentation

#### 2.5.3.1 const char∗ rtBCDToString (OSUINT32 *numocts*, const OSOCTET ∗ *data*, char ∗ *buffer*, size_t *bufsiz*, OSBOOL *isTBCD*)

This function converts a packed BCD value to a standard null-terminated string. Octet values may contain filler digits if the number of BCD digits is odd.

BCD digits can be 0(0000) to 9(1001). Filler digits can be A(1010), B(1011), C(1100), D(1101), E(1110) or F(1111)

**Parameters**

***numocts*** The number of octets in the BCD value.

***data*** The pointer to the BCD value.

***buffer*** The destination buffer. Should not be less than buffsiz argument is.

***bufsiz*** The size of the destination buffer (in octets). The buffer size should be atleast ((numocts ∗ 2) + 1) to hold the BCD to String conversion.

***isTBCD*** Whether the input data is formatted as a TBCD string or not.

**Returns**

The converted null-terminated string. NULL, if error has occurred or destination buffer is not enough.

**See also**

rtTBCDToString

#### 2.5.3.2 int rtStringToBCD (const char * *str*, OSOCTET * *bcdStr*, size_t *bufsiz*, OSBOOL *isTBCD*)

This function converts a standard null-terminated string into a BCD value. The source string should contain only characters '0' - '9', 'A' - 'F', or 'a' - 'f'. Otherwise, an error will occur.

**Parameters**

***str*** The source standard null-terminated string.

***bcdStr*** The destination buffer. Should not be less than bufsiz is.

***bufsiz*** The size of the destination buffer (in octets).

***isTBCD*** Whether the string is a TBCD string or not.

**Returns**

The number of octets in the resulting BCD value or a negative value if an error occurs.

**See also**

rtStringToTBCD

#### 2.5.3.3 int rtStringToDynBCD (OSCTXT * *pctxt*, const char * *str*, ASN1DynOctStr * *poctstr*)

This function converts a standard null-terminated string into a BCD value. The source string should contain only characters '0' - '9', 'A' - 'F', or 'a' - 'f'. Otherwise, an error will occur.

**Parameters**

***str*** The source standard null-terminated string.

***pctxt*** Pointer to a context structure block.

***poctstr*** Pointer to a dynamic octet string variable. Memory will be allocated for the data member of this structure with rtMemAlloc.

**Returns**

The number of octets in the resulting BCD value or a negative value if an error occurs.

#### 2.5.3.4 int rtStringToTBCD (const char * *str*, OSOCTET * *bcdStr*, size_t *bufsiz*)

This function converts a standard null-terminated string into a TBCD value. The source string should contain only characters '0' - '9', 'A' - 'F', or 'a' - 'f'. Otherwise, an error will occur. A TBCD string differs from a normal BCD string in that its bytes are flipped. The BCD string 0x12345f would be represented 0x2143f5 as a TBCD string.

**Parameters**

***str*** The source standard null-terminated string.

***bcdStr*** The destination buffer. Should not be less than bufsiz is.

***bufsiz*** The size of the destination buffer (in octets).

**Returns**

The number of octets in the resulting BCD value or a negative value if an error occurs.

**Since**

6.06

#### 2.5.3.5 const char* rtTBCDToString (OSUINT32 *numocts*, const OSOCTET * *data*, char * *buffer*, size_t *bufsiz*)

This function converts a packed TBCD value to a standard null-terminated string. Octet value can contain the filler digit to represent the odd number of BCD digits. A TBCD string differs from a normal BCD string in that the byte values of the octets are flipped. The BCD string 0x12345f would be represented 0x2143f5 as a TBCD string.

TBCD digits can be 0(0000) to 9(1001). Filler digits can be A(1010), B(1011), C(1100), D(1101), E(1110) or F(1111)

**Parameters**

***numocts*** The number of octets in the BCD value.

***data*** The pointer to the BCD value.

***buffer*** The destination buffer. Should not be less than buffsiz argument is.

***bufsiz*** The size of the destination buffer (in octets). The buffer size should be atleast ((numocts * 2) + 1) to hold the BCD to String conversion.

**Returns**

The converted null-terminated string. NULL, if error has occurred or destination buffer is not enough.

**Since**

6.06

## 2.6 Comparison Functions

### Defines

- #define rtCmpOID rtCmpOIDValue
- #define rtCmpOID64 rtCmpOID64Value

### Functions

- OSBOOL rtCmpBoolean (const char *name, OSBOOL value, OSBOOL compValue, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpInt8 (const char *name, OSINT8 value, OSINT8 compValue, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpSInt (const char *name, OSINT16 value, OSINT16 compValue, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpUInt8 (const char *name, OSUINT8 value, OSUINT8 compValue, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpUSInt (const char *name, OSUINT16 value, OSUINT16 compValue, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpInteger (const char *name, OSINT32 value, OSINT32 compValue, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpUnsigned (const char *name, OSUINT32 value, OSUINT32 compValue, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpInt64 (const char *name, OSINT64 value, OSINT64 compValue, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpUInt64 (const char *name, OSUINT64 value, OSUINT64 compValue, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpBitStr (const char *name, OSSIZE numbits, const OSOCTET *data, OSSIZE compNumbits, const OSOCTET *compData, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpOctStr (const char *name, OSSIZE numocts, const OSOCTET *data, OSSIZE compNumocts, const OSOCTET *compData, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpCharStr (const char *name, const char *cstring, const char *compCstring, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmp16BitCharStr (const char *name, Asn116BitCharString *bstring, Asn116BitCharString *compBstring, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmp32BitCharStr (const char *name, Asn132BitCharString *bstring, Asn132BitCharString *compBstring, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpReal (const char *name, OSREAL value, OSREAL compValue, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpOIDValue (const char *name, ASN1OBJID *pOID, ASN1OBJID *pcompOID, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpOID64Value (const char *name, ASN1OID64 *pOID, ASN1OID64 *pcompOID, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpOpenType (const char *name, OSSIZE numocts, const OSOCTET *data, OSSIZE compNumocts, const OSOCTET *compData, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpOpenTypeExt (const char *name, OSRTDList *pElemList, OSRTDList *pCompElemList, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpTag (const char *name, int tag, int compTag, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpSeqOfElements (const char *name, OSSIZE noOfElems, OSSIZE compNoOfElems, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpOptional (const char *name, unsigned presentBit, unsigned compPresentBit, char *errBuff, OSSIZE errBuffSize)

### 2.6.1 Detailed Description

The group of functions allows comparing the values of primitive ASN.1 types. These functions are used in the comparison routines that are generated by the ASN1C complier when the *-gencompare* option is used.

Information on elements that do not match is written to the given error buffer for each function. This makes it possible to compare complex structures and get back specific information as to what elements within those structures are different.

### 2.6.2 Define Documentation

#### 2.6.2.1 #define rtCmpOID rtCmpOIDValue

For backwards compatibility, we define rtCmpOID to be the same as rtCmpOIDValue.

#### 2.6.2.2 #define rtCmpOID64 rtCmpOID64Value

For backwards compatibility, we define rtCmpOID64 to be the same as rtCmpOID64Value.

### 2.6.3 Function Documentation

#### 2.6.3.1 OSBOOL rtCmp16BitCharStr (const char * *name*, Asn116BitCharString * *bstring*, Asn116BitCharString * *compBstring*, char * *errBuff*, OSSIZE *errBuffSize*)

The rtCmp16BitCharStr function compares two ASN.1 16-bit character string values (including BMPString).

**Parameters**

***name*** The name of value. Used for constructing errBuff if values are not matching.

***bstring*** The pointer to the first 16-bit character string structure to compare.

***compBstring*** The pointer to the second 16-bit character string structure to compare.

***errBuff*** The pointer to the buffer to receive the null-terminated string if the comparison fails. If the comparison failed, this buffer will contain the null-terminated string that explains the reason of comparison failure.

***errBuffSize*** The size of the errBuff buffer.

**Returns**

The comparison result. TRUE, if values matched, otherwise FALSE.

#### 2.6.3.2 OSBOOL rtCmp32BitCharStr (const char * *name*, Asn132BitCharString * *bstring*, Asn132BitCharString * *compBstring*, char * *errBuff*, OSSIZE *errBuffSize*)

The rtCmp32BitCharStr function compares two 32-bit character string values (including UniversalString).

**Parameters**

***name*** The name of value. Used for constructing errBuff if values are not matching.

***bstring*** The pointer to the first 32-bit character string structure to compare.

***compBstring*** The pointer to the second 32-bit character string structure to compare.

***errBuff*** The pointer to the buffer to receive the null-terminated string if the comparison fails. If the comparison failed, this buffer will contain the null-terminated string that explains the reason of comparison failure.

***errBuffSize*** The size of the errBuff buffer.

**Returns**

The comparison result. TRUE, if values matched, otherwise FALSE.

#### 2.6.3.3 OSBOOL rtCmpBitStr (const char * *name*, OSSIZE *numbits*, const OSOCTET * *data*, OSSIZE *compNumbits*, const OSOCTET * *compData*, char * *errBuff*, OSSIZE *errBuffSize*)

The rtCmpBitStr function compares two ASN.1 BIT STRING values.

**Parameters**

***name*** The name of value. Used for constructing errBuff if values are not matching.

***numbits*** The number of bits in the first value to compare.

***data*** The pointer to the data of the first value to compare.

***compNumbits*** The number of bits in the second value to compare.

***compData*** The pointer to the data of the second value to compare.

***errBuff*** The pointer to the buffer to receive the null-terminated string if the comparison fails. If the comparison failed, this buffer will contain the null-terminated string that explains the reason of comparison failure.

***errBuffSize*** The size of the errBuff buffer.

**Returns**

The comparison result. TRUE, if values matched, otherwise FALSE.

#### 2.6.3.4 OSBOOL rtCmpBoolean (const char * *name*, OSBOOL *value*, OSBOOL *compValue*, char * *errBuff*, OSSIZE *errBuffSize*)

The rtCmpBoolean function compares two ASN.1 Boolean values. The return value is TRUE (matched) or FALSE (unmatched).

**Parameters**

***name*** The name of value. Used for constructing errBuff if values are not matching.

***value*** First value to compare.

***compValue*** Second value to compare.

***errBuff*** The pointer to the buffer to receive the null-terminated string if the comparison fails. If the comparison failed, this buffer will contain the null-terminated string that explains the reason of comparison failure.

***errBuffSize*** The size of the errBuff buffer.

**Returns**

The comparison result. TRUE, if values matched, otherwise FALSE.

#### 2.6.3.5 OSBOOL rtCmpCharStr (const char * *name*, const char * *cstring*, const char * *compCstring*, char * *errBuff*, OSSIZE *errBuffSize*)

The rtCmpCharStr function compares two ASN.1 8-bit character sting values (including IA5String, VisibleString, PrintableString, NumericString, etc.)

**Parameters**

***name*** The name of value. Used for constructing errBuff if values are not matching.

***cstring*** The first standard null-terminated string to compare.

***compCstring*** The second standard null-terminated string to compare.

***errBuff*** The pointer to the buffer to receive the null-terminated string if the comparison fails. If the comparison failed, this buffer will contain the null-terminated string that explains the reason of comparison failure.

***errBuffSize*** The size of the errBuff buffer.

**Returns**

The comparison result. TRUE, if values matched, otherwise FALSE.

#### 2.6.3.6 OSBOOL rtCmpInt64 (const char * *name*, OSINT64 *value*, OSINT64 *compValue*, char * *errBuff*, OSSIZE *errBuffSize*)

The rtCompInt64 function compares two 64-bit ASN.1 INTEGER values.

**Parameters**

***name*** The name of value. Used for constructing errBuff if values are not matching.

***value*** First value to compare.

***compValue*** Second value to compare.

***errBuff*** The pointer to the buffer to receive the null-terminated string if the comparison fails. If the comparison failed, this buffer will contain the null-terminated string that explains the reason of comparison failure.

***errBuffSize*** The size of the errBuff buffer.

**Returns**

The comparison result. TRUE, if values matched, otherwise FALSE.

#### 2.6.3.7 OSBOOL rtCmpInt8 (const char * *name*, OSINT8 *value*, OSINT8 *compValue*, char * *errBuff*, OSSIZE *errBuffSize*)

The rtCmpInt8 function compares two ASN.1 8-bit integers. The return value is TRUE (matched) or FALSE (unmatched).

**Parameters**

***name*** The name of value. Used for constructing errBuff if values are not matching.

***value*** First value to compare.

***compValue*** Second value to compare.

***errBuff*** The pointer to the buffer to receive the null-terminated string if the comparison fails. If the comparison failed, this buffer will contain the null-terminated string that explains the reason of comparison failure.

***errBuffSize*** The size of the errBuff buffer.

**Returns**

The comparison result. TRUE, if values matched, otherwise FALSE.

#### 2.6.3.8 OSBOOL rtCmpInteger (const char * *name*, OSINT32 *value*, OSINT32 *compValue*, char * *errBuff*, OSSIZE *errBuffSize*)

The rtCmpInteger function compars two ASN.1 INTEGER values.

**Parameters**

***name*** The name of value. Used for constructing errBuff if values are not matching.

***value*** First value to compare.

***compValue*** Second value to compare.

***errBuff*** The pointer to the buffer to receive the null-terminated string if the comparison fails. If the comparison failed, this buffer will contain the null-terminated string that explains the reason of comparison failure.

***errBuffSize*** The size of the errBuff buffer.

**Returns**

The comparison result. TRUE, if values matched, otherwise FALSE.

#### 2.6.3.9 OSBOOL rtCmpOctStr (const char * *name*, OSSIZE *numocts*, const OSOCTET * *data*, OSSIZE *compNumocts*, const OSOCTET * *compData*, char * *errBuff*, OSSIZE *errBuffSize*)

The rtCmpOctStr function compares two ASN.1 OCTET STRING values.

**Parameters**

***name*** The name of value. Used for constructing errBuff if values are not matching.

***numocts*** The number of the octets in the first value to compare.

***data*** The pointer to the data of the first value to compare.

***compNumocts*** The number of the octets in the second value to compare.

***compData*** The pointer to the data of the second value to compare.

***errBuff*** The pointer to the buffer to receive the null-terminated string if the comparison fails. If the comparison failed, this buffer will contain the null-terminated string that explains the reason of comparison failure.

***errBuffSize*** The size of the errBuff buffer.

**Returns**

The comparison result. TRUE, if values matched, otherwise FALSE.

#### 2.6.3.10 OSBOOL rtCmpOID64Value (const char * *name*, ASN1OID64 * *pOID*, ASN1OID64 * *pcompOID*, char * *errBuff*, OSSIZE *errBuffSize*)

The rtCmpOID64Value function compares two 64-bit ASN.1 OBJECT IDENTIFIER or RELATIVE-OID values.

**Parameters**

***name*** The name of value. Used for constructing errBuff if values are not matching.

***pOID*** The pointer to the first value to compare.

***pcompOID*** The pointer to the second value to compare.

***errBuff*** The pointer to the buffer to receive the null-terminated string if the comparison fails. If the comparison failed, this buffer will contain the null-terminated string that explains the reason of comparison failure.

***errBuffSize*** The size of the errBuff buffer.

**Returns**

The comparison result. TRUE, if values matched, otherwise FALSE.

#### 2.6.3.11 OSBOOL rtCmpOIDValue (const char * *name*, ASN1OBJID * *pOID*, ASN1OBJID * *pcompOID*, char * *errBuff*, OSSIZE *errBuffSize*)

The rtCmpOIDValue function compares two ASN.1 OBJECT IDENTIFIER or REALTIVE-OID values.

**Parameters**

***name*** The name of value. Used for constructing errBuff if values are not matching.

***pOID*** The pointer to the first value to compare.

***pcompOID*** The pointer to the second value to compare.

***errBuff*** The pointer to the buffer to receive the null-terminated string if the comparison fails. If the comparison failed, this buffer will contain the null-terminated string that explains the reason of comparison failure.

***errBuffSize*** The size of the errBuff buffer.

**Returns**

The comparison result. TRUE, if values matched, otherwise FALSE.

#### 2.6.3.12 OSBOOL rtCmpOpenType (const char * *name*, OSSIZE *numocts*, const OSOCTET * *data*, OSSIZE *compNumocts*, const OSOCTET * *compData*, char * *errBuff*, OSSIZE *errBuffSize*)

The rtCmpOpenType function compares two ASN.1 values of the old (pre- 1994) ASN.1 ANY type or other elements defined in the later standards to be Open Types (for example, a variable type declaration in a CLASS construct defined in X.681).

**Parameters**

***name*** The name of value. Used for constructing errBuff if values are not matching.

***numocts*** The number of octets in the first value to compare.

***data*** The pointer to the data of the first value to compare.

***compNumocts*** The number of octets in the second value to compare.

***compData*** The pointer to the data of the second value to compare.

***errBuff*** The pointer to the buffer to receive the null-terminated string if the comparison fails. If the comparison failed, this buffer will contain the null-terminated string that explains the reason of comparison failure.

***errBuffSize*** The size of the errBuff buffer.

**Returns**

The comparison result. TRUE, if values matched, otherwise FALSE.

#### 2.6.3.13 OSBOOL rtCmpOpenTypeExt (const char * *name*, OSRTDList * *pElemList*, OSRTDList * *pCompElemList*, char * *errBuff*, OSSIZE *errBuffSize*)

The rtCmpOpenTypeExt function compares two ASN.1 open type extension values.

An open type extension is defined as an extensibility marker on a constructed type without any extension elements defined (for example, SEQUENCE {a INTEGER, ...}). The difference is that this is an implicit field that can span one or more elements whereas the standard Open Type is assumed to be a single tagged field.

**Parameters**

***name*** The name of value. Used for constructing errBuff if values are not matching.

***pElemList*** The first pointer to a linked list structure. The list should consist of ASN1OpenType elements.

***pCompElemList*** The second pointer to a linked list structure. The list should consist of ASN1OpenType elements.

***errBuff*** The pointer to the buffer to receive the null-terminated string if the comparison fails. If the comparison failed, this buffer will contain the null-terminated string that explains the reason of comparison failure.

***errBuffSize*** The size of the errBuff buffer.

**Returns**

The comparison result. TRUE, if values matched, otherwise FALSE.

#### 2.6.3.14 OSBOOL rtCmpOptional (const char * *name*, unsigned *presentBit*, unsigned *compPresentBit*, char * *errBuff*, OSSIZE *errBuffSize*)

The rtCmpOptional function compares two ASN.1 OPTIONAL bits. The return value is TRUE (matched) or FALSE (unmatched).

**Parameters**

***name*** The name of value. Used for constructing errBuff if values are not matching.

***presentBit*** First bit to compare.

***compPresentBit*** Second bit to compare.

***errBuff*** The pointer to the buffer to receive the null-terminated string if the comparison fails. If the comparison failed, this buffer will contain the null-terminated string that explains the reason of comparison failure.

***errBuffSize*** The size of the errBuff buffer.

**Returns**

The comparison result. TRUE, if values matched, otherwise FALSE.

#### 2.6.3.15 OSBOOL rtCmpReal (const char * *name*, OSREAL *value*, OSREAL *compValue*, char * *errBuff*, OSSIZE *errBuffSize*)

The rtCmpReal function compares two ASN.1 REAL values.

**Parameters**

***name*** The name of value. Used for constructing errBuff if values are not matching.

***value*** First value to compare.

***compValue*** Second value to compare.

***errBuff*** The pointer to the buffer to receive the null-terminated string if the comparison fails. If the comparison failed, this buffer will contain the null-terminated string that explains the reason of comparison failure.

***errBuffSize*** The size of the errBuff buffer.

**Returns**

The comparison result. TRUE, if values matched, otherwise FALSE.

#### 2.6.3.16 OSBOOL rtCmpSeqOfElements (const char * *name*, OSSIZE *noOfElems*, OSSIZE *compNoOfElems*, char * *errBuff*, OSSIZE *errBuffSize*)

This function compares two ASN.1 SEQUENCE OF sizes. The return value is TRUE (matched) or FALSE (unmatched).

**Parameters**

***name*** The name of value. Used for constructing errBuff if values are not matching.

***noOfElems*** First size value to compare.

***compNoOfElems*** Second size value to compare.

***errBuff*** The pointer to the buffer to receive the null-terminated string if the comparison fails. If the comparison failed, this buffer will contain the null-terminated string that explains the reason of comparison failure.

***errBuffSize*** The size of the errBuff buffer.

**Returns**

The comparison result. TRUE, if values matched, otherwise FALSE.

#### 2.6.3.17 OSBOOL rtCmpSInt (const char * *name*, OSINT16 *value*, OSINT16 *compValue*, char * *errBuff*, OSSIZE *errBuffSize*)

The rtCmpSInt function compares two ASN.1 16-bit integers. The return value is TRUE (matched) or FALSE (unmatched).

**Parameters**

***name*** The name of value. Used for constructing errBuff if values are not matching.

***value*** First value to compare.

***compValue*** Second value to compare.

***errBuff*** The pointer to the buffer to receive the null-terminated string if the comparison fails. If the comparison failed, this buffer will contain the null-terminated string that explains the reason of comparison failure.

***errBuffSize*** The size of the errBuff buffer.

**Returns**

The comparison result. TRUE, if values matched, otherwise FALSE.

#### 2.6.3.18 OSBOOL rtCmpTag (const char * *name*, int *tag*, int *compTag*, char * *errBuff*, OSSIZE *errBuffSize*)

The rtCmpTag function compares two ASN.1 tag values. The return value is TRUE (matched) or FALSE (unmatched).

**Parameters**

***name*** The name of value. Used for constructing errBuff if values are not matching.

***tag*** First tag value to compare.

***compTag*** Second tag value to compare.

***errBuff*** The pointer to the buffer to receive the null-terminated string if the comparison fails. If the comparison failed, this buffer will contain the null-terminated string that explains the reason of comparison failure.

***errBuffSize*** The size of the errBuff buffer.

**Returns**

The comparison result. TRUE, if values matched, otherwise FALSE.

#### 2.6.3.19 OSBOOL rtCmpUInt64 (const char * *name*, OSUINT64 *value*, OSUINT64 *compValue*, char * *errBuff*, OSSIZE *errBuffSize*)

The rtCmpUInt64 function compares two 64-bit ASN.1 unsigned INTEGER values.

**Parameters**

***name*** The name of value. Used for constructing errBuff if values are not matching.

***value*** First value to compare.

***compValue*** Second value to compare.

***errBuff*** The pointer to the buffer to receive the null-terminated string if the comparison fails. If the comparison failed, this buffer will contain the null-terminated string that explains the reason of comparison failure.

***errBuffSize*** The size of the errBuff buffer.

**Returns**

The comparison result. TRUE, if values matched, otherwise FALSE.

#### 2.6.3.20 OSBOOL rtCmpUInt8 (const char * *name*, OSUINT8 *value*, OSUINT8 *compValue*, char * *errBuff*, OSSIZE *errBuffSize*)

The rtCmpUInt8 function compares unsigned ASN.1 8-bit integers. The return value is TRUE (matched) or FALSE (unmatched).

**Parameters**

***name*** The name of value. Used for constructing errBuff if values are not matching.

***value*** First value to compare.

***compValue*** Second value to compare.

***errBuff*** The pointer to the buffer to receive the null-terminated string if the comparison fails. If the comparison failed, this buffer will contain the null-terminated string that explains the reason of comparison failure.

***errBuffSize*** The size of the errBuff buffer.

**Returns**

The comparison result. TRUE, if values matched, otherwise FALSE.

#### 2.6.3.21 OSBOOL rtCmpUnsigned (const char * *name*, OSUINT32 *value*, OSUINT32 *compValue*, char * *errBuff*, OSSIZE *errBuffSize*)

The rtCmpUnsigned function compares two ASN.1 unsigned INTEGER values.

**Parameters**

***name*** The name of value. Used for constructing errBuff if values are not matching.

***value*** First value to compare.

***compValue*** Second value to compare.

***errBuff*** The pointer to the buffer to receive the null-terminated string if the comparison fails. If the comparison failed, this buffer will contain the null-terminated string that explains the reason of comparison failure.

***errBuffSize*** The size of the errBuff buffer.

**Returns**

The comparison result. TRUE, if values matched, otherwise FALSE.

#### 2.6.3.22 OSBOOL rtCmpUSInt (const char * *name*, OSUINT16 *value*, OSUINT16 *compValue*, char * *errBuff*, OSSIZE *errBuffSize*)

The rtCmpUSInt function compares ASN.1 unsigned 16-bit integers. The return value is TRUE (matched) or FALSE (unmatched).

**Parameters**

***name*** The name of value. Used for constructing errBuff if values are not matching.

***value*** First value to compare.

***compValue*** Second value to compare.

***errBuff*** The pointer to the buffer to receive the null-terminated string if the comparison fails. If the comparison failed, this buffer will contain the null-terminated string that explains the reason of comparison failure.

***errBuffSize*** The size of the errBuff buffer.

**Returns**

The comparison result. TRUE, if values matched, otherwise FALSE.

## 2.7 Comparison to Standard Output Functions

### Functions

- OSBOOL rtCmpToStdoutBoolean (const char *name, OSBOOL value, OSBOOL compValue)
- OSBOOL rtCmpToStdoutInteger (const char *name, OSINT32 value, OSINT32 compValue)
- OSBOOL rtCmpToStdoutInt64 (const char *name, OSINT64 value, OSINT64 compValue)
- OSBOOL rtCmpToStdoutUnsigned (const char *name, OSUINT32 value, OSUINT32 compValue)
- OSBOOL rtCmpToStdoutUInt64 (const char *name, OSUINT64 value, OSUINT64 compValue)
- OSBOOL rtCmpToStdoutBitStr (const char *name, OSSIZE numbits, const OSOCTET *data, OSSIZE compNumbits, const OSOCTET *compData)
- OSBOOL rtCmpToStdoutOctStr (const char *name, OSSIZE numocts, const OSOCTET *data, OSSIZE compNumocts, const OSOCTET *compData)
- OSBOOL rtCmpToStdoutCharStr (const char *name, const char *cstring, const char *compCstring)
- OSBOOL rtCmpToStdout16BitCharStr (const char *name, Asn116BitCharString *bstring, Asn116BitCharString *compBstring)
- OSBOOL rtCmpToStdout32BitCharStr (const char *name, Asn132BitCharString *bstring, Asn132BitCharString *compBstring)
- OSBOOL rtCmpToStdoutReal (const char *name, OSREAL value, OSREAL compValue)
- OSBOOL rtCmpToStdoutOID (const char *name, ASN1OBJID *pOID, ASN1OBJID *pcompOID)
- OSBOOL rtCmpToStdoutOIDValue (const char *name, ASN1OBJID *pOID, ASN1OBJID *pcompOID)
- OSBOOL rtCmpToStdoutOID64 (const char *name, ASN1OID64 *pOID, ASN1OID64 *pcompOID)
- OSBOOL rtCmpToStdoutOID64Value (const char *name, ASN1OID64 *pOID, ASN1OID64 *pcompOID)
- OSBOOL rtCmpToStdoutOpenType (const char *name, OSSIZE numocts, const OSOCTET *data, OSSIZE compNumocts, const OSOCTET *compData)
- OSBOOL rtCmpToStdoutOpenTypeExt (const char *name, OSRTDList *pElemList, OSRTDList *pCompElemList)
- OSBOOL rtCmpToStdoutTag (const char *name, int tag, int compTag)
- OSBOOL rtCmpToStdoutSeqOfElements (const char *name, OSSIZE noOfElems, OSSIZE compNoOfElems)
- OSBOOL rtCmpToStdoutOptional (const char *name, unsigned presentBit, unsigned compPresentBit)

### 2.7.1 Detailed Description

cmp

The rtCmpToStdout<type> functions do the same actions as the rtCmp<type> ones, but they print the comparison results to stdout instead of putting it into the buffer. The prototypes of these functions are almost the same as for the rtCmp<type> except the last two parameters - they re absent in the rtCmpToStdout<type> functions.

### 2.7.2 Function Documentation

#### 2.7.2.1 OSBOOL rtCmpToStdout16BitCharStr (const char * *name*, Asn116BitCharString * *bstring*, Asn116BitCharString * *compBstring*)

**Parameters**

***name*** The name of value.

***bstring*** The first value to compare.

***compBstring*** The second value to compare.

#### 2.7.2.2 OSBOOL rtCmpToStdout32BitCharStr (const char * *name*, Asn132BitCharString * *bstring*, Asn132BitCharString * *compBstring*)

**Parameters**

***name*** The name of value.

***bstring*** The first value to compare.

***compBstring*** The second value to compare.

#### 2.7.2.3 OSBOOL rtCmpToStdoutBitStr (const char * *name*, OSSIZE *numbits*, const OSOCTET * *data*, OSSIZE *compNumbits*, const OSOCTET * *compData*)

**Parameters**

***name*** The name of value.

***numbits*** The first value to compare.

***data*** The pointer to the first value.

***compNumbits*** The second value to compare.

***compData*** The pointer to the second value.

#### 2.7.2.4 OSBOOL rtCmpToStdoutBoolean (const char * *name*, OSBOOL *value*, OSBOOL *compValue*)

**Parameters**

***name*** The name of value.

***value*** The first value to compare.

***compValue*** The second value to compare.

#### 2.7.2.5 OSBOOL rtCmpToStdoutCharStr (const char * *name*, const char * *cstring*, const char * *compCstring*)

**Parameters**

***name*** The name of value.

***cstring*** The first value to compare.

***compCstring*** The second value to compare.

#### 2.7.2.6 OSBOOL rtCmpToStdoutInt64 (const char * *name*, OSINT64 *value*, OSINT64 *compValue*)

**Parameters**

***name*** The name of value.

***value*** The first value to compare.

***compValue*** The second value to compare.

#### 2.7.2.7 OSBOOL rtCmpToStdoutInteger (const char * *name*, OSINT32 *value*, OSINT32 *compValue*)

**Parameters**

***name*** The name of value.

***value*** The first value to compare.

***compValue*** The second value to compare.

#### 2.7.2.8 OSBOOL rtCmpToStdoutOctStr (const char * *name*, OSSIZE *numocts*, const OSOCTET * *data*, OSSIZE *compNumocts*, const OSOCTET * *compData*)

**Parameters**

***name*** The name of value.

***numocts*** The first value to compare.

***data*** The pointer to the data of the first value.

***compNumocts*** The second value to compare.

***compData*** The pointer to the data of the second value.

#### 2.7.2.9 OSBOOL rtCmpToStdoutOID (const char * *name*, ASN1OBJID * *pOID*, ASN1OBJID * *pcompOID*)

**Parameters**

***name*** The name of value.

***pOID*** The first value to compare.

***pcompOID*** The second value to compare.

#### 2.7.2.10 OSBOOL rtCmpToStdoutOID64 (const char * *name*, ASN1OID64 * *pOID*, ASN1OID64 * *pcompOID*)

**Parameters**

***name*** The name of value.

***pOID*** The first value to compare.

***pcompOID*** The second value to compare.

#### 2.7.2.11 OSBOOL rtCmpToStdoutOID64Value (const char * *name*, ASN1OID64 * *pOID*, ASN1OID64 * *pcompOID*)

**Parameters**

***name*** The name of value.

***pOID*** The first value to compare.

***pcompOID*** The second value to compare.

#### 2.7.2.12 OSBOOL rtCmpToStdoutOIDValue (const char * *name*, ASN1OBJID * *pOID*, ASN1OBJID * *pcompOID*)

**Parameters**

***name*** The name of value.

***pOID*** The first value to compare.

***pcompOID*** The second value to compare.

#### 2.7.2.13 OSBOOL rtCmpToStdoutOpenType (const char * *name*, OSSIZE *numocts*, const OSOCTET * *data*, OSSIZE *compNumocts*, const OSOCTET * *compData*)

**Parameters**

***name*** The name of value.

***numocts*** The number of octets in the first value to compare.

***data*** The pointer to the data in the frist value to compare.

***compNumocts*** The number of octets in the second value to compare.

***compData*** The pointer to the data in the second value to compare.

#### 2.7.2.14 OSBOOL rtCmpToStdoutOpenTypeExt (const char * *name*, OSRTDList * *pElemList*, OSRTDList * *pCompElemList*)

**Parameters**

***name*** The name of value.

***pElemList*** The first value to compare.

***pCompElemList*** The second value to compare.

#### 2.7.2.15 OSBOOL rtCmpToStdoutOptional (const char * *name*, unsigned *presentBit*, unsigned *compPresentBit*)

**Parameters**

***name*** The name of value.

***presentBit*** The first value to compare.

***compPresentBit*** The second value to compare.

#### 2.7.2.16 OSBOOL rtCmpToStdoutReal (const char * *name*, OSREAL *value*, OSREAL *compValue*)

**Parameters**

***name*** The name of value.

***value*** The first value to compare.

***compValue*** The second value to compare.

#### 2.7.2.17 OSBOOL rtCmpToStdoutSeqOfElements (const char * *name*, OSSIZE *noOfElems*, OSSIZE *compNoOfElems*)

**Parameters**

***name*** The name of value.

***noOfElems*** The first value to compare.

***compNoOfElems*** The second value to compare.

#### 2.7.2.18 OSBOOL rtCmpToStdoutTag (const char * *name*, int *tag*, int *compTag*)

**Parameters**

***name*** The name of value.

***tag*** The first value to compare.

***compTag*** The second value to compare.

#### 2.7.2.19 OSBOOL rtCmpToStdoutUInt64 (const char * *name*, OSUINT64 *value*, OSUINT64 *compValue*)

**Parameters**

***name*** The name of value.

***value*** The first value to compare.

***compValue*** The second value to compare.

#### 2.7.2.20 OSBOOL rtCmpToStdoutUnsigned (const char * *name*, OSUINT32 *value*, OSUINT32 *compValue*)

**Parameters**

***name*** The name of value.

***value*** The first value to compare.

***compValue*** The second value to compare.

## 2.8 Copy Functions

### Functions

- OSBOOL rtCopyBitStr (OSUINT32 srcNumbits, const OSOCTET *pSrcData, OSUINT32 *pDstNumbits, OSOCTET *pDstData)
- OSBOOL rtCopyDynBitStr (OSCTXT *pctxt, const ASN1DynBitStr *pSrcData, ASN1DynBitStr *pDstData)
- OSBOOL rtCopyOctStr (OSUINT32 srcNumocts, const OSOCTET *pSrcData, OSUINT32 *pDstNumocts, OSOCTET *pDstData)
- OSBOOL rtCopyDynOctStr (OSCTXT *pctxt, const ASN1DynOctStr *pSrcData, ASN1DynOctStr *pDstData)
- OSBOOL rtCopyCharStr (OSCTXT *pctxt, const char *srcStr, char **dstStr)
- OSBOOL rtCopy16BitCharStr (OSCTXT *pctxt, const Asn116BitCharString *srcStr, Asn116BitCharString *dstStr)
- OSBOOL rtCopy32BitCharStr (OSCTXT *pctxt, const Asn132BitCharString *srcStr, Asn132BitCharString *dstStr)
- OSBOOL rtCopyOID (const ASN1OBJID *srcOID, ASN1OBJID *dstOID)
- OSBOOL rtCopyOID64 (const ASN1OID64 *srcOID, ASN1OID64 *dstOID)
- OSBOOL rtCopyOpenType (OSCTXT *pctxt, const ASN1OpenType *srcOT, ASN1OpenType *dstOT)
- OSBOOL rtCopyOpenTypeExt (OSCTXT *pctxt, const OSRTDList *srcList, OSRTDList *dstList)

### 2.8.1 Detailed Description

This group of functions allows copying values of primitive ASN.1 types.

These functions are used in copy routines that are generated by the ASN.1 complier when *-gencopy* option is used. Some primitive types that are mapped onto C standard primitive types (such as BOOLEAN, INTEGER REAL) do not need copy functions. The standard assingment operator can be used to copy these types.

### 2.8.2 Function Documentation

#### 2.8.2.1 OSBOOL rtCopy16BitCharStr (OSCTXT * *pctxt*, const Asn116BitCharString * *srcStr*, Asn116BitCharString * *dstStr*)

The rtCopy16BitCharStr function copies one ASN.1 16-bit character string value to another (generally BMPString).

The return value is one of the TRUE (copied successfully) or FALSE (error has occurred).

**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.

***srcStr*** The pointer to the source 16-bit character string structure to copy.

***dstStr*** The pointer to destination 16-bit character string structure to receive the copied string. The memory will be allocated dynamically via call to rtxMemAlloc function.

**Returns**

The copying result. TRUE, if success, otherwise FALSE.

#### 2.8.2.2 OSBOOL rtCopy32BitCharStr (OSCTXT * *pctxt*, const Asn132BitCharString * *srcStr*, Asn132BitCharString * *dstStr*)

The rtCopy32BitCharStr function copies one ASN.1 32-bit character string value to another (generally UniversalString).

The return value is one of the TRUE (copied successfully) or FALSE (error has occurred).

**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.

***srcStr*** The pointer to the source 32-bit character string structure to copy.

***dstStr*** The pointer to destination 32-bit character string structure to receive the copied string. The memory will be allocated dynamically via call to rtxMemAlloc function.

**Returns**

The copying result. TRUE, if success, otherwise FALSE.

#### 2.8.2.3 OSBOOL rtCopyBitStr (OSUINT32 *srcNumbits*, const OSOCTET * *pSrcData*, OSUINT32 * *pDstNumbits*, OSOCTET * *pDstData*)

The rtCopyBitStr function copies one ASN.1 BIT STRING value to another.

The return value is one of the TRUE (copied successfully) or FALSE (error has occurred).

**Parameters**

***srcNumbits*** The number of bits in the source value to copy.

***pSrcData*** The pointer to data of the source value to copy.

***pDstNumbits*** The pointer to destination number of bits. The srcNumbits argument will be copied into it.

***pDstData*** The pointer to the destination buffer to receive the copied data. The buffer is assumed to be already allocated or static and should be enough to receive the copying data.

**Returns**

The copying result. TRUE, if success, otherwise FALSE.

#### 2.8.2.4 OSBOOL rtCopyCharStr (OSCTXT * *pctxt*, const char * *srcStr*, char ** *dstStr*)

The rtCopyCharStr function copies one ASN.1 8-bit character string value to another (including IA5String, VisibleString, PrintableString, NumericString, etc).

The return value is one of the TRUE (copied successfully) or FALSE (error has occurred).

**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.

***srcStr*** The pointer to the source standard null-terminated string to copy.

***dstStr*** The pointer to pointer destination string to receive the copied string. The memory will be allocated dynamically via a call to rtxMemAlloc function.

**Returns**

The copying result. TRUE, if success, otherwise FALSE.

#### 2.8.2.5 OSBOOL rtCopyDynBitStr (OSCTXT * *pctxt*, const ASN1DynBitStr * *pSrcData*, ASN1DynBitStr * *pDstData*)

The rtCopyDynBitStr function is similar to the rtCopyBitStr, but it copies a dynamic ASN.1 BIT STRING value.

The return vale is one of the TRUE (copied successfully) or FALSE (error has occurred).

**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.

***pSrcData*** The pointer to data of the source value to copy.

***pDstData*** The pointer to the destination dynamic bit string structure to receive the copied data. The memory will be allocated dynamically via call to rtxMemAlloc function.

**Returns**

The copying result. TRUE, if success, otherwise FALSE.

#### 2.8.2.6 OSBOOL rtCopyDynOctStr (OSCTXT * *pctxt*, const ASN1DynOctStr * *pSrcData*, ASN1DynOctStr * *pDstData*)

The rtCopyDynOctStr funtion is similar to rtCopyOctStr, but it copies a dynamic ASN.1 OCTET STRING value.

The return value is one of the TRUE (copied successfully) or FALSE (error has occurred).

**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.

***pSrcData*** The pointer to the source dynamic octet string structure to copy.

***pDstData*** The point to destination dynamic octet string structure to receive the copied data. The memory will be allocated dynamically via a call to rtxMemAlloc function.

**Returns**

The copying result. TRUE, if success, otherwise FALSE.

#### 2.8.2.7 OSBOOL rtCopyOctStr (OSUINT32 *srcNumocts*, const OSOCTET * *pSrcData*, OSUINT32 * *pDstNumocts*, OSOCTET * *pDstData*)

The rtCopyOctStr function copies one ASN.1 OCTET STRING value to another.

The return value is one of the TRUE (copied successfully) or FALSE (error has occurred).

**Parameters**

***srcNumocts*** The number of octets in the source value to copy.

***pSrcData*** The pointer to data of the source value to copy.

***pDstNumocts*** The pointer to the destination number of octets. The srcNumocts argument will be copied into it.

***pDstData*** The pointer to the destination buffer to receive the copied data. The buffer is assumed to be already allocated or static and should be enough to receive the copying data.

**Returns**

The copying result. TRUE, if success, otherwise FALSE.

#### 2.8.2.8 OSBOOL rtCopyOID (const ASN1OBJID * *srcOID*, ASN1OBJID * *dstOID*)

The rtCopyIOD function copies one ASN.1 OBJECT IDENTIFIER or RELATED-IOD value to another.

The return value is one of the TRUE (copied successfully) or FALSE (error has occurred).

**Parameters**

***srcOID*** The pointer to the source object identifier structure to copy.

***dstOID*** The pointer to destination structure t receive the copied string.

**Returns**

The copying result. TRUE, if success, otherwise FALSE.

#### 2.8.2.9 OSBOOL rtCopyOID64 (const ASN1OID64 * *srcOID*, ASN1OID64 * *dstOID*)

The rtCopyOID64 function copies one 64-bit ASN.1 OBJECT IDENTIFIER or RELATIVE-OID value to another.

The return value is one of the TRUE (copied successfully) or FALSE (error has occurred).

**Parameters**

***srcOID*** The pointer to the source object identifier structure to copy.

***dstOID*** The pointer to destination structure t receive the copied string.

**Returns**

The copying result. TRUE, if success, otherwise FALSE.

#### 2.8.2.10 OSBOOL rtCopyOpenType (OSCTXT * *pctxt*, const ASN1OpenType * *srcOT*, ASN1OpenType * *dstOT*)

The rtCopyOpenType copies ASN.1 value of the old (pre- 1994) ASN.1 ANY type or other elements defined in the later standards to be Open Types (for example, a variable type declaration in a CLASS construct as defined in X.681).

The return value is one of the TRUE (copied successfully) or FALSE (error has occurred).

**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.

***srcOT*** The pointer to the source Open Type structure to copy.

***dstOT*** The pointer to the destination Open Type structure to receive the copied data. The memory will be allocated dynamically via call to the rtxMemAlloc function.

**Returns**

The copying result. TRUE, if success, otherwise FALSE.

#### 2.8.2.11 OSBOOL rtCopyOpenTypeExt (OSCTXT * *pctxt*, const OSRTDList * *srcList*, OSRTDList * *dstList*)

The rtCopyOpenTypeExt function copies an ASN.1 open type extension value.

The return value is one of the TRUE (copied successfully) or FALSE (error has occurred). An open type extension is defined as extensibility marker on a constructed type without any extension elements defined (for example, SEQUENCE { a INTEGER, ... }). The difference is that this is an implicit field that can span more elements whereas the standard Open Type is assumed to be a single tagged field.

**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.

***srcList*** The pointer to the source linked list structure to copy. The list should consist of ASN1OpenType elements.

***dstList*** The pointer to destination linked list structure to receive the copied data. The memory for list nodes and data will be allocated dynamically via call to the rtxMemAlloc function. The list nodes will contain the data of ASN1OpenType type.

## 2.9 Character string functions

### Functions

- int rtxStricmp (const char *str1, const char *str2)
- char * rtxStrcat (char *dest, size_t bufsiz, const char *src)
- char * rtxStrncat (char *dest, size_t bufsiz, const char *src, size_t nchars)
- char * rtxStrcpy (char *dest, size_t bufsiz, const char *src)
- char * rtxStrncpy (char *dest, size_t bufsiz, const char *src, size_t nchars)
- char * rtxStrdup (OSCTXT *pctxt, const char *src)
- const char * rtxStrJoin (char *dest, size_t bufsiz, const char *str1, const char *str2, const char *str3, const char *str4, const char *str5)
- const char * rtxStrDynJoin (OSCTXT *pctxt, const char *str1, const char *str2, const char *str3, const char *str4, const char *str5)
- int rtxIntToCharStr (OSINT32 value, char *dest, size_t bufsiz, char padchar)
- int rtxUIntToCharStr (OSUINT32 value, char *dest, size_t bufsiz, char padchar)
- int rtxInt64ToCharStr (OSINT64 value, char *dest, size_t bufsiz, char padchar)
- int rtxUInt64ToCharStr (OSUINT64 value, char *dest, size_t bufsiz, char padchar)
- int rtxSizeToCharStr (size_t value, char *dest, size_t bufsiz, char padchar)
- int rtxHexCharsToBinCount (const char *hexstr, size_t nchars)
- int rtxHexCharsToBin (const char *hexstr, size_t nchars, OSOCTET *binbuf, size_t bufsize)
- int rtxCharStrToInt (const char *cstr, OSINT32 *pvalue)
- int **rtxCharStrToInt8** (const char *cstr, OSINT8 *pvalue)
- int **rtxCharStrToInt16** (const char *cstr, OSINT16 *pvalue)
- int **rtxCharStrToUInt** (const char *cstr, OSUINT32 *pvalue)
- int **rtxCharStrToUInt8** (const char *cstr, OSUINT8 *pvalue)
- int **rtxCharStrToUInt16** (const char *cstr, OSUINT16 *pvalue)

### 2.9.1 Detailed Description

These functions are more secure versions of several of the character string functions available in the standard C run-time library.

### 2.9.2 Function Documentation

#### 2.9.2.1 int rtxCharStrToInt (const char * *cstr*, OSINT32 * *pvalue*)

This function converts the given character string to an integer value. It consumes all leading whitespace before the digits start. It then comsumes digits until a non-digit character is encountered.

**Parameters**

***cstr*** Character string to convert.

***pvalue*** Pointer to integer value to receive converted data.

**Returns**

Number of bytes or negative status value if fail.

#### 2.9.2.2 int rtxHexCharsToBin (const char * *hexstr*, size_t *nchars*, OSOCTET * *binbuf*, size_t *bufsize*)

This function converts the given hex string to binary. The result is stored in the given binary buffer. Any whitespace characters in the string are ignored.

**Parameters**

***hexstr*** Hex character string to convert.

***nchars*** Number of characters in string. If zero, characters are read up to null-terminator.

***binbuf*** Buffer to hold converted binary data.

***bufsize*** Size of the binary data buffer.

**Returns**

Number of bytes or negative status value if fail.

#### 2.9.2.3 int rtxHexCharsToBinCount (const char * *hexstr*, size_t *nchars*)

This function returns a count of the number of bytes the would result from the conversion of a hexadecimal character string to binary. Any whitespace characters in the string are ignored.

**Parameters**

***hexstr*** Hex character string to convert.

***nchars*** Number of characters in string. If zero, characters are read up to null-terminator.

**Returns**

Number of bytes or negative status value if fail.

#### 2.9.2.4 int rtxInt64ToCharStr (OSINT64 *value*, char * *dest*, size_t *bufsiz*, char *padchar*)

This function converts a signed 64-bit integer into a character string. It is similar to the C `itoa` function.

**Parameters**

***value*** Integer to convert.

***dest*** Pointer to destination buffer to receive string.

***bufsiz*** Size of the destination buffer.

***padchar*** Left pad char, set to zero for no padding.

**Returns**

Number of characters or negative status value if fail.

#### 2.9.2.5 int rtxIntToCharStr (OSINT32 *value*, char * *dest*, size_t *bufsiz*, char *padchar*)

This function converts a signed 32-bit integer into a character string. It is similar to the C `itoa` function.

**Parameters**

***value*** Integer to convert.

***dest*** Pointer to destination buffer to receive string.

***bufsiz*** Size of the destination buffer.

***padchar*** Left pad char, set to zero for no padding.

**Returns**

Number of characters or negative status value if fail.

#### 2.9.2.6 int rtxSizeToCharStr (size_t *value*, char * *dest*, size_t *bufsiz*, char *padchar*)

This function converts a value of type 'size_t' into a character string. It is similar to the C `itoa` function.

**Parameters**

***value*** Size value to convert.

***dest*** Pointer to destination buffer to receive string.

***bufsiz*** Size of the destination buffer.

***padchar*** Left pad char, set to zero for no padding.

**Returns**

Number of characters or negative status value if fail.

#### 2.9.2.7 char* rtxStrcat (char * *dest*, size_t *bufsiz*, const char * *src*)

This function concatanates the given string onto the string buffer. It is similar to the C `strcat` function except more secure because it checks for buffer overrun.

**Parameters**

***dest*** Pointer to destination buffer to receive string.

***bufsiz*** Size of the destination buffer.

***src*** Pointer to null-terminated string to copy.

**Returns**

Pointer to destination buffer or NULL if copy failed.

#### 2.9.2.8 char* rtxStrcpy (char * *dest*, size_t *bufsiz*, const char * *src*)

This function copies a null-terminated string to a target buffer. It is similar to the C `strcpy` function except more secure because it checks for buffer overrun.

**Parameters**

***dest*** Pointer to destination buffer to receive string.

***bufsiz*** Size of the destination buffer.

***src*** Pointer to null-terminated string to copy.

**Returns**

Pointer to destination buffer or NULL if copy failed.

#### 2.9.2.9 char* rtxStrdup (OSCTXT * *pctxt*, const char * *src*)

This function creates a duplicate copy of a null-terminated string. Memory is allocated for the target string using the rtxMemAlloc function. The string is then copied into this memory block. It is similar to the C `strdup` function except more secure because it checks for buffer overrun.

**Parameters**

***pctxt*** Pointer to a standard context structure.

***src*** Pointer to null-terminated string to copy.

**Returns**

Pointer to destination buffer or NULL if copy failed.

#### 2.9.2.10 const char* rtxStrDynJoin (OSCTXT * *pctxt*, const char * *str1*, const char * *str2*, const char * *str3*, const char * *str4*, const char * *str5*)

This function allocates memory for and concatanates up to five substrings together into a single string.

**Parameters**

***pctxt*** Pointer to a standard context structure.

***str1*** Pointer to substring to join.

***str2*** Pointer to substring to join.

***str3*** Pointer to substring to join.

***str4*** Pointer to substring to join.

***str5*** Pointer to substring to join.

**Returns**

Composite string consisting of all parts.

#### 2.9.2.11 int rtxStricmp (const char * *str1*, const char * *str2*)

This is an implementation of the non-standard stricmp function. It does not check for greater than/less than however, only for equality.

**Parameters**

***str1*** Pointer to first string to compare.

***str2*** Pointer to second string to compare.

**Returns**

0 if strings are equal, non-zero if not.

#### 2.9.2.12 const char* rtxStrJoin (char * *dest*, size_t *bufsiz*, const char * *str1*, const char * *str2*, const char * *str3*, const char * *str4*, const char * *str5*)

This function concatanates up to five substrings together into a single string.

**Parameters**

***dest*** Pointer to destination buffer to receive string.

***bufsiz*** Size of the destination buffer.

***str1*** Pointer to substring to join.

***str2*** Pointer to substring to join.

***str3*** Pointer to substring to join.

***str4*** Pointer to substring to join.

***str5*** Pointer to substring to join.

**Returns**

Composite string consisting of all parts.

#### 2.9.2.13 char* rtxStrncat (char * *dest*, size_t *bufsiz*, const char * *src*, size_t *nchars*)

This function concatanates the given number of characters from the given string onto the string buffer. It is similar to the C `strncat` function except more secure because it checks for buffer overrun.

**Parameters**

***dest*** Pointer to destination buffer to receive string.

***bufsiz*** Size of the destination buffer.

***src*** Pointer to null-terminated string to copy.

***nchars*** Number of characters to copy.

**Returns**

Pointer to destination buffer or NULL if copy failed.

#### 2.9.2.14 char* rtxStrncpy (char * *dest*, size_t *bufsiz*, const char * *src*, size_t *nchars*)

This function copies the given number of characters from a string to a target buffer. It is similar to the C `strncpy` function except more secure because it checks for buffer overrun and ensures a null-terminator is copied to the end of the target buffer if the given buffer size is greater than the number of characters being copied.

**Parameters**

***dest*** Pointer to destination buffer to receive string.

***bufsiz*** Size of the destination buffer.

***src*** Pointer to null-terminated string to copy.

***nchars*** Number of characters to copy.

**Returns**

Pointer to destination buffer or NULL if copy failed.

#### 2.9.2.15 int rtxUInt64ToCharStr (OSUINT64 *value*, char * *dest*, size_t *bufsiz*, char *padchar*)

This function converts an unsigned 64-bit integer into a character string. It is similar to the C `itoa` function.

**Parameters**

***value*** Integer to convert.

***dest*** Pointer to destination buffer to receive string.

***bufsiz*** Size of the destination buffer.

***padchar*** Left pad char, set to zero for no padding.

**Returns**

Number of characters or negative status value if fail.

#### 2.9.2.16 int rtxUIntToCharStr (OSUINT32 *value*, char * *dest*, size_t *bufsiz*, char *padchar*)

This function converts an unsigned 32-bit integer into a character string. It is similar to the C `itoa` function.

**Parameters**

***value*** Integer to convert.

***dest*** Pointer to destination buffer to receive string.

***bufsiz*** Size of the destination buffer.

***padchar*** Left pad char, set to zero for no padding.

**Returns**

Number of characters or negative status value if fail.

## 2.10 Date/time conversion functions

### Functions

- int rtxDateToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
- int rtxTimeToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
- int rtxDateTimeToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
- int rtxGYearToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
- int rtxGYearMonthToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
- int rtxGMonthToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
- int rtxGMonthDayToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
- int rtxGDayToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
- int rtxGetCurrDateTime (OSNumDateTime *pvalue)
- int rtxCmpDate (const OSNumDateTime *pvalue1, const OSNumDateTime *pvalue2)
- int rtxCmpDate2 (const OSNumDateTime *pvalue, OSINT32 year, OSUINT8 mon, OSUINT8 day, OSBOOL tzflag, OSINT32 tzo)
- int rtxCmpTime (const OSNumDateTime *pvalue1, const OSNumDateTime *pvalue2)
- int rtxCmpTime2 (const OSNumDateTime *pvalue, OSUINT8 hour, OSUINT8 min, OSREAL sec, OSBOOL tzflag, OSINT32 tzo)
- int rtxCmpDateTime (const OSNumDateTime *pvalue1, const OSNumDateTime *pvalue2)
- int rtxCmpDateTime2 (const OSNumDateTime *pvalue, OSINT32 year, OSUINT8 mon, OSUINT8 day, OSUINT8 hour, OSUINT8 min, OSREAL sec, OSBOOL tzflag, OSINT32 tzo)
- int rtxParseDateString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
- int rtxParseTimeString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
- int rtxParseDateTimeString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
- int rtxParseGYearString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
- int rtxParseGYearMonthString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
- int rtxParseGMonthString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
- int rtxParseGMonthDayString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
- int rtxParseGDayString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
- int rtxMSecsToDuration (OSINT32 msecs, OSUTF8CHAR *buf, OSUINT32 bufsize)
- int rtxDurationToMSecs (OSUTF8CHAR *buf, OSUINT32 bufsize, OSINT32 *msecs)
- int rtxSetDateTime (OSNumDateTime *pvalue, struct tm *timeStruct)
- int rtxSetLocalDateTime (OSNumDateTime *pvalue, time_t timeMs)
- int rtxSetUtcDateTime (OSNumDateTime *pvalue, time_t timeMs)
- int rtxGetDateTime (const OSNumDateTime *pvalue, time_t *timeMs)
- OSBOOL rtxDateIsValid (const OSNumDateTime *pvalue)
- OSBOOL rtxTimeIsValid (const OSNumDateTime *pvalue)
- OSBOOL rtxDateTimeIsValid (const OSNumDateTime *pvalue)

### 2.10.1 Detailed Description

These functions handle the conversion of date/time to and from various internal formats to XML schema standard string forms.

## 2.10.2 Function Documentation

### 2.10.2.1 int rtxCmpDate (const OSNumDateTime * *pvalue1*, const OSNumDateTime * *pvalue2*)

This function compares the date part of two OSNumDateTime structures and returns the result of the comparison.

**Parameters**

***pvalue1*** Pointer to OSNumDateTime structure.

***pvalue2*** Pointer to OSNumDateTime structure.

**Returns**

Completion status of operation:

- 0 Dates are same
- +1 First Date/Time is greater than second.
- -1 First Date/Time is less than second.

### 2.10.2.2 int rtxCmpDate2 (const OSNumDateTime * *pvalue*, OSINT32 *year*, OSUINT8 *mon*, OSUINT8 *day*, OSBOOL *tzflag*, OSINT32 *tzo*)

This function compares the date part of OSNumDateTime structure and date components, specified as parameters.

**Parameters**

***pvalue*** Pointer to OSNumDateTime structure.

***year*** Year (-inf..inf)

***mon*** Month (1..12)

***day*** Day (1..31)

***tzflag*** TRUE, if time zone offset is set (see tzo parameter).

***tzo*** Time zone offset (-840..840).

**Returns**

Completion status of operation:

- 0 Dates are same
- +1 First Date/Time is greater than second.
- -1 First Date/Time is less than second.

### 2.10.2.3 int rtxCmpDateTime (const OSNumDateTime * *pvalue1*, const OSNumDateTime * *pvalue2*)

This function compares two OSNumDateTime structures and returns the result of the comparison.

**Parameters**

***pvalue1*** Pointer to OSNumDateTime structure.

***pvalue2*** Pointer to OSNumDateTime structure.

**Returns**

Completion status of operation:

- 0 Dates are same
- +1 First Date/Time is greater than second.
- -1 First Date/Time is less than second.

#### 2.10.2.4 int rtxCmpDateTime2 (const OSNumDateTime * *pvalue*, OSINT32 *year*, OSUINT8 *mon*, OSUINT8 *day*, OSUINT8 *hour*, OSUINT8 *min*, OSREAL *sec*, OSBOOL *tzflag*, OSINT32 *tzo*)

This function compares the OSNumDateTime structure and dateTime components, specified as parameters.

**Parameters**

***pvalue*** Pointer to OSNumDateTime structure.

***year*** Year (-inf..inf)

***mon*** Month (1..12)

***day*** Day (1..31)

***hour*** Hour (0..23)

***min*** Minutes (0..59)

***sec*** Seconds (0.0..59.(9))

***tzflag*** TRUE, if time zone offset is set (see tzo parameter).

***tzo*** Time zone offset (-840..840).

**Returns**

Completion status of operation:

- 0 Dates are same
- +1 First Date/Time is greater than second.
- -1 First Date/Time is less than second.

#### 2.10.2.5 int rtxCmpTime (const OSNumDateTime * *pvalue1*, const OSNumDateTime * *pvalue2*)

This function compares the time part of two OSNumDateTime structures and returns the result of the comparison.

**Parameters**

***pvalue1*** Pointer to OSNumDateTime structure.

***pvalue2*** Pointer to OSNumDateTime structure.

**Returns**

Completion status of operation:

- 0 Times are same
- +1 First Date/Time is greater than second.
- -1 First Date/Time is less than second.

#### 2.10.2.6 int rtxCmpTime2 (const OSNumDateTime * *pvalue*, OSUINT8 *hour*, OSUINT8 *min*, OSREAL *sec*, OSBOOL *tzflag*, OSINT32 *tzo*)

This function compares the time part of OSNumDateTime structure and time components, specified as parameters.

**Parameters**

***pvalue*** Pointer to OSNumDateTime structure.

***hour*** Hour (0..23)

***min*** Minutes (0..59)

***sec*** Seconds (0.0..59.(9))

***tzflag*** TRUE, if time zone offset is set (see tzo parameter).

***tzo*** Time zone offset (-840..840).

**Returns**

Completion status of operation:

- 0 Times are same
- +1 First Date/Time is greater than second.
- -1 First Date/Time is less than second.

#### 2.10.2.7 OSBOOL rtxDateIsValid (const OSNumDateTime * *pvalue*)

This function verifies that date members (year, month, day, timezone) of the OSNumDateTime structure contains valid values.

**Parameters**

***pvalue*** Pointer to OSNumDateTime structure to be checked.

**Returns**

Boolean result: true means data is valid.

#### 2.10.2.8 OSBOOL rtxDateTimeIsValid (const OSNumDateTime * *pvalue*)

This function verifies that all members of the OSNumDateTime structure contains valid values.

**Parameters**

***pvalue*** Pointer to OSNumDateTime structure to be checked.

**Returns**

Boolean result: true means data is valid.

#### 2.10.2.9 int rtxDateTimeToString (const OSNumDateTime * *pvalue*, OSUTF8CHAR * *buffer*, size_t *bufsize*)

This function formats a numeric date/time value of all components in the OSNumDateTime structure into XML schema standard format (CCYY-MM-DDTHH:MM:SS[.frac][TZ]).

**Parameters**

***pvalue*** Pointer to OSNumDateTime structure containing date/time components to be formatted.

***buffer*** Buffer into which date is to be formatted. This is a fixed-sized buffer. The user must provide a buffer that is large enough to hold the formatted time string.

***bufsize*** Size of the buffer to receive the formatted date.

**Returns**

Completion status of operation:

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

#### 2.10.2.10 int rtxDateToString (const OSNumDateTime * *pvalue*, OSUTF8CHAR * *buffer*, size_t *bufsize*)

This function formats a numeric date value consisting of individual date components (year, month, day) into XML schema standard format (CCYY-MM-DD).

**Parameters**

***pvalue*** Pointer to OSNumDateTime structure containing date components to be formatted.

***buffer*** Buffer into which date is to be formatted. This is a fixed-sized buffer. The user must provide a buffer that is at least nine bytes long to hold the formatted date and a null-terminator character.

***bufsize*** Size of the buffer to receive the formatted date.

**Returns**

Completion status of operation:

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

#### 2.10.2.11 int rtxDurationToMSecs (OSUTF8CHAR * *buf*, OSUINT32 *bufsize*, OSINT32 * *msecs*)

This function converts a duration string to milliseconds. In the case of a string prepended with a minus sign (-) the duration in milliseconds will have negative value.

**Parameters**

***buf*** Pointer to OSUTF8CHAR array.

***bufsize*** OSINT32 indicates the bufsize to be read.

***msecs*** OSINT32 updated after calculation.

**Returns**

Completion status of operation:

- 0(RT_OK) = success,
- negative return value is error (RTERR_NOTINIT/RTERR_INVFORMAT/RTERR_TOOBIG). Return value is taken from rtxErrCodes.h header file

#### 2.10.2.12 int rtxGDayToString (const OSNumDateTime * *pvalue*, OSUTF8CHAR * *buffer*, size_t *bufsize*)

This function formats a gregorian day value to a string (DD).

**Parameters**

***pvalue*** Pointer to OSNumDateTime structure containing day value to be formatted.

***buffer*** Buffer into which date is to be formatted. This is a fixed-sized buffer. The user must provide a buffer that is large enough to hold the formatted time string (in this case, at least 3 characters long).

***bufsize*** Size of the buffer to receive the formatted date.

**Returns**

Completion status of operation:

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

#### 2.10.2.13 int rtxGetCurrDateTime (OSNumDateTime * *pvalue*)

This function reads the system date and time and stores the value in the given OSNumDateTime structure variable.

**Parameters**

***pvalue*** Pointer to OSNumDateTime structure.

**Returns**

Completion status of operation:

- 0 in case success
- negative in case failure

#### 2.10.2.14 int rtxGetDateTime (const OSNumDateTime * *pvalue*, time_t * *timeMs*)

This function converts an OSNumDateTime structure to a calendar time encoded as a value of type time_t.

**Parameters**

***pvalue*** The pointed OSNumDateTime structure variable to be converted.

***timeMs*** A pointer to time_t value to be set.

**Returns**

Completion status of operation:

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

#### 2.10.2.15 int rtxGMonthDayToString (const OSNumDateTime * *pvalue*, OSUTF8CHAR * *buffer*, size_t *bufsize*)

This function formats a gregorian month and day value to a string (MM-DD).

**Parameters**

***pvalue*** Pointer to OSNumDateTime structure containing month and day value to be formatted.

***buffer*** Buffer into which date is to be formatted. This is a fixed-sized buffer. The user must provide a buffer that is large enough to hold the formatted time string (in this case, at least 6 characters long).

***bufsize*** Size of the buffer to receive the formatted date.

**Returns**

Completion status of operation:

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

#### 2.10.2.16 int rtxGMonthToString (const OSNumDateTime * *pvalue*, OSUTF8CHAR * *buffer*, size_t *bufsize*)

This function formats a gregorian month value to a string (MM).

**Parameters**

***pvalue*** Pointer to OSNumDateTime structure containing month value to be formatted.

***buffer*** Buffer into which date is to be formatted. This is a fixed-sized buffer. The user must provide a buffer that is large enough to hold the formatted time string (in this case, at least 3 characters long).

***bufsize*** Size of the buffer to receive the formatted date.

**Returns**

Completion status of operation:

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

#### 2.10.2.17 int rtxGYearMonthToString (const OSNumDateTime * *pvalue*, OSUTF8CHAR * *buffer*, size_t *bufsize*)

This function formats a gregorian year and month value to a string (CCYY-MM).

**Parameters**

***pvalue*** Pointer to OSNumDateTime structure containing year and month value to be formatted.

***buffer*** Buffer into which date is to be formatted. This is a fixed-sized buffer. The user must provide a buffer that is large enough to hold the formatted time string (in this case, at least 8 characters long).

***bufsize*** Size of the buffer to receive the formatted date.

**Returns**

Completion status of operation:

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

#### 2.10.2.18 int rtxGYearToString (const OSNumDateTime * *pvalue*, OSUTF8CHAR * *buffer*, size_t *bufsize*)

This function formats a gregorian year value to a string (CCYY).

**Parameters**

***pvalue*** Pointer to OSNumDateTime structure containing year value to be formatted.

***buffer*** Buffer into which date is to be formatted. This is a fixed-sized buffer. The user must provide a buffer that is large enough to hold the formatted time string (in this case, at least 5 characters long).

***bufsize*** Size of the buffer to receive the formatted date.

**Returns**

Completion status of operation:

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

#### 2.10.2.19 int rtxMSecsToDuration (OSINT32 *msecs*, OSUTF8CHAR * *buf*, OSUINT32 *bufsize*)

This function converts millisecs to a duration string with format "PnYnMnDTnHnMnS". In case of negative duration a minus sign is prepended to the output string

**Parameters**

***msecs*** Number of milliseconds.

***buf*** Output buffer to recieve formatted duration.

***bufsize*** Output buffer size.

**Returns**

Completion status of operation: 0 successful are same -1 unsuccessul

#### 2.10.2.20 int rtxParseDateString (const OSUTF8CHAR * *inpdata*, size_t *inpdatalen*, OSNumDateTime * *pvalue*)

This function decodes a date value from a supplied string and sets the given OSNumDateTime argument to the decoded date value.

**Parameters**

***inpdata*** Date string to be parsed/decoded as OSNumDateTime.

- The format of date is CCYY-MM-DD
- The value of CCYY is from 0000-9999
- The value of MM is 01 - 12
- The value of DD is 01 - XX (where XX is the Days in MM month in CCYY year)

***inpdatalen*** For decoding, consider inpdata string up to this length.

***pvalue*** The OSNumDateTime structure variable will be set to the decoded date value.

- Only year, month,day value will be set.
- The value of pvalue->year is in range 0 to 9999

- The value of pvalue->mon is in range 1 to 12
- The value of pvalue->day is in range 1 to XX

**Returns**

Completion status of operation:

- 0(RT_OK) = success,
- negative return value is error (RTERR_NOTINIT/RTERR_INVFORMAT/RTERR_BADVALUE). Return value is taken from rtxErrCodes.h header file

#### 2.10.2.21 int rtxParseDateTimeString (const OSUTF8CHAR * *inpdata*, size_t *inpdatalen*, OSNumDateTime * *pvalue*)

This function decodes a datetime value from a supplied string and sets the given OSNumDateTime to the decoded date and time value.

**Parameters**

***inpdata*** Input date/time string to be parsed.

***inpdatalen*** For decoding, consider the inpdata string up to this length.

***pvalue*** The pointed OSNumDateTime structure variable will be set to the decoded date and time value.

**Returns**

Completion status of operation:

- 0(RT_OK) = success,
- negative return value is error (RTERR_NOTINIT/RTERR_INVFORMAT/RTERR_BADVALUE). Return value is taken from rtxErrCodes.h header file

#### 2.10.2.22 int rtxParseGDayString (const OSUTF8CHAR * *inpdata*, size_t *inpdatalen*, OSNumDateTime * *pvalue*)

This function decodes a gregorian day value from a supplied string and sets the day field in the given OSNumDateTime to the decoded value.

**Parameters**

***inpdata*** Input string to be parsed.

***inpdatalen*** For decoding, consider the inpdata string up to this length.

***pvalue*** The day field in the given OSNumDateTime variable will be set to the decoded value.

**Returns**

Completion status of operation:

- 0(RT_OK) = success,
- negative return value is error (RTERR_NOTINIT/RTERR_INVFORMAT/RTERR_BADVALUE). Return value is taken from rtxErrCodes.h header file

#### 2.10.2.23 int rtxParseGMonthDayString (const OSUTF8CHAR * *inpdata*, size_t *inpdatalen*, OSNumDateTime * *pvalue*)

This function decodes a gregorian month and day value from a supplied string and sets the month and day fields in the given OSNumDateTime to the decoded values.

**Parameters**

***inpdata*** Input string to be parsed.

***inpdatalen*** For decoding, consider the inpdata string up to this length.

***pvalue*** The month and day fields in the given OSNumDateTime variable will be set to the decoded values.

**Returns**

Completion status of operation:

- 0(RT_OK) = success,
- negative return value is error (RTERR_NOTINIT/RTERR_INVFORMAT/RTERR_BADVALUE). Return value is taken from rtxErrCodes.h header file

#### 2.10.2.24 int rtxParseGMonthString (const OSUTF8CHAR * *inpdata*, size_t *inpdatalen*, OSNumDateTime * *pvalue*)

This function decodes a gregorian month value from a supplied string and sets the month field in the given OSNumDateTime to the decoded value.

**Parameters**

***inpdata*** Input string to be parsed.

***inpdatalen*** For decoding, consider the inpdata string up to this length.

***pvalue*** The month field in the given OSNumDateTime variable will be set to the decoded value.

**Returns**

Completion status of operation:

- 0(RT_OK) = success,
- negative return value is error (RTERR_NOTINIT/RTERR_INVFORMAT/RTERR_BADVALUE). Return value is taken from rtxErrCodes.h header file

#### 2.10.2.25 int rtxParseGYearMonthString (const OSUTF8CHAR * *inpdata*, size_t *inpdatalen*, OSNumDateTime * *pvalue*)

This function decodes a gregorian year and month value from a supplied string and sets the year and month fields in the given OSNumDateTime to the decoded values.

**Parameters**

***inpdata*** Input string to be parsed.

***inpdatalen*** For decoding, consider the inpdata string up to this length.

***pvalue*** The year and month fields in the given OSNumDateTime variable will be set to the decoded value.

**Returns**

Completion status of operation:

- 0(RT_OK) = success,
- negative return value is error (RTERR_NOTINIT/RTERR_INVFORMAT/RTERR_BADVALUE). Return value is taken from rtxErrCodes.h header file

#### 2.10.2.26 int rtxParseGYearString (const OSUTF8CHAR * *inpdata*, size_t *inpdatalen*, OSNumDateTime * *pvalue*)

This function decodes a gregorian year value from a supplied string and sets the year in the given OSNumDateTime to the decoded value.

**Parameters**

***inpdata*** Input string to be parsed.

***inpdatalen*** For decoding, consider the inpdata string up to this length.

***pvalue*** The year field in the given OSNumDateTime structure variable will be set to the decoded value.

**Returns**

Completion status of operation:

- 0(RT_OK) = success,
- negative return value is error (RTERR_NOTINIT/RTERR_INVFORMAT/RTERR_BADVALUE). Return value is taken from rtxErrCodes.h header file

#### 2.10.2.27 int rtxParseTimeString (const OSUTF8CHAR * *inpdata*, size_t *inpdatalen*, OSNumDateTime * *pvalue*)

This function decodes a time value from a supplied string and sets the given OSNumDateTime structure to the decoded time value.

**Parameters**

***inpdata*** The inpdata is a time string to be parsed/decoded as OSNumDateTime.

- The format of date is hh:mm:ss.ss (1) or hh:mm:ss.ssZ (2) or hh:mm:ss.ss+HH:MM (3) or hh:mm:ss.ss-HH:MM (4)
- The value of hh is from 00-23
- The value of mm is 00 - 59
- The value of ss.ss is 00.00 - 59.99
- The value of HH:MM is 00.00 - 24.00

***inpdatalen*** For decoding, consider the inpdata string up to this length.

***pvalue*** The OSNumDateTime structure variable will be set to the decoded time value.

- Only hour, min, sec value will be set.
- The value of pvalue->hour is in range 0 to 23
- The value of pvalue->mon is in range 0 to 59
- The value of pvalue->day is in range 0 to 59.99
- The value of pvalue->tz_flag is FALSE for format(1) otherwise TRUE

- The value of pvalue->tzo is 0 for format(2) otherwise Calculation of pvalue->tzo for format (3),(4) is HH∗60+MM
- The value of pvalue->tzo is -840 <= tzo <= 840 for format(3),(4) otherwise

**Returns**

Completion status of operation:

- 0(RT_OK) = success,
- negative return value is error (RTERR_NOTINIT/RTERR_INVFORMAT/RTERR_BADVALUE). Return value is taken from rtxErrCodes.h header file

#### 2.10.2.28 int rtxSetDateTime (OSNumDateTime ∗ *pvalue*, struct tm ∗ *timeStruct*)

This function converts a structure of type 'struct tm' to an OSNumDateTime structure.

**Parameters**

***pvalue*** The pointed OSNumDateTime structure variable will be set to time value.

***timeStruct*** A pointer to tm structure to be converted.

**Returns**

Completion status of operation:

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

#### 2.10.2.29 int rtxSetLocalDateTime (OSNumDateTime ∗ *pvalue*, time_t *timeMs*)

This function converts a local date and time value to an OSNumDateTime structure.

**Parameters**

***pvalue*** The pointed OSNumDateTime structure variable will be set to time value.

***timeMs*** A calendar time encoded as a value of type time_t.

**Returns**

Completion status of operation:

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

#### 2.10.2.30 int rtxSetUtcDateTime (OSNumDateTime ∗ *pvalue*, time_t *timeMs*)

This function converts a UTC date and time value to an OSNumDateTime structure.

**Parameters**

***pvalue*** The pointed OSNumDateTime structure variable will be set to time value.

***timeMs*** A calendar time encoded as a value of type time_t. The time is represented as seconds elapsed since midnight (00:00:00), January 1, 1970, coordinated universal time (UTC).

**Returns**

Completion status of operation:

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

#### 2.10.2.31 OSBOOL rtxTimeIsValid (const OSNumDateTime * *pvalue*)

This function verifies that time members (hour, minute, second, timezone) of the OSNumDateTime structure contains valid values.

**Parameters**

***pvalue*** Pointer to OSNumDateTime structure to be checked.

**Returns**

Boolean result: true means data is valid.

#### 2.10.2.32 int rtxTimeToString (const OSNumDateTime * *pvalue*, OSUTF8CHAR * *buffer*, size_t *bufsize*)

This function formats a numeric time value consisting of individual time components (hour, minute, second, fraction-of-second, time zone) into XML schema standard format (HH:MM:SS[.frac][TZ]).

**Parameters**

***pvalue*** Pointer to OSNumDateTime structure containing time components to be formatted.

***buffer*** Buffer into which date is to be formatted. This is a fixed-sized buffer. The user must provide a buffer that is large enough to hold the formatted time string.

***bufsize*** Size of the buffer to receive the formatted date.

**Returns**

Completion status of operation:

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

## 2.11 Floating-point number utility functions

### Functions

- OSREAL rtxGetMinusInfinity (void)
- OSREAL rtxGetMinusZero (void)
- OSREAL rtxGetNaN (void)
- OSREAL rtxGetPlusInfinity (void)
- OSBOOL rtxIsMinusInfinity (OSREAL value)
- OSBOOL rtxIsMinusZero (OSREAL value)
- OSBOOL rtxIsNaN (OSREAL value)
- OSBOOL rtxIsPlusInfinity (OSREAL value)
- OSBOOL rtxIsApproximate (OSREAL a, OSREAL b, OSREAL delta)
- OSBOOL rtxIsApproximateAbs (OSREAL a, OSREAL b, OSREAL delta)

### 2.11.1 Detailed Description

Floating-point utility function provide run-time functions for handling floating-point number types defined within a schema.

### 2.11.2 Function Documentation

#### 2.11.2.1 OSREAL rtxGetMinusInfinity (void)

Returns the IEEE negative infinity value. This is defined as 0xfff0000000000000 in IEEE standard 754. We assume the presence of the IEEE double type, that is, 64-bits of precision.

#### 2.11.2.2 OSREAL rtxGetMinusZero (void)

Returns the IEEE minus zero value. This is defined as 0x8000000000000000 in IEEE standard 754. We assume the presence of the IEEE double type, that is, 64-bits of precision.

#### 2.11.2.3 OSREAL rtxGetNaN (void)

Returns the IEEE Not-A-Number (NaN) value. This is defined as 0x7ff8000000000000 in IEEE standard 754. We assume the presence of the IEEE double type, that is, 64-bits of precision.

#### 2.11.2.4 OSREAL rtxGetPlusInfinity (void)

Returns the IEEE posative infinity value. This is defined as 0x7ff0000000000000 in IEEE standard 754. We assume the presence of the IEEE double type, that is, 64-bits of precision.

#### 2.11.2.5 OSBOOL rtxIsApproximate (OSREAL *a*, OSREAL *b*, OSREAL *delta*)

A utility function that return TRUE when first number are approximate to second number with given precision.

**Parameters**

***a*** The input real value.

***b*** The input real value.

***delta*** difference must be low than delta ∗ a 1E-7 - set best precision for float; 1E-15 - set best precision for double.

#### 2.11.2.6 OSBOOL rtxIsApproximateAbs (OSREAL *a*, OSREAL *b*, OSREAL *delta*)

A utility function that return TRUE when first number are approximate to second number with given absolute precision.

**Parameters**

***a*** The input real value.

***b*** The input real value.

***delta*** difference must be low than delta

#### 2.11.2.7 OSBOOL rtxIsMinusInfinity (OSREAL *value*)

A utility function that compares the given input value to the IEEE 754 value for negative infinity.

**Parameters**

***value*** The input real value.

#### 2.11.2.8 OSBOOL rtxIsMinusZero (OSREAL *value*)

A utility function that compares the given input value to the IEEE 754 value for minus zero.

**Parameters**

***value*** The input real value.

#### 2.11.2.9 OSBOOL rtxIsNaN (OSREAL *value*)

A utility function that compares the given input value to the IEEE 754 value for Not-A-Number (NaN).

**Parameters**

***value*** The input real value.

#### 2.11.2.10 OSBOOL rtxIsPlusInfinity (OSREAL *value*)

A utility function that compares the given input value to the IEEE 754 value for positive infinity.

**Parameters**

***value*** The input real value.

## 2.12 Decimal number utility functions

### Functions

- const char * **rtxNR3toDecimal** (OSCTXT *pctxt, const char *object_p)

### 2.12.1 Detailed Description

Decimal utility function provide run-time functions for handling decimal number types defined within a schema.

## 2.13 UTF-8 String Functions

### Defines

- #define **rtxUTF8StrToInt32** rtxUTF8StrToInt
- #define **rtxUTF8StrToUInt32** rtxUTF8StrToUInt
- #define RTUTF8STRCMPL(name, lstr) rtxUTF8Strcmp(name,(const OSUTF8CHAR*)lstr)
- #define **OSRTCHKUTF8LEN**(str, lower, upper, stat)

### Functions

- long rtxUTF8ToUnicode (OSCTXT *pctxt, const OSUTF8CHAR *inbuf, OSUNICHAR *outbuf, size_t outbufsiz)
- int rtxValidateUTF8 (OSCTXT *pctxt, const OSUTF8CHAR *inbuf)
- size_t rtxUTF8Len (const OSUTF8CHAR *inbuf)
- size_t **rtxCalcUTF8Len** (const OSUTF8CHAR *inbuf, size_t inbufBytes)
- size_t rtxUTF8LenBytes (const OSUTF8CHAR *inbuf)
- int rtxUTF8CharSize (OS32BITCHAR wc)
- int rtxUTF8EncodeChar (OS32BITCHAR wc, OSOCTET *buf, size_t bufsiz)
- int rtxUTF8DecodeChar (OSCTXT *pctxt, const OSUTF8CHAR *pinbuf, int *pInsize)
- OS32BITCHAR rtxUTF8CharToWC (const OSUTF8CHAR *buf, OSUINT32 *len)
- OSUTF8CHAR * rtxUTF8StrChr (OSUTF8CHAR *utf8str, OS32BITCHAR utf8char)
- OSUTF8CHAR * rtxUTF8Strdup (OSCTXT *pctxt, const OSUTF8CHAR *utf8str)
- OSUTF8CHAR * rtxUTF8Strndup (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, size_t nbytes)
- OSUTF8CHAR * rtxUTF8StrRefOrDup (OSCTXT *pctxt, const OSUTF8CHAR *utf8str)
- OSBOOL rtxUTF8StrEqual (const OSUTF8CHAR *utf8str1, const OSUTF8CHAR *utf8str2)
- OSBOOL rtxUTF8StrnEqual (const OSUTF8CHAR *utf8str1, const OSUTF8CHAR *utf8str2, size_t count)
- int rtxUTF8Strcmp (const OSUTF8CHAR *utf8str1, const OSUTF8CHAR *utf8str2)
- int rtxUTF8Strncmp (const OSUTF8CHAR *utf8str1, const OSUTF8CHAR *utf8str2, size_t count)
- OSUTF8CHAR * rtxUTF8Strcpy (OSUTF8CHAR *dest, size_t bufsiz, const OSUTF8CHAR *src)
- OSUTF8CHAR * rtxUTF8Strncpy (OSUTF8CHAR *dest, size_t bufsiz, const OSUTF8CHAR *src, size_t nchars)
- OSUINT32 rtxUTF8StrHash (const OSUTF8CHAR *str)
- const OSUTF8CHAR * rtxUTF8StrJoin (OSCTXT *pctxt, const OSUTF8CHAR *str1, const OSUTF8CHAR *str2, const OSUTF8CHAR *str3, const OSUTF8CHAR *str4, const OSUTF8CHAR *str5)
- int rtxUTF8StrToBool (const OSUTF8CHAR *utf8str, OSBOOL *pvalue)
- int rtxUTF8StrnToBool (const OSUTF8CHAR *utf8str, size_t nbytes, OSBOOL *pvalue)
- int rtxUTF8StrToDouble (const OSUTF8CHAR *utf8str, OSREAL *pvalue)
- int rtxUTF8StrnToDouble (const OSUTF8CHAR *utf8str, size_t nbytes, OSREAL *pvalue)
- int rtxUTF8StrToInt (const OSUTF8CHAR *utf8str, OSINT32 *pvalue)
- int rtxUTF8StrnToInt (const OSUTF8CHAR *utf8str, size_t nbytes, OSINT32 *pvalue)
- int rtxUTF8StrToUInt (const OSUTF8CHAR *utf8str, OSUINT32 *pvalue)
- int rtxUTF8StrnToUInt (const OSUTF8CHAR *utf8str, size_t nbytes, OSUINT32 *pvalue)
- int rtxUTF8StrToSize (const OSUTF8CHAR *utf8str, size_t *pvalue)
- int rtxUTF8StrnToSize (const OSUTF8CHAR *utf8str, size_t nbytes, size_t *pvalue)
- int rtxUTF8StrToInt64 (const OSUTF8CHAR *utf8str, OSINT64 *pvalue)
- int rtxUTF8StrnToInt64 (const OSUTF8CHAR *utf8str, size_t nbytes, OSINT64 *pvalue)
- int rtxUTF8StrToUInt64 (const OSUTF8CHAR *utf8str, OSUINT64 *pvalue)
- int rtxUTF8StrnToUInt64 (const OSUTF8CHAR *utf8str, size_t nbytes, OSUINT64 *pvalue)

- int rtxUTF8ToDynUniStr (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, const OSUNICHAR **ppdata, OSUINT32 *pnchars)
- int rtxUTF8RemoveWhiteSpace (const OSUTF8CHAR *utf8instr, size_t nbytes, const OSUTF8CHAR **putf8outstr)
- int rtxUTF8StrToDynHexStr (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, OSDynOctStr *pvalue)
- int rtxUTF8StrnToDynHexStr (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, size_t nbytes, OSDynOctStr *pvalue)
- int rtxUTF8StrToNamedBits (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, const OSBitMapItem *pBitMap, OSOCTET *pvalue, OSUINT32 *pnbits, OSUINT32 bufsize)
- const OSUTF8CHAR * rtxUTF8StrNextTok (OSUTF8CHAR *utf8str, OSUTF8CHAR **ppNext)

### 2.13.1 Detailed Description

The UTF-8 string functions handle string operations on UTF-8 encoded strings. This is the default character string data type used for encoded XML data. UTF-8 strings are represented in C as strings of unsigned characters (bytes) to cover the full range of possible single character encodings.

### 2.13.2 Define Documentation

#### 2.13.2.1 #define OSRTCHKUTF8LEN(str, lower, upper, stat)

**Value:**

```
do { size_t nchars = rtxUTF8Len (str); \
stat = (nchars >= lower && nchars <= upper) ? 0 : RTERR_CONSVIO; } while(0)
```

#### 2.13.2.2 #define RTUTF8STRCMPL(name, lstr) rtxUTF8Strcmp(name,(const OSUTF8CHAR*)lstr)

Compare UTF-8 string to a string literal.

**Parameters**

***name*** UTF-8 string variable.

***lstr*** C string literal value (quoted contant such as "a")

### 2.13.3 Function Documentation

#### 2.13.3.1 int rtxUTF8CharSize (OS32BITCHAR *wc*)

This function will return the number of bytes needed to encode the given 32-bit universal character value as a UTF-8 character.

**Parameters**

***wc*** 32-bit wide character value.

**Returns**

Number of bytes needed to encode as UTF-8.

#### 2.13.3.2 OS32BITCHAR rtxUTF8CharToWC (const OSUTF8CHAR * *buf*, OSUINT32 * *len*)

Thia function will convert a UTF-8 encoded character value into a wide character.

**Parameters**

***buf*** Pointer to UTF-8 character value.

***len*** Pointer to integer to receive decoded size (in bytes) of the UTF-8 character value sequence.

**Returns**

Converted wide character value.

#### 2.13.3.3 int rtxUTF8DecodeChar (OSCTXT * *pctxt*, const OSUTF8CHAR * *pinbuf*, int * *pInsize*)

This function will convert an encoded UTF-8 character byte string into a wide character value.

**Parameters**

***pctxt*** A pointer to a context structure.

***pinbuf*** Pointer to UTF-8 byte sequence to be decoded.

***pInsize*** Number of bytes that were consumed (i.e. size of the character).

**Returns**

32-bit wide character value.

#### 2.13.3.4 int rtxUTF8EncodeChar (OS32BITCHAR *wc*, OSOCTET * *buf*, size_t *bufsiz*)

This function will convert a wide character into an encoded UTF-8 character byte string.

**Parameters**

***wc*** 32-bit wide character value.

***buf*** Buffer to receive encoded UTF-8 character value.

***bufsiz*** Size of the buffer ot receive the encoded value.

**Returns**

Number of bytes consumed to encode character or negative status code if error.

#### 2.13.3.5 size_t rtxUTF8Len (const OSUTF8CHAR * *inbuf*)

This function will return the length (in characters) of a null-terminated UTF-8 encoded string.

**Parameters**

***inbuf*** A pointer to the null-terminated UTF-8 encoded string.

**Returns**

Number of characters in string. Note that this may be different than the number of bytes as UTF-8 characters can span multiple-bytes.

#### 2.13.3.6 size_t rtxUTF8LenBytes (const OSUTF8CHAR * *inbuf*)

This function will return the length (in bytes) of a null-terminated UTF-8 encoded string.

**Parameters**

***inbuf*** A pointer to the null-terminated UTF-8 encoded string.

**Returns**

Number of bytes in the string.

#### 2.13.3.7 int rtxUTF8RemoveWhiteSpace (const OSUTF8CHAR * *utf8instr*, size_t *nbytes*, const OSUTF8CHAR ** *putf8outstr*)

This function removes leading and trailing whitespace from a string.

**Parameters**

***utf8instr*** Input UTF-8 string from which to removed whitespace.

***nbytes*** Size in bytes of utf8instr.

***putf8outstr*** Pointer to receive result string.

**Returns**

Positive value = length of result string, negative value = error code.

#### 2.13.3.8 OSUTF8CHAR* rtxUTF8StrChr (OSUTF8CHAR * *utf8str*, OS32BITCHAR *utf8char*)

This function finds a character in the given UTF-8 character string. It is similar to the C `strchr` function.

**Parameters**

***utf8str*** Null-terminated UTF-8 string to be searched.

***utf8char*** 32-bit Unicode character to find.

**Returns**

Pointer to to the first occurrence of character in string, or NULL if character is not found.

#### 2.13.3.9 int rtxUTF8Strcmp (const OSUTF8CHAR * *utf8str1*, const OSUTF8CHAR * *utf8str2*)

This function compares two UTF-8 character strings and returns a trinary result (equal, less than, greater than). It is similar to the C `strcmp` function.

**Parameters**

***utf8str1*** UTF-8 string to be compared.

***utf8str2*** UTF-8 string to be compared.

**Returns**

-1 if utf8str1 is less than utf8str2, 0 if the two string are equal, and +1 if the utf8str1 is greater than utf8str2.

#### 2.13.3.10 OSUTF8CHAR* rtxUTF8Strcpy (OSUTF8CHAR * *dest*, size_t *bufsiz*, const OSUTF8CHAR * *src*)

This function copies a null-terminated UTF-8 string to a target buffer. It is similar to the C `strcpy` function except more secure because it checks for buffer overrun.

**Parameters**

***dest*** Pointer to destination buffer to receive string.

***bufsiz*** Size of the destination buffer.

***src*** Pointer to null-terminated string to copy.

**Returns**

Pointer to destination buffer or NULL if copy failed.

#### 2.13.3.11 OSUTF8CHAR* rtxUTF8Strdup (OSCTXT * *pctxt*, const OSUTF8CHAR * *utf8str*)

This function creates a duplicate copy of the given UTF-8 character string. It is similar to the C `strdup` function. Memory for the duplicated string is allocated using the `rtxMemAlloc` function.

**Parameters**

***pctxt*** A pointer to a context structure.

***utf8str*** Null-terminated UTF-8 string to be duplicated.

**Returns**

Pointer to duplicated string value.

#### 2.13.3.12 OSBOOL rtxUTF8StrEqual (const OSUTF8CHAR * *utf8str1*, const OSUTF8CHAR * *utf8str2*)

This function compares two UTF-8 string values for equality.

**Parameters**

***utf8str1*** UTF-8 string to be compared.

***utf8str2*** UTF-8 string to be compared.

**Returns**

TRUE if equal, FALSE if not.

#### 2.13.3.13 OSUINT32 rtxUTF8StrHash (const OSUTF8CHAR * *str*)

This function computes a hash code for the given string value.

**Parameters**

***str*** Pointer to string.

**Returns**

Hash code value.

#### 2.13.3.14 const OSUTF8CHAR* rtxUTF8StrJoin (OSCTXT * *pctxt*, const OSUTF8CHAR * *str1*, const OSUTF8CHAR * *str2*, const OSUTF8CHAR * *str3*, const OSUTF8CHAR * *str4*, const OSUTF8CHAR * *str5*)

This function concatanates up to five substrings together into a single string.

**Parameters**

***pctxt*** Pointer to a context block structure.

***str1*** Pointer to substring to join.

***str2*** Pointer to substring to join.

***str3*** Pointer to substring to join.

***str4*** Pointer to substring to join.

***str5*** Pointer to substring to join.

**Returns**

Composite string consisting of all parts. Memory is allocated for this string using rtxMemAlloc and must be freed using either rtxMemFreePtr or rtxMemFree. If memory allocation for the string fails, NULL is returned.

#### 2.13.3.15 int rtxUTF8Strncmp (const OSUTF8CHAR * *utf8str1*, const OSUTF8CHAR * *utf8str2*, size_t *count*)

This function compares two UTF-8 character strings and returns a trinary result (equal, less than, greater than). In this case, a maximum count of the number of bytes to compare can be specified. It is similar to the C `strncmp` function.

**Parameters**

***utf8str1*** UTF-8 string to be compared.

***utf8str2*** UTF-8 string to be compared.

***count*** Number of bytes to compare.

**Returns**

-1 if utf8str1 is less than utf8str2, 0 if the two string are equal, and +1 if the utf8str1 is greater than utf8str2.

#### 2.13.3.16 OSUTF8CHAR* rtxUTF8Strncpy (OSUTF8CHAR * *dest*, size_t *bufsiz*, const OSUTF8CHAR * *src*, size_t *nchars*)

This function copies the given number of characters from a UTF-8 string to a target buffer. It is similar to the C `strncpy` function except more secure because it checks for buffer overrun and ensures a null-terminator is copied to the end of the target buffer

**Parameters**

***dest*** Pointer to destination buffer to receive string.

***bufsiz*** Size of the destination buffer.

***src*** Pointer to null-terminated string to copy.

***nchars*** Number of characters to copy.

**Returns**

Pointer to destination buffer or NULL if copy failed.

#### 2.13.3.17 OSUTF8CHAR* rtxUTF8Strndup (OSCTXT * *pctxt*, const OSUTF8CHAR * *utf8str*, size_t *nbytes*)

This function creates a duplicate copy of the given UTF-8 character string. It is similar to the `rtxUTF8Strdup` function except that it allows the number of bytes to convert to be specified. Memory for the duplicated string is allocated using the `rtxMemAlloc` function.

**Parameters**

***pctxt*** A pointer to a context structure.

***utf8str*** UTF-8 string to be duplicated.

***nbytes*** Number of bytes from `utf8str` to duplicate.

**Returns**

Pointer to duplicated string value.

#### 2.13.3.18 OSBOOL rtxUTF8StrnEqual (const OSUTF8CHAR * *utf8str1*, const OSUTF8CHAR * *utf8str2*, size_t *count*)

This function compares two UTF-8 string values for equality. It is similar to the `rtxUTF8StrEqual` function except that it allows the number of bytes to compare to be specified.

**Parameters**

***utf8str1*** UTF-8 string to be compared.

***utf8str2*** UTF-8 string to be compared.

***count*** Number of bytes to compare.

**Returns**

TRUE if equal, FALSE if not.

#### 2.13.3.19 const OSUTF8CHAR* rtxUTF8StrNextTok (OSUTF8CHAR * *utf8str*, OSUTF8CHAR ** *ppNext*)

This function returns the next whitespace-separated token from the input string. It also returns a pointer to the first non-whitespace chracter after the parsed token. Note that the input string is altered in the operation as null-terminators are insterted to mark the token boundaries.

**Parameters**

***utf8str*** Null-terminated UTF-8 string to parse. This string will be altered. Use rtxUTF8Strdup to make a copy of original string before calling this function if the original string cannot be altered.

***ppNext*** Pointer to receive next location in string after parsed token. This can be used as input to get the next token. If NULL returned, all tokens in in the string have been parsed.

**Returns**

Pointer to next parsed token. NULL if no more tokens.

#### 2.13.3.20 int rtxUTF8StrnToBool (const OSUTF8CHAR * *utf8str*, size_t *nbytes*, OSBOOL * *pvalue*)

This function converts the given part of UTF-8 string to a boolean (true/false) value. It is assumed the string contains only the tokens 'true', 'false', '1', or '0'.

**Parameters**

***utf8str*** Null-terminated UTF-8 string to convert

***nbytes*** Size in bytes of utf8Str.

***pvalue*** Pointer to boolean value to receive result

**Returns**

Status: 0 = OK, negative value = error

#### 2.13.3.21 int rtxUTF8StrnToDouble (const OSUTF8CHAR * *utf8str*, size_t *nbytes*, OSREAL * *pvalue*)

This function converts the given part of UTF-8 string to a double value. It is assumed the string contains only numeric digits, whitespace, and other special floating point characters. It is similar to the C atof function except that the result is returned as a separate argument and an error status value is returned if the conversion cannot be performed successfully.

**Parameters**

***utf8str*** UTF-8 string to convert. Not necessary to be null-terminated.

***nbytes*** Size in bytes of utf8Str.

***pvalue*** Pointer to double to receive result

**Returns**

Status: 0 = OK, negative value = error

#### 2.13.3.22 int rtxUTF8StrnToDynHexStr (OSCTXT * *pctxt*, const OSUTF8CHAR * *utf8str*, size_t *nbytes*, OSDynOctStr * *pvalue*)

This function converts the given part of UTF-8 string to a octet string value. The string consists of a series of hex digits. This is the dynamic version in which memory is allocated for the returned octet string variable.

**Parameters**

***pctxt*** Pointer to context block structure.

***utf8str*** Null-terminated UTF-8 string to convert

***nbytes*** Size in bytes of utf8Str.

***pvalue*** Pointer to a variable to receive the decoded octet string value.

**Returns**

Completion status of operation:

- 0 = success,
- negative return value is error.

#### 2.13.3.23 int rtxUTF8StrnToInt (const OSUTF8CHAR * *utf8str*, size_t *nbytes*, OSINT32 * *pvalue*)

This function converts the given part of UTF-8 string to an integer value. It is assumed the string contains only numeric digits and whitespace. It is similar to the C atoi function except that the result is returned as a separate argument and an error status value is returned if the conversion cannot be performed successfully.

**Parameters**

***utf8str*** UTF-8 string to convert. Not necessary to be null-terminated.

***nbytes*** Size in bytes of utf8Str.

***pvalue*** Pointer to integer to receive result

**Returns**

Status: 0 = OK, negative value = error

#### 2.13.3.24 int rtxUTF8StrnToInt64 (const OSUTF8CHAR * *utf8str*, size_t *nbytes*, OSINT64 * *pvalue*)

This function converts the given part of UTF-8 string to a 64-bit integer value. It is assumed the string contains only numeric digits and whitespace.

**Parameters**

***utf8str*** UTF-8 string to convert. Not necessary to be null-terminated.

***nbytes*** Size in bytes of utf8Str.

***pvalue*** Pointer to integer to receive result

**Returns**

Status: 0 = OK, negative value = error

#### 2.13.3.25 int rtxUTF8StrnToSize (const OSUTF8CHAR * *utf8str*, size_t *nbytes*, size_t * *pvalue*)

This function converts the given part of UTF-8 string to a size value (type size_t). It is assumed the string contains only numeric digits and whitespace.

**Parameters**

***utf8str*** UTF-8 string to convert. Not necessary to be null-terminated.

***nbytes*** Size in bytes of utf8Str.

***pvalue*** Pointer to size_t value to receive result

**Returns**

Status: 0 = OK, negative value = error

#### 2.13.3.26 int rtxUTF8StrnToUInt (const OSUTF8CHAR * *utf8str*, size_t *nbytes*, OSUINT32 * *pvalue*)

This function converts the given part of UTF-8 string to an unsigned integer value. It is assumed the string contains only numeric digits and whitespace.

**Parameters**

***utf8str*** UTF-8 string to convert. Not necessary to be null-terminated.

***nbytes*** Size in bytes of utf8Str.

***pvalue*** Pointer to integer to receive result

**Returns**

Status: 0 = OK, negative value = error

#### 2.13.3.27 int rtxUTF8StrnToUInt64 (const OSUTF8CHAR * *utf8str*, size_t *nbytes*, OSUINT64 * *pvalue*)

This function converts the given part of UTF-8 string to an unsigned 64-bit integer value. It is assumed the string contains only numeric digits and whitespace.

**Parameters**

***utf8str*** UTF-8 string to convert. Not necessary to be null-terminated.

***nbytes*** Size in bytes of utf8Str.

***pvalue*** Pointer to integer to receive result

**Returns**

Status: 0 = OK, negative value = error

#### 2.13.3.28 OSUTF8CHAR* rtxUTF8StrRefOrDup (OSCTXT * *pctxt*, const OSUTF8CHAR * *utf8str*)

This function check to see if the given UTF8 string pointer exists on the memory heap. If it does, its reference count is incremented; otherwise, a duplicate copy is made.

**Parameters**

***pctxt*** A pointer to a context structure.

***utf8str*** Null-terminated UTF-8 string variable.

**Returns**

Pointer to string value. This will either be the existing UTF-8 string pointer value (utf8str) or a new value.

#### 2.13.3.29 int rtxUTF8StrToBool (const OSUTF8CHAR * *utf8str*, OSBOOL * *pvalue*)

This function converts the given null-terminated UTF-8 string to a boolean (true/false) value. It is assumed the string contains only the tokens 'true', 'false', '1', or '0'.

**Parameters**

***utf8str*** Null-terminated UTF-8 string to convert

***pvalue*** Pointer to boolean value to receive result

**Returns**

Status: 0 = OK, negative value = error

#### 2.13.3.30 int rtxUTF8StrToDouble (const OSUTF8CHAR * *utf8str*, OSREAL * *pvalue*)

This function converts the given null-terminated UTF-8 string to a floating point (C/C++ double) value. It is assumed the string contains only numeric digits, special floating point characters (+,-,E,.), and whitespace. It is similar to the C atof function except that the result is returned as a separate argument and an error status value is returned if the conversion cannot be performed successfully.

**Parameters**

***utf8str*** Null-terminated UTF-8 string to convert

***pvalue*** Pointer to double to receive result

**Returns**

Status: 0 = OK, negative value = error

#### 2.13.3.31 int rtxUTF8StrToDynHexStr (OSCTXT * *pctxt*, const OSUTF8CHAR * *utf8str*, OSDynOctStr * *pvalue*)

This function converts the given null-terminated UTF-8 string to a octet string value. The string consists of a series of hex digits. This is the dynamic version in which memory is allocated for the returned octet string variable.

**Parameters**

***pctxt*** Pointer to context block structure.

***utf8str*** Null-terminated UTF-8 string to convert

***pvalue*** Pointer to a variable to receive the decoded octet string value.

**Returns**

Completion status of operation:

- 0 = success,
- negative return value is error.

#### 2.13.3.32 int rtxUTF8StrToInt (const OSUTF8CHAR * *utf8str*, OSINT32 * *pvalue*)

This function converts the given null-terminated UTF-8 string to an integer value. It is assumed the string contains only numeric digits and whitespace. It is similar to the C atoi function except that the result is returned as a separate argument and an error status value is returned if the conversion cannot be performed successfully.

**Parameters**

***utf8str*** Null-terminated UTF-8 string to convert

***pvalue*** Pointer to integer to receive result

**Returns**

Status: 0 = OK, negative value = error

#### 2.13.3.33 int rtxUTF8StrToInt64 (const OSUTF8CHAR * *utf8str*, OSINT64 * *pvalue*)

This function converts the given null-terminated UTF-8 string to a 64-bit integer value. It is assumed the string contains only numeric digits and whitespace.

**Parameters**

***utf8str*** Null-terminated UTF-8 string to convert

***pvalue*** Pointer to integer to receive result

**Returns**

Status: 0 = OK, negative value = error

#### 2.13.3.34 int rtxUTF8StrToNamedBits (OSCTXT * *pctxt*, const OSUTF8CHAR * *utf8str*, const OSBitMapItem * *pBitMap*, OSOCTET * *pvalue*, OSUINT32 * *pnbits*, OSUINT32 *bufsize*)

This function converts the given null-terminated UTF-8 string to named bit items. The token-to-bit mappings are defined by a bit map table that is passed into the function. It is assumed the string contains a space-separated list of named bit token values.

**Parameters**

***pctxt*** Context structure

***utf8str*** Null-terminated UTF-8 string to convert

***pBitMap*** Bit map defining bit to otken mappings

***pvalue*** Pointer to byte array to receive result.

***pnbits*** Pointer to integer to received number of bits.

***bufsize*** Size of byte array to received decoded bits.

**Returns**

Status: 0 = OK, negative value = error

#### 2.13.3.35 int rtxUTF8StrToSize (const OSUTF8CHAR * *utf8str*, size_t * *pvalue*)

This function converts the given null-terminated UTF-8 string to a size value (type size_t). It is assumed the string contains only numeric digits and whitespace.

**Parameters**

***utf8str*** Null-terminated UTF-8 string to convert

***pvalue*** Pointer to size_t value to receive result

**Returns**

Status: 0 = OK, negative value = error

#### 2.13.3.36 int rtxUTF8StrToUInt (const OSUTF8CHAR * *utf8str*, OSUINT32 * *pvalue*)

This function converts the given null-terminated UTF-8 string to an unsigned integer value. It is assumed the string contains only numeric digits and whitespace.

**Parameters**

***utf8str*** Null-terminated UTF-8 string to convert

***pvalue*** Pointer to integer to receive result

**Returns**

Status: 0 = OK, negative value = error

#### 2.13.3.37 int rtxUTF8StrToUInt64 (const OSUTF8CHAR * *utf8str*, OSUINT64 * *pvalue*)

This function converts the given null-terminated UTF-8 string to an unsigned 64-bit integer value. It is assumed the string contains only numeric digits and whitespace.

**Parameters**

***utf8str*** Null-terminated UTF-8 string to convert

***pvalue*** Pointer to integer to receive result

**Returns**

Status: 0 = OK, negative value = error

#### 2.13.3.38 int rtxUTF8ToDynUniStr (OSCTXT * *pctxt*, const OSUTF8CHAR * *utf8str*, const OSUNICHAR ** *ppdata*, OSUINT32 * *pnchars*)

This function converts the given UTF-8 string to a Unicode string. Memory is allocated for the Unicode string using the rtxMemAlloc function. This memory will be freed when the context is freed (rtxFreeContext) or it can be freed using rtxMemFreePtr.

**Parameters**

***pctxt*** A pointer to a context structure.

***utf8str*** UTF-8 string to convert, null-terminated.

***ppdata*** Pointer to pointer to receive output string.

***pnchars*** Pointer to integer to receive number of chars decoded.

**Returns**

Status: 0 = OK, negative value = error

#### 2.13.3.39 long rtxUTF8ToUnicode (OSCTXT * *pctxt*, const OSUTF8CHAR * *inbuf*, OSUNICHAR * *outbuf*, size_t *outbufsiz*)

This function converts a UTF-8 string to a Unicode string (UTF-16). The Unicode string is stored as an array of 16-bit characters (unsigned short integers).

**Parameters**

***pctxt*** A pointer to a context structure.

***inbuf*** UTF-8 string to convert.

***outbuf*** Output buffer to receive converted Unicode data.

***outbufsiz*** Size of the output buffer in bytes.

**Returns**

Completion status of operation:

- number of Unicode characters in the string
- negative return value is error.

#### 2.13.3.40 int rtxValidateUTF8 (OSCTXT * *pctxt*, const OSUTF8CHAR * *inbuf*)

This function will validate a UTF-8 encoded string to ensure that it is encoded correctly.

**Parameters**

***pctxt*** A pointer to a context structure.

***inbuf*** A pointer to the null-terminated UTF-8 encoded string.

**Returns**

Completion status of operation:

- 0 = success,
- negative return value is error.

## 2.14 Bit String Functions

### Defines

- #define OSRTBYTEARRAYSIZE(numbits) ((numbits+7)/8)

### Functions

- OSUINT32 rtxGetBitCount (OSUINT32 value)
- int rtxSetBit (OSOCTET *pBits, OSSIZE numbits, OSSIZE bitIndex)
- OSUINT32 rtxSetBitFlags (OSUINT32 flags, OSUINT32 mask, OSBOOL action)
- int rtxClearBit (OSOCTET *pBits, OSSIZE numbits, OSSIZE bitIndex)
- OSBOOL rtxTestBit (const OSOCTET *pBits, OSSIZE numbits, OSSIZE bitIndex)
- OSSIZE rtxLastBitSet (const OSOCTET *pBits, OSSIZE numbits)
- int rtxCheckBitBounds (OSCTXT *pctxt, OSOCTET **ppBits, OSSIZE *pNumocts, OSSIZE minRequiredBits, OSSIZE preferredLimitBits)

### 2.14.1 Detailed Description

Bit string functions allow bits to be set, cleared, or tested in arbitrarily sized byte arrays.

### 2.14.2 Define Documentation

#### 2.14.2.1 #define OSRTBYTEARRAYSIZE(numbits) ((numbits+7)/8)

This macro is used to calculate the byte array size required to hold the given number of bits.

### 2.14.3 Function Documentation

#### 2.14.3.1 int rtxCheckBitBounds (OSCTXT * *pctxt*, OSOCTET ** *ppBits*, OSSIZE * *pNumocts*, OSSIZE *minRequiredBits*, OSSIZE *preferredLimitBits*)

Check whether the given bit string is large enough, and expand it if necessary.

**Parameters**

***pctxt*** The context to use should memory need to be allocated.

***ppBits*** *ppBits is a pointer to the bit string, or NULL if one has not been created. If the string is expanded, *ppBits receives a pointer to the new bit string.

***pNumocts*** pNumocts points to the current size of the bit string in octets. If the bit string is expanded, *pNumocts receives the new size.

***minRequiredBits*** The minimum number of bits needed in the bit string. On return, pBits will point to a bit string with at least this many bits.

***preferredLimitBits*** The number of bits over which we prefer not to go. If nonzero, no more bytes will be allocated than necessary for this many bits, unless explicitly required by minRequiredBits.

**Returns**

If successful, 0. Otherwise, an error code.

#### 2.14.3.2 int rtxClearBit (OSOCTET * *pBits*, OSSIZE *numbits*, OSSIZE *bitIndex*)

This function clears the specified zero-counted bit in the bit string.

**Parameters**

***pBits*** Pointer to octets of bit string.

***numbits*** Number of bits in the bit string.

***bitIndex*** Index of bit to be cleared. The bit with index 0 is a most significant bit in the octet with index 0.

**Returns**

If successful, returns the previous state of bit. If bit was set the return value is positive, if bit was not set the return value is zero. Otherwise, return value is an error code:

- RTERR_OUTOFBND = bitIndex is out of bounds

#### 2.14.3.3 OSUINT32 rtxGetBitCount (OSUINT32 *value*)

This function returns the minimum size of the bit field required to hold the given integer value.

**Parameters**

***value*** Integer value

**Returns**

Minimum size of the the field in bits required to hold value.

#### 2.14.3.4 OSSIZE rtxLastBitSet (const OSOCTET * *pBits*, OSSIZE *numbits*)

This function returns the zero-counted index of the last bit set in a bit string.

**Parameters**

***pBits*** Pointer to the octets of the bit string.

***numbits*** The number of bits in the bit string.

**Returns**

Index of the last bit set in the bit string.

#### 2.14.3.5 int rtxSetBit (OSOCTET * *pBits*, OSSIZE *numbits*, OSSIZE *bitIndex*)

This function sets the specified zero-counted bit in the bit string.

**Parameters**

***pBits*** Pointer to octets of bit string.

***numbits*** Number of bits in the bit string.

***bitIndex*** Index of bit to be set. The bit with index 0 is a most significant bit in the octet with index 0.

**Returns**

If successful, returns the previous state of bit. If bit was set the return value is positive, if bit was not set the return value is zero. Otherwise, return value is an error code:

- RTERR_OUTOFBND = bitIndex is out of bounds

#### 2.14.3.6 OSUINT32 rtxSetBitFlags (OSUINT32 *flags*, OSUINT32 *mask*, OSBOOL *action*)

This function sets one or more bits to TRUE or FALSE in a 32-bit unsigned bit flag set.

**Parameters**

***flags*** Flags to which mask will be applied.

***mask*** Mask with one or more bits set that will be applied to pBitMask.

***action*** Boolean action indicating if bits in flags should be set (TRUE) or cleared (FALSE).

**Returns**

Updated flags after mask is applied.

#### 2.14.3.7 OSBOOL rtxTestBit (const OSOCTET * *pBits*, OSSIZE *numbits*, OSSIZE *bitIndex*)

This function tests the specified zero-counted bit in the bit string.

**Parameters**

***pBits*** Pointer to octets of bit string.

***numbits*** Number of bits in the bit string.

***bitIndex*** Index of bit to be tested. The bit with index 0 is a most significant bit in the octet with index 0.

**Returns**

True if bit set or false if not set or array index is beyond range of number of bits in the string.

## 2.15 Context Management Functions

### Classes

- struct OSRTErrLocn
- struct OSRTErrInfo
- struct OSRTErrInfoList
- struct OSRTBuffer
- struct OSRTBufSave
- struct OSCTXT

### Defines

- #define **OSRTERRSTKSIZ** 8
- #define **OSRTMAXERRPRM** 5
- #define **OSDIAG** 0x80000000
- #define **OSTRACE** 0x40000000
- #define **OSDISSTRM** 0x20000000
- #define **OSNOSTRMBACKOFF** 0x8000000
- #define **OS3GMOBORIG** 0x4000000
- #define **OSCDECL**
- #define **OSRT_GET_FIRST_ERROR_INFO**(pctxt)
- #define **OSRT_GET_LAST_ERROR_INFO**(pctxt)
- #define rtxCtxtGetMsgPtr(pctxt) (pctxt)->buffer.data
- #define rtxCtxtGetMsgLen(pctxt) (pctxt)->buffer.byteIndex
- #define rtxCtxtTestFlag(pctxt, mask) (((pctxt)->flags & mask) != 0)
- #define rtxCtxtPeekElemName(pctxt)
- #define rtxByteAlign(pctxt)
- #define rtxCtxtSetProtocolVersion(pctxt, value) (pctxt)->version = value
- #define **rtxMarkBitPos**(pctxt, ppos, pbitoff) (*(pbitoff) = (OSUINT8) (pctxt)->buffer.bitOffset, rtxMarkPos (pctxt, ppos))
- #define **rtxResetToBitPos**(pctxt, pos, bitoff) ((pctxt)->buffer.bitOffset = (OSUINT8) bitoff, rtxResetToPos (pctxt, pos))
- #define **RTXCTXTPUSHARRAYELEMNAME**(pctxt, name, idx) rtxCtxtPushArrayElemName(pctxt,OSUTF8(name),idx)
- #define **RTXCTXTPOPARRAYELEMNAME**(pctxt) rtxCtxtPopArrayElemName(pctxt)
- #define **RTXCTXTPUSHELEMNAME**(pctxt, name) rtxCtxtPushElemName(pctxt,OSUTF8(name))
- #define **RTXCTXTPOPELEMNAME**(pctxt) rtxCtxtPopElemName(pctxt)
- #define **RTXCTXTPUSHTYPENAME**(pctxt, name) rtxCtxtPushTypeName(pctxt,OSUTF8(name))
- #define **RTXCTXTPOPTYPENAME**(pctxt) rtxCtxtPopTypeName(pctxt)

### Typedefs

- typedef OSUINT32 **OSRTFLAGS**
- typedef int(* OSFreeCtxtAppInfoPtr )(struct OSCTXT *pctxt)
- typedef int(* OSResetCtxtAppInfoPtr )(struct OSCTXT *pctxt)
- typedef void(* OSFreeCtxtGlobalPtr )(struct OSCTXT *pctxt)
- typedef struct OSCTXT **OSCTXT**

## Functions

- int rtxInitContext (OSCTXT *pctxt)
- int rtxInitContextExt (OSCTXT *pctxt, OSMallocFunc malloc_func, OSReallocFunc realloc_func, OSFreeFunc free_func)
- int rtxInitThreadContext (OSCTXT *pctxt, const OSCTXT *pSrcCtxt)
- int rtxInitContextBuffer (OSCTXT *pctxt, OSOCTET *bufaddr, OSSIZE bufsiz)
- int rtxCtxtSetBufPtr (OSCTXT *pctxt, OSOCTET *bufaddr, OSSIZE bufsiz)
- OSSIZE rtxCtxtGetBitOffset (OSCTXT *pctxt)
- int rtxCtxtSetBitOffset (OSCTXT *pctxt, OSSIZE offset)
- OSSIZE rtxCtxtGetIOByteCount (OSCTXT *pctxt)
- int rtxCheckContext (OSCTXT *pctxt)
- void rtxFreeContext (OSCTXT *pctxt)
- void rtxCopyContext (OSCTXT *pdest, OSCTXT *psrc)
- void rtxCtxtSetFlag (OSCTXT *pctxt, OSUINT32 mask)
- void rtxCtxtClearFlag (OSCTXT *pctxt, OSUINT32 mask)
- int rtxCtxtPushArrayElemName (OSCTXT *pctxt, const OSUTF8CHAR *elemName, OSSIZE idx)
- int rtxCtxtPushElemName (OSCTXT *pctxt, const OSUTF8CHAR *elemName)
- int rtxCtxtPushTypeName (OSCTXT *pctxt, const OSUTF8CHAR *typeName)
- OSBOOL rtxCtxtPopArrayElemName (OSCTXT *pctxt)
- const OSUTF8CHAR * rtxCtxtPopElemName (OSCTXT *pctxt)
- const OSUTF8CHAR * rtxCtxtPopTypeName (OSCTXT *pctxt)
- int **rtxPreInitContext** (OSCTXT *pctxt)
- void rtxMemHeapSetFlags (OSCTXT *pctxt, OSUINT32 flags)
- void rtxMemHeapClearFlags (OSCTXT *pctxt, OSUINT32 flags)
- int rtxMarkPos (OSCTXT *pctxt, OSSIZE *ppos)
- int rtxResetToPos (OSCTXT *pctxt, OSSIZE pos)

### 2.15.1 Detailed Description

Context initialization functions handle the allocation, initialization, and destruction of context variables (variables of type OSCTXT). 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.

### 2.15.2 Define Documentation

#### 2.15.2.1 #define OSRT_GET_FIRST_ERROR_INFO(pctxt)

**Value:**

```
(((pctxt)->errInfo.list.head == 0) ? (OSRTErrInfo*)0 : \
(OSRTErrInfo*)((pctxt)->errInfo.list.head->data))
```

#### 2.15.2.2 #define OSRT_GET_LAST_ERROR_INFO(pctxt)

**Value:**

```
(((pctxt)->errInfo.list.tail == 0) ? (OSRTErrInfo*)0 : \
(OSRTErrInfo*)((pctxt)->errInfo.list.tail->data))
```

#### 2.15.2.3 #define rtxByteAlign(pctxt)

**Value:**

```
if ((pctxt)->buffer.bitOffset != 8) { \
(pctxt)->buffer.byteIndex++; (pctxt)->buffer.bitOffset = 8; }
```

This macro will byte-align the context buffer.

#### 2.15.2.4 #define rtxCtxtGetMsgLen(pctxt) (pctxt)->buffer.byteIndex

This macro returns the length of an encoded message. It will only work for in-memory encoding, not for encode to stream.

Note that this macro will not work with ASN.1 BER in-memory encoding. In this case, the BER-specific version of the function must be used.

**Parameters**

***pctxt*** Pointer to a context structure.

#### 2.15.2.5 #define rtxCtxtGetMsgPtr(pctxt) (pctxt)->buffer.data

This macro returns the start address of an encoded message. If a static buffer was used, this is simply the start address of the buffer. If dynamic encoding was done, this will return the start address of the dynamic buffer allocated by the encoder.

Note that this macro will not work with ASN.1 BER in-memory encoding. In this case, the BER-specific version of the function must be used.

**Parameters**

***pctxt*** Pointer to a context structure.

#### 2.15.2.6 #define rtxCtxtPeekElemName(pctxt)

**Value:**

```
(((pctxt)->elemNameStack.count > 0) ? \
 (const OSUTF8CHAR*)(pctxt)->elemNameStack.tail->data : (const OSUTF8CHAR*)0)
```

This macro returns the last element name from the context stack.

**Parameters**

***pctxt*** Pointer to a context structure.

**Returns**

Element name from top of stack or NULL if stack is empty.

#### 2.15.2.7 #define rtxCtxtSetProtocolVersion(pctxt, value) (pctxt)->version = value

This macro sets the protocol version in the context. This version number may be used in application code to do version specific operations. It is used in generated ASN.1 code with the extension addition version numbers to determine if an addition should be decoded.

For example, if this value is set to 8 and an extension addition group exists with version number 9 ([[ 9: ... ]]), its contents will not be decoded.

**Parameters**

***pctxt*** Pointer to a context structure.

***value*** The version number value.

#### 2.15.2.8 #define rtxCtxtTestFlag(pctxt, mask) (((pctxt)->flags & mask) != 0)

This macro tests if the given bit flag is set in the context.

**Parameters**

***pctxt*** - A pointer to a context structure.

***mask*** - Bit flag to be tested

### 2.15.3 Typedef Documentation

#### 2.15.3.1 typedef int(* OSFreeCtxtAppInfoPtr)(struct OSCTXT *pctxt)

OSRTFreeCtxtAppInfoPtr is a pointer to pctxt->pAppInfo free function, The pctxt->pAppInfo (pXMLInfo and pASN1Info) should contain the pointer to a structure and its first member should be a pointer to an appInfo free function.

#### 2.15.3.2 typedef void(* OSFreeCtxtGlobalPtr)(struct OSCTXT *pctxt)

OSRTFreeCtxtGlobalPtr is a pointer to a memory free function. This type describes the custom global memory free function generated by the compiler to free global nmemory. A pointer to a function of this type may be stored in the context gblFreeFunc field in order to free global data (pGlobalData) when rtxFreeContext is called.

#### 2.15.3.3 typedef int(* OSResetCtxtAppInfoPtr)(struct OSCTXT *pctxt)

OSRTResetCtxtAppInfoPtr is a pointer to pctxt->pAppInfo reset function, The pctxt->pAppInfo (pXMLInfo and pASN1Info) should contain the pointer to a structure and its second member should be a pointer to appInfo reset function.

### 2.15.4 Function Documentation

#### 2.15.4.1 int rtxCheckContext (OSCTXT * *pctxt*)

This function verifies that the given context structure is initialized and ready for use.

**Parameters**

***pctxt*** Pointer to a context structure.

**Returns**

Completion status of operation:

- 0 = success,
- RTERR_NOTINIT status code if not initialized

#### 2.15.4.2 void rtxCopyContext (OSCTXT * *pdest*, OSCTXT * *psrc*)

This function creates a copy of a context structure. The copy is a "shallow copy" (i.e. new copies of dynamic memory blocks held within the context are not made, only the pointers are transferred to the new context structure). This function is mainly for use from within compiler-generated code.

**Parameters**

***pdest*** - Context structure to which data is to be copied.

***psrc*** - Context structure from which data is to be copied.

#### 2.15.4.3 void rtxCtxtClearFlag (OSCTXT * *pctxt*, OSUINT32 *mask*)

This function is used to clear a processing flag within the context structure.

**Parameters**

***pctxt*** - A pointer to a context structure.

***mask*** - Mask containing bit(s) to be cleared.

#### 2.15.4.4 OSSIZE rtxCtxtGetBitOffset (OSCTXT * *pctxt*)

This function returns the total bit offset to the current element in the context buffer.

**Parameters**

***pctxt*** Pointer to a context structure.

**Returns**

Bit offset.

#### 2.15.4.5 OSSIZE rtxCtxtGetIOByteCount (OSCTXT * *pctxt*)

This function returns the count of bytes either written to a stream or memory buffer.

**Parameters**

***pctxt*** Pointer to a context structure.

**Returns**

I/O byte count.

#### 2.15.4.6 OSBOOL rtxCtxtPopArrayElemName (OSCTXT * *pctxt*)

This function pops the last element name from the context stack. This name is assumed to be an array element name pushed by the rtxCtxtPushArrayElemName function. The name is therefore dynamic and memory is freed for it using the rtxMemFreePtr function.

**Parameters**

***pctxt*** Pointer to a context structure.

**Returns**

True if name popped from stack or false if stack is empty.

#### 2.15.4.7 const OSUTF8CHAR* rtxCtxtPopElemName (OSCTXT * *pctxt*)

This function pops the last element name from the context stack.

**Parameters**

***pctxt*** Pointer to a context structure.

**Returns**

Element name popped from stack or NULL if stack is empty.

#### 2.15.4.8 const OSUTF8CHAR* rtxCtxtPopTypeName (OSCTXT * *pctxt*)

This function pops the type name from the context stack. The name is only popped if the item count is one.

**Parameters**

***pctxt*** Pointer to a context structure.

**Returns**

Type name popped from stack or NULL if stack count not equal to one.

#### 2.15.4.9 int rtxCtxtPushArrayElemName (OSCTXT * *pctxt*, const OSUTF8CHAR * *elemName*, OSSIZE *idx*)

This function is used to push an array element name onto the context element name stack. The name is formed by combining the given element name with the index to create a name of format name[index]. Dynamic memory is allocated for the resulting name using the rtxMemAlloc function.

**Parameters**

***pctxt*** Pointer to a context structure.

***elemName*** Name of element to be pushed on stack.

***idx*** Index or the array element.

**Returns**

Completion status of operation:

- 0 = success,
- RTERR_NOMEM if mem alloc for name fails.

#### 2.15.4.10 int rtxCtxtPushElemName (OSCTXT * *pctxt*, const OSUTF8CHAR * *elemName*)

This function is used to push an element name onto the context element name stack.

**Parameters**

***pctxt*** Pointer to a context structure.

***elemName*** Name of element to be pushed on stack. Note that a copy of the name is not made, the pointer to the name that is passed is stored.

**Returns**

Completion status of operation:

- 0 = success,
- RTERR_NOMEM if mem alloc for name fails.

#### 2.15.4.11 int rtxCtxtPushTypeName (OSCTXT * *pctxt*, const OSUTF8CHAR * *typeName*)

This function is used to push a type name onto the context element name stack. The name is only added for the top-level type. This is determined by testing to ensure that there are no existing names on the stack.

**Parameters**

***pctxt*** Pointer to a context structure.

***typeName*** Name of type to be pushed on stack. Note that a copy of the name is not made, the pointer to the name that is passed is stored.

**Returns**

Completion status of operation:

- 0 = success,
- RTERR_NOMEM if mem alloc for name fails.

#### 2.15.4.12 int rtxCtxtSetBitOffset (OSCTXT * *pctxt*, OSSIZE *offset*)

This function sets the bit offset in the context to the given value.

**Parameters**

***pctxt*** Pointer to a context structure.

***offset*** Bit offset.

**Returns**

Completion status of operation:

- 0 = success,
- Negative status code if error

#### 2.15.4.13 int rtxCtxtSetBufPtr (OSCTXT * *pctxt*, OSOCTET * *bufaddr*, OSSIZE *bufsiz*)

This function is used to set the internal buffer pointer for in-memory encoding or decoding. It must be called after the context variable is initialized before any other compiler generated or run-time library encode function.

**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.

***bufaddr*** A pointer to a memory buffer to use to encode a message or that holds a message to be decoded. The buffer should be declared as an array of unsigned characters (OCTETs). This parameter can be set to NULL to specify dynamic encoding (i.e., the encode functions will dynamically allocate a buffer to hold the encoded message).

***bufsiz*** The length of the memory buffer in bytes. Should be set to zero if NULL was specified for bufaddr (i.e. dynamic encoding was selected).

#### 2.15.4.14 void rtxCtxtSetFlag (OSCTXT * *pctxt*, OSUINT32 *mask*)

This function is used to set a processing flag within the context structure.

**Parameters**

***pctxt*** - A pointer to a context structure.

***mask*** - Mask containing bit(s) to be set.

#### 2.15.4.15 void rtxFreeContext (OSCTXT * *pctxt*)

This function frees all dynamic memory associated with a context. This includes all memory allocated using the rtxMem functions using the given context parameter.

**Parameters**

***pctxt*** Pointer to a context structure.

#### 2.15.4.16 int rtxInitContext (OSCTXT * *pctxt*)

This function initializes an OSCTXT block. It sets all key working parameters to their correct initial state values. It is required that this function be invoked before using a context variable.

**Parameters**

***pctxt*** Pointer to the context structure variable to be initialized.

**Returns**

Completion status of operation:

- 0 = success,
- negative return value is error.

#### 2.15.4.17 int rtxInitContextBuffer (OSCTXT * *pctxt*, OSOCTET * *bufaddr*, OSSIZE *bufsiz*)

This function assigns a message buffer to a context block. The block should have been previously initialized by rtxInitContext.

**Parameters**

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

***bufaddr*** For encoding, the address of a memory buffer to receive the encoded message. If this address is NULL (0), encoding to a dynamic buffer will be done. For decoding, the address of a buffer that contains the message data to be decoded.

***bufsiz*** The size of the memory buffer. For encoding, this argument may be set to zero to indicate a dynamic memory buffer should be used.

**Returns**

Completion status of operation:

- 0 = success,
- negative return value is error.

#### 2.15.4.18 int rtxInitContextExt (OSCTXT * *pctxt*, OSMallocFunc *malloc_func*, OSReallocFunc *realloc_func*, OSFreeFunc *free_func*)

This function initializes an OSCTXT block. It sets all key working parameters to their correct initial state values. It is required that this function be invoked before using a context variable.

**Parameters**

***pctxt*** Pointer to the context structure variable to be initialized.

***malloc_func*** Pointer to the memory allocation function.

***realloc_func*** Pointer to the memory reallocation function.

***free_func*** Pointer to the memory deallocation function.

**Returns**

Completion status of operation:

- 0 = success,
- negative return value is error.

#### 2.15.4.19 int rtxInitThreadContext (OSCTXT * *pctxt*, const OSCTXT * *pSrcCtxt*)

This function initializes a context for use in a thread. It is the same as rtxInitContext except that it copies the pointer to constant data from the given source context into the newly initialized thread context. It is assumed that the source context has been initialized and the custom generated global initialization function has been called. The main purpose of this function is to prevent multiple copies of global static data from being created within different threads.

**Parameters**

***pctxt*** Pointer to the context structure variable to be initialized.

***pSrcCtxt*** Pointer to source context which has been fully initialized including a pointer to global constant data initialized via a call to a generated 'Init_<project>_Global' function.

**Returns**

Completion status of operation:

- 0 = success,
- negative return value is error.

#### 2.15.4.20 int rtxMarkPos (OSCTXT * *pctxt*, OSSIZE * *ppos*)

This function saves the current position in a message buffer or stream.

**Parameters**

***pctxt*** Pointer to a context block.

***ppos*** Pointer to saved position.

**Returns**

Completion status of operation:

- 0 = success,
- negative return value is error.

#### 2.15.4.21 void rtxMemHeapClearFlags (OSCTXT * *pctxt*, OSUINT32 *flags*)

This function clears memory heap flags.

**Parameters**

***pctxt*** Pointer to a memory block structure that contains the list of dynamic memory block maintained by these functions.

***flags*** The flags

#### 2.15.4.22 void rtxMemHeapSetFlags (OSCTXT * *pctxt*, OSUINT32 *flags*)

This function sets flags to a heap. May be used to control the heap's behavior.

**Parameters**

***pctxt*** Pointer to a memory block structure that contains the list of dynamic memory block maintained by these functions.

***flags*** The flags.

#### 2.15.4.23 int rtxResetToPos (OSCTXT * *pctxt*, OSSIZE *pos*)

This function resets a message buffer or stream back to the given position.

**Parameters**

***pctxt*** Pointer to a context block.

***pos*** Context position.

**Returns**

Completion status of operation:

- 0 = success,
- negative return value is error.

## 2.16 Memory Allocation Macros and Functions

### Defines

- #define OSRTALLOCTYPE(pctxt, type) (type∗) rtxMemHeapAlloc (&(pctxt)->pMemHeap, sizeof(type))
- #define OSRTALLOCTYPEZ(pctxt, type) (type∗) rtxMemHeapAllocZ (&(pctxt)->pMemHeap, sizeof(type))
- #define OSRTREALLOCARRAY(pctxt, pseqof, type)
- #define **OSCRTMALLOC0**(nbytes) malloc(nbytes)
- #define **OSCRTFREE0**(ptr) free(ptr)
- #define **OSCRTMALLOC** rtxMemAlloc
- #define **OSCRTFREE** rtxMemFreePtr
- #define rtxMemAlloc(pctxt, nbytes) rtxMemHeapAlloc(&(pctxt)->pMemHeap,nbytes)
- #define rtxMemSysAlloc(pctxt, nbytes) rtxMemHeapSysAlloc(&(pctxt)->pMemHeap,nbytes)
- #define rtxMemAllocZ(pctxt, nbytes) rtxMemHeapAllocZ(&(pctxt)->pMemHeap,nbytes)
- #define rtxMemSysAllocZ(pctxt, nbytes) rtxMemHeapSysAllocZ(&(pctxt)->pMemHeap,nbytes)
- #define rtxMemRealloc(pctxt, mem_p, nbytes) rtxMemHeapRealloc(&(pctxt)->pMemHeap, (void∗)mem_p, nbytes)
- #define rtxMemSysRealloc(pctxt, mem_p, nbytes) rtxMemHeapSysRealloc(&(pctxt)->pMemHeap,(void∗)mem_p,nbytes)
- #define rtxMemFreePtr(pctxt, mem_p) rtxMemHeapFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)
- #define rtxMemSysFreePtr(pctxt, mem_p) rtxMemHeapSysFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)
- #define rtxMemAllocType(pctxt, ctype) (ctype∗)rtxMemHeapAlloc(&(pctxt)->pMemHeap,sizeof(ctype))
- #define rtxMemSysAllocType(pctxt, ctype) (ctype∗)rtxMemHeapSysAlloc(&(pctxt)->pMemHeap,sizeof(ctype))
- #define rtxMemAllocTypeZ(pctxt, ctype) (ctype∗)rtxMemHeapAllocZ(&(pctxt)->pMemHeap,sizeof(ctype))
- #define rtxMemSysAllocTypeZ(pctxt, ctype) (ctype∗)rtxMemHeapSysAllocZ(&(pctxt)->pMemHeap,sizeof(ctype))
- #define rtxMemFreeType(pctxt, mem_p) rtxMemHeapFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)
- #define rtxMemSysFreeType(pctxt, mem_p) rtxMemHeapSysFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)
- #define rtxMemAllocArray(pctxt, n, type) (type∗)rtxMemHeapAlloc (&(pctxt)->pMemHeap, sizeof(type)∗n)
- #define rtxMemSysAllocArray(pctxt, n, type) (type∗)rtxMemHeapSysAlloc (&(pctxt)->pMemHeap, sizeof(type)∗n)
- #define rtxMemAllocArrayZ(pctxt, n, type) (type∗)rtxMemHeapAllocZ (&(pctxt)->pMemHeap, sizeof(type)∗n)
- #define rtxMemFreeArray(pctxt, mem_p) rtxMemHeapFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)
- #define rtxMemSysFreeArray(pctxt, mem_p) rtxMemHeapSysFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)
- #define rtxMemReallocArray(pctxt, mem_p, n, type) (type∗)rtxMemHeapRealloc(&(pctxt)->pMemHeap, (void∗)mem_p, sizeof(type)∗n)
- #define rtxMemNewAutoPtr(pctxt, nbytes) rtxMemHeapAlloc(&(pctxt)->pMemHeap, nbytes)
- #define rtxMemAutoPtrRef(pctxt, ptr) rtxMemHeapAutoPtrRef(&(pctxt)->pMemHeap, (void∗)(ptr))
- #define rtxMemAutoPtrUnref(pctxt, ptr) rtxMemHeapAutoPtrUnref(&(pctxt)->pMemHeap, (void∗)(ptr))
- #define rtxMemAutoPtrGetRefCount(pctxt, ptr) rtxMemHeapAutoPtrGetRefCount(&(pctxt)->pMemHeap, (void∗)(ptr))
- #define rtxMemCheckPtr(pctxt, mem_p) rtxMemHeapCheckPtr(&(pctxt)->pMemHeap, (void∗)mem_p)
- #define rtxMemCheck(pctxt) rtxMemHeapCheck(&(pctxt)->pMemHeap, __FILE__, __LINE__)
- #define rtxMemPrint(pctxt) rtxMemHeapPrint(&(pctxt)->pMemHeap)
- #define rtxMemSetProperty(pctxt, propId, pProp) rtxMemHeapSetProperty (&(pctxt)->pMemHeap, propId, pProp)

## Functions

- void **rtxMemHeapAddRef** (void **ppvMemHeap)
- void * **rtxMemHeapAlloc** (void **ppvMemHeap, size_t nbytes)
- void * **rtxMemHeapAllocZ** (void **ppvMemHeap, size_t nbytes)
- void * **rtxMemHeapSysAlloc** (void **ppvMemHeap, size_t nbytes)
- void * **rtxMemHeapSysAllocZ** (void **ppvMemHeap, size_t nbytes)
- int **rtxMemHeapCheckPtr** (void **ppvMemHeap, const void *mem_p)
- int **rtxMemHeapCreate** (void **ppvMemHeap)
- int **rtxMemHeapCreateExt** (void **ppvMemHeap, OSMallocFunc malloc_func, OSReallocFunc realloc_-func, OSFreeFunc free_func)
- void **rtxMemHeapFreeAll** (void **ppvMemHeap)
- void **rtxMemHeapFreePtr** (void **ppvMemHeap, void *mem_p)
- void **rtxMemHeapSysFreePtr** (void **ppvMemHeap, void *mem_p)
- void * **rtxMemHeapRealloc** (void **ppvMemHeap, void *mem_p, size_t nbytes_)
- void * **rtxMemHeapSysRealloc** (void **ppvMemHeap, void *mem_p, size_t nbytes_)
- void **rtxMemHeapRelease** (void **ppvMemHeap)
- void **rtxMemHeapReset** (void **ppvMemHeap)
- void **rtxMemHeapSetProperty** (void **ppvMemHeap, OSUINT32 propId, void *pProp)
- void * **rtxMemNewArray** (size_t nbytes)
- void * **rtxMemNewArrayZ** (size_t nbytes)
- void **rtxMemDeleteArray** (void *mem_p)
- void * **rtxMemHeapAutoPtrRef** (void **ppvMemHeap, void *ptr)
- int **rtxMemHeapAutoPtrUnref** (void **ppvMemHeap, void *ptr)
- int **rtxMemHeapAutoPtrGetRefCount** (void **ppvMemHeap, void *mem_p)
- void **rtxMemHeapInvalidPtrHook** (void **ppvMemHeap, const void *mem_p)
- void **rtxMemHeapCheck** (void **ppvMemHeap, const char *file, int line)
- void **rtxMemHeapPrint** (void **ppvMemHeap)
- void rtxMemSetAllocFuncs (OSMallocFunc malloc_func, OSReallocFunc realloc_func, OSFreeFunc free_-func)
- void **rtxMemFreeOpenSeqExt** (OSCTXT *pctxt, struct OSRTDList *pElemList)
- OSUINT32 rtxMemHeapGetDefBlkSize (OSCTXT *pctxt)
- void rtxMemSetDefBlkSize (OSUINT32 blkSize)
- OSUINT32 rtxMemGetDefBlkSize ()
- OSBOOL rtxMemHeapIsEmpty (OSCTXT *pctxt)
- OSBOOL rtxMemIsZero (const void *pmem, size_t memsiz)
- void rtxMemFree (OSCTXT *pctxt)
- void rtxMemReset (OSCTXT *pctxt)

### 2.16.1 Detailed Description

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

## 2.16.2 Define Documentation

### 2.16.2.1 #define OSRTALLOCTYPE(pctxt, type) (type*) rtxMemHeapAlloc (&(pctxt)->pMemHeap, sizeof(type))

This macro allocates a single element of the given type.

**Parameters**

***pctxt*** - Pointer to a context block

***type*** - Data type of record to allocate

### 2.16.2.2 #define OSRTALLOCTYPEZ(pctxt, type) (type*) rtxMemHeapAllocZ (&(pctxt)->pMemHeap, sizeof(type))

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

### 2.16.2.3 #define OSRTREALLOCARRAY(pctxt, pseqof, type)

**Value:**

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

Reallocate an array. This macro reallocates an array (either expands or contracts) to hold the given number of elements. The number of elements is specified in the *n* member variable of the pseqof argument.

**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

### 2.16.2.4 #define rtxMemAlloc(pctxt, nbytes) rtxMemHeapAlloc(&(pctxt)->pMemHeap,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.

#### 2.16.2.5 #define rtxMemAllocArray(pctxt, n, type) (type∗)rtxMemHeapAlloc (&(pctxt)->pMemHeap, sizeof(type)∗n)

Allocate a dynamic array. This macro allocates a dynamic array of records of the given type. The pointer to the allocated array is returned to the caller.

**Parameters**

***pctxt*** - Pointer to a context block

***n*** - Number of records to allocate

***type*** - Data type of an array record

#### 2.16.2.6 #define rtxMemAllocArrayZ(pctxt, n, type) (type∗)rtxMemHeapAllocZ (&(pctxt)->pMemHeap, sizeof(type)∗n)

Allocate a dynamic array and zero memory. This macro allocates a dynamic array of records of the given type and writes zeros over the allocated memory. The pointer to the allocated array is returned to the caller.

**Parameters**

***pctxt*** - Pointer to a context block

***n*** - Number of records to allocate

***type*** - Data type of an array record

#### 2.16.2.7 #define rtxMemAllocType(pctxt, ctype) (ctype∗)rtxMemHeapAlloc(&(pctxt)->pMemHeap,sizeof(ctype))

Allocate type. This macro allocates memory to hold a variable of the given type.

**Parameters**

***pctxt*** - Pointer to a context block

***ctype*** - Name of C typedef

**Returns**

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

#### 2.16.2.8 #define rtxMemAllocTypeZ(pctxt, ctype) (ctype∗)rtxMemHeapAllocZ(&(pctxt)->pMemHeap,sizeof(ctype))

Allocate type and zero memory. This macro allocates memory to hold a variable of the given type and initializes the allocated memory to zero.

**Parameters**

***pctxt*** - Pointer to a context block

***ctype*** - Name of C typedef

**Returns**

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

#### 2.16.2.9 #define rtxMemAllocZ(pctxt, nbytes) rtxMemHeapAllocZ(&(pctxt)->pMemHeap,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.

#### 2.16.2.10 #define rtxMemAutoPtrGetRefCount(pctxt, ptr) rtxMemHeapAutoPtrGetRefCount(&(pctxt)->pMemHeap, (void*)(ptr))

This function returns the reference count of the given pointer. goes to zero, the memory is freed.

**Parameters**

***pctxt*** Pointer to a context structure.

***ptr*** Pointer on which reference count is to be fetched.

**Returns**

Pointer reference count.

#### 2.16.2.11 #define rtxMemAutoPtrRef(pctxt, ptr) rtxMemHeapAutoPtrRef(&(pctxt)->pMemHeap, (void*)(ptr))

This function increments the auto-pointer reference count.

**Parameters**

***pctxt*** Pointer to a context structure.

***ptr*** Pointer on which reference count is to be incremented.

**Returns**

Referenced pointer value (ptr argument) or NULL if reference count could not be incremented.

#### 2.16.2.12 #define rtxMemAutoPtrUnref(pctxt, ptr) rtxMemHeapAutoPtrUnref(&(pctxt)->pMemHeap, (void*)(ptr))

This function decrements the auto-pointer reference count. If the count goes to zero, the memory is freed.

**Parameters**

***pctxt*** Pointer to a context structure.

***ptr*** Pointer on which reference count is to be decremented.

**Returns**

Positive reference count or a negative error code. If zero, memory held by pointer will have been freed.

#### 2.16.2.13 #define rtxMemCheck(pctxt) rtxMemHeapCheck(&(pctxt)->pMemHeap, __FILE__, __LINE__)

Check memory heap.

**Parameters**

***pctxt*** - Pointer to a context block

#### 2.16.2.14 #define rtxMemCheckPtr(pctxt, mem_p) rtxMemHeapCheckPtr(&(pctxt)->pMemHeap, (void*)mem_p)

Check memory pointer. This macro check pointer on presence in heap.

**Parameters**

***pctxt*** - Pointer to a context block

***mem_p*** - Pointer to memory block.

**Returns**

1 - pointer refer to memory block in heap; 0 - poiter refer not memory heap block.

#### 2.16.2.15 #define rtxMemFreeArray(pctxt, mem_p) rtxMemHeapFreePtr(&(pctxt)->pMemHeap, (void*)mem_p)

Free memory pointer. This macro frees memory at the given pointer. The memory must have been allocated using the rtxMemAlloc (or similar) macros or the rtxMem 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 rtxMemAlloc or rtxMemAlloc macro or the rtxMemHeapAlloc function.

#### 2.16.2.16 #define rtxMemFreePtr(pctxt, mem_p) rtxMemHeapFreePtr(&(pctxt)->pMemHeap, (void*)mem_p)

Free memory pointer. This macro frees memory at the given pointer. The memory must have been allocated using the rtxMemAlloc (or similar) macros or the rtxMem 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 rtxMemAlloc macro or the rtxMemHeapAlloc function.

#### 2.16.2.17 #define rtxMemFreeType(pctxt, mem_p) rtxMemHeapFreePtr(&(pctxt)->pMemHeap, (void*)mem_p)

Free memory pointer. This macro frees memory at the given pointer. The memory must have been allocated using the rtxMemAlloc (or similar) macros or the rtxMem 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 rtxMemAlloc or rtxMemAlloc macro or the rtxMemHeapAlloc function.

#### 2.16.2.18 #define rtxMemNewAutoPtr(pctxt, nbytes) rtxMemHeapAlloc(&(pctxt)->pMemHeap, nbytes)

This function allocates a new block of memory and creates an auto-pointer with reference count set to one. The `rtxMemAutoPtrRef` and `rtxMemAutoPtrUnref` functions can be used to increment and decrement the reference count. When the count goes to zero, the memory held by the pointer is freed.

**Parameters**

***pctxt*** Pointer to a context structure.

***nbytes*** Number of bytes to allocate.

**Returns**

Pointer to allocated memory or NULL if not enough memory is available.

#### 2.16.2.19 #define rtxMemPrint(pctxt) rtxMemHeapPrint(&(pctxt)->pMemHeap)

Print memory heap structure to stderr.

**Parameters**

***pctxt*** - Pointer to a context block

#### 2.16.2.20 #define rtxMemRealloc(pctxt, mem_p, nbytes) rtxMemHeapRealloc(&(pctxt)->pMemHeap, (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 rtxMemAlloc macro or the rtxMemHeapAlloc 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 mem_p pointer that was passed in if the block did not need to be relocated.

#### 2.16.2.21 #define rtxMemReallocArray(pctxt, mem_p, n, type) (type*)rtxMemHeapRealloc(&(pctxt)->pMemHeap, (void*)mem_p, sizeof(type)*n)

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 rtxMemAlloc macro or the rtxMemHeapAlloc function.

***n*** - Number of items of the given type to be allocated.

***type*** - Array element data type (for example, int).

**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.

#### 2.16.2.22 #define rtxMemSetProperty(pctxt, propId, pProp) rtxMemHeapSetProperty (&(pctxt)->pMemHeap, propId, pProp)

Set memory heap property.

**Parameters**

***pctxt*** - Pointer to a context block

***propId*** - Property Id.

***pProp*** - Pointer to property value.

#### 2.16.2.23 #define rtxMemSysAlloc(pctxt, nbytes) rtxMemHeapSysAlloc(&(pctxt)->pMemHeap,nbytes)

This macro makes a direct call to the configured system memory allocation function. By default, this is the C malloc function, but it is possible to configure to use a custom allocation 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.

#### 2.16.2.24 #define rtxMemSysAllocArray(pctxt, n, type) (type*)rtxMemHeapSysAlloc (&(pctxt)->pMemHeap, sizeof(type)*n)

Allocate a dynamic array. This macro allocates a dynamic array of records of the given type. The pointer to the allocated array is returned to the caller.

This macro makes a direct call to the configured system memory allocation function. By default, this is the C malloc function, but it is possible to configure to use a custom allocation function.

**Parameters**

***pctxt*** - Pointer to a context block

***n*** - Number of records to allocate

***type*** - Data type of an array record

#### 2.16.2.25 #define rtxMemSysAllocType(pctxt, ctype) (ctype∗)rtxMemHeapSysAlloc(&(pctxt)->pMemHeap,sizeof(ctype))

Allocate type. This macro allocates memory to hold a variable of the given type.

This macro makes a direct call to the configured system memory allocation function. By default, this is the C malloc function, but it is possible to configure to use a custom allocation function.

**Parameters**

***pctxt*** - Pointer to a context block

***ctype*** - Name of C typedef

**Returns**

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

#### 2.16.2.26 #define rtxMemSysAllocTypeZ(pctxt, ctype) (ctype∗)rtxMemHeapSysAllocZ(&(pctxt)->pMemHeap,sizeof(ctype))

Allocate type and zero memory. This macro allocates memory to hold a variable of the given type and initializes the allocated memory to zero.

This macro makes a direct call to the configured system memory allocation function. By default, this is the C malloc function, but it is possible to configure to use a custom allocation function.

**Parameters**

***pctxt*** - Pointer to a context block

***ctype*** - Name of C typedef

**Returns**

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

#### 2.16.2.27 #define rtxMemSysAllocZ(pctxt, nbytes) rtxMemHeapSysAllocZ(&(pctxt)->pMemHeap,nbytes)

Allocate and zero memory. This macro allocates the given number of bytes and then initializes the memory block to zero.

This macro makes a direct call to the configured system memory allocation function. By default, this is the C malloc function, but it is possible to configure to use a custom allocation 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.

#### 2.16.2.28 #define rtxMemSysFreeArray(pctxt, mem_p) rtxMemHeapSysFreePtr(&(pctxt)->pMemHeap, (void*)mem_p)

Free memory pointer. This macro frees memory at the given pointer. The memory must have been allocated using the rtxMemSysAlloc (or similar) macros or the rtxMemSys 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 rtxMemSysAlloc or rtxMemSysAlloc macro or the rtxMemSysHeapAlloc function.

#### 2.16.2.29 #define rtxMemSysFreePtr(pctxt, mem_p) rtxMemHeapSysFreePtr(&(pctxt)->pMemHeap, (void*)mem_p)

This macro makes a direct call to the configured system memory free function. By default, this is the C free function, but it is possible to configure to use a custom free function.

**Parameters**

***pctxt*** - Pointer to a context block

***mem_p*** - Pointer to memory block to free. This must have been allocated using the rtxMemSysAlloc macro or the rtxMemHeapSysAlloc function.

#### 2.16.2.30 #define rtxMemSysFreeType(pctxt, mem_p) rtxMemHeapSysFreePtr(&(pctxt)->pMemHeap, (void*)mem_p)

Free memory pointer. This macro frees memory at the given pointer. The memory must have been allocated using the rtxMemSysAlloc (or similar) macros or the rtxMemSys 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 rtxMemSysAlloc or rtxMemSysAlloc macro or the rtxMemSysHeapAlloc function.

#### 2.16.2.31 #define rtxMemSysRealloc(pctxt, mem_p, nbytes) rtxMemHeapSysRealloc(&(pctxt)->pMemHeap,(void*)mem_p,nbytes)

This macro makes a direct call to the configured system memory reallocation function to do the reallocation.. By default, this is the C realloc function, but it is possible to configure to use a custom reallocation function.

**Parameters**

***pctxt*** - Pointer to a context block

***mem_p*** - Pointer to memory block to reallocate. This must have been allocated using the rtxMemSysAlloc macro or the rtxMemHeapSysAlloc 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 mem_p pointer that was passed in if the block did not need to be relocated.

### 2.16.3 Function Documentation

#### 2.16.3.1 void rtxMemFree (OSCTXT * *pctxt*)

Free memory associated with a context. This macro frees all memory held within a context. This is all memory allocated using the rtxMemAlloc (and similar macros) and the rtxMem memory allocation functions using the given context variable.

**Parameters**

***pctxt*** - Pointer to a context block

#### 2.16.3.2 OSUINT32 rtxMemGetDefBlkSize ()

This function returns the actual granularity of memory blocks.

**Returns**

The currently used minimum size and the granularity of memory blocks.

#### 2.16.3.3 OSUINT32 rtxMemHeapGetDefBlkSize (OSCTXT * *pctxt*)

This function returns the actual granularity of memory blocks in the context.

**Parameters**

***pctxt*** Pointer to a context block.

#### 2.16.3.4 OSBOOL rtxMemHeapIsEmpty (OSCTXT * *pctxt*)

This function determines if the memory heap defined in the give context is empty (i.e. contains no outstanding memory allocations).

**Parameters**

***pctxt*** Pointer to a context block.

**Returns**

Boolean true value if heap is empty.

#### 2.16.3.5 OSBOOL rtxMemIsZero (const void * *pmem*, size_t *memsiz*)

This helper function determines if an arbitrarily sized block of memory is set to zero.

**Parameters**

***pmem*** Pointer to memory block to check

***memsiz*** Size of the memory block

**Returns**

Boolean result: true if memory is all zero

#### 2.16.3.6 void rtxMemReset (OSCTXT * *pctxt*)

Reset memory associated with a context. This macro resets all memory held within a context. This is all memory allocated using the rtxMemAlloc (and similar macros) and the rtxMem memory allocation functions using the given context variable.

The difference between this and the OSMEMFREE 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

#### 2.16.3.7 void rtxMemSetAllocFuncs (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 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 the standard functions.

**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).

#### 2.16.3.8 void rtxMemSetDefBlkSize (OSUINT32 *blkSize*)

This function sets the minimum size and the granularity of memory blocks for newly created memory heaps.

**Parameters**

***blkSize*** The minimum size and the granularity of memory blocks.

## 2.17 Memory Buffer Management Functions

### Classes

- struct OSRTMEMBUF

### Defines

- #define **OSMBDFLTSEGSIZE** 1024
- #define **OSMEMBUFPTR**(pmb) ((pmb)->buffer + (pmb)->startidx)
- #define **OSMEMBUFENDPTR**(pmb) ((pmb)->buffer + (pmb)->startidx + (pmb)->usedcnt)
- #define **OSMEMBUFUSEDSIZE**(pmb) ((OSSIZE)(pmb)->usedcnt)
- #define **OSMBAPPENDSTR**(pmb, str)
- #define **OSMBAPPENDSTRL**(pmb, str) rtxMemBufAppend(pmb,(OSOCTET*)str,OSCRTLSTRLEN(str))
- #define **OSMBAPPENDUTF8**(pmb, str)

### Typedefs

- typedef struct OSRTMEMBUF **OSRTMEMBUF**

### Functions

- int rtxMemBufAppend (OSRTMEMBUF *pMemBuf, const OSOCTET *pdata, OSSIZE nbytes)
- int rtxMemBufCut (OSRTMEMBUF *pMemBuf, OSSIZE fromOffset, OSSIZE nbytes)
- void rtxMemBufFree (OSRTMEMBUF *pMemBuf)
- OSOCTET * rtxMemBufGetData (const OSRTMEMBUF *pMemBuf, int *length)
- OSOCTET * rtxMemBufGetDataExt (const OSRTMEMBUF *pMemBuf, OSSIZE *length)
- OSSIZE rtxMemBufGetDataLen (const OSRTMEMBUF *pMemBuf)
- void rtxMemBufInit (OSCTXT *pCtxt, OSRTMEMBUF *pMemBuf, OSSIZE segsize)
- void rtxMemBufInitBuffer (OSCTXT *pCtxt, OSRTMEMBUF *pMemBuf, OSOCTET *buf, OSSIZE bufsize, OSSIZE segsize)
- int rtxMemBufPreAllocate (OSRTMEMBUF *pMemBuf, OSSIZE nbytes)
- void rtxMemBufReset (OSRTMEMBUF *pMemBuf)
- int rtxMemBufSet (OSRTMEMBUF *pMemBuf, OSOCTET value, OSSIZE nbytes)
- OSBOOL rtxMemBufSetExpandable (OSRTMEMBUF *pMemBuf, OSBOOL isExpandable)
- OSBOOL rtxMemBufSetUseSysMem (OSRTMEMBUF *pMemBuf, OSBOOL value)
- OSSIZE rtxMemBufTrimW (OSRTMEMBUF *pMemBuf)

### 2.17.1 Detailed Description

Memory buffer management functions handle the allocation, expansion, and deallocation of dynamic memory buffers used by some encode/decode functions. Dynamic memory buffers are buffers that can grow or shrink to hold variable sized amounts of data. This group of functions allows data to be appended to buffers, to be set within buffers, and to be retrieved from buffers. Currently, these functions are used within the generated SAX decode routines to collect data as it is parsed by an XML parser.

## 2.17.2 Define Documentation

### 2.17.2.1 #define OSMBAPPENDSTR(pmb, str)

**Value:**

```
if (0 != str) \
rtxMemBufAppend(pmb,(OSOCTET*)str,OSCRTLSTRLEN(str))
```

### 2.17.2.2 #define OSMBAPPENDUTF8(pmb, str)

**Value:**

```
if (0 != str) \
rtxMemBufAppend(pmb,(OSOCTET*)str,rtxUTF8LenBytes(str))
```

## 2.17.3 Function Documentation

### 2.17.3.1 int rtxMemBufAppend (OSRTMEMBUF * *pMemBuf*, const OSOCTET * *pdata*, OSSIZE *nbytes*)

This function appends the data to the end of a memory buffer. If the buffer was dynamic and full then the buffer will be reallocated. If it is static (the static buffer was assigned by a call to rtxMemBufInitBuffer) or it is empty (no memory previously allocated) then a new buffer will be allocated.

**Parameters**

***pMemBuf*** A pointer to a memory buffer structure.

***pdata*** The pointer to the buffer to be appended. The data will be copied at the end of the memory buffer.

***nbytes*** The number of bytes to be copied from pData.

**Returns**

Completion status of operation:

- 0 = success,
- negative return value is error.

### 2.17.3.2 int rtxMemBufCut (OSRTMEMBUF * *pMemBuf*, OSSIZE *fromOffset*, OSSIZE *nbytes*)

This function cuts off the part of memory buffer. The beginning of the cutting area is specified by offset "fromOffset" and the length is specified by "nbytes". All data in this part will be lost. The data from the offset "fromOffset + nbytes" will be moved to "fromOffset" offset.

**Parameters**

***pMemBuf*** A pointer to a memory buffer structure.

***fromOffset*** The offset of the beginning part, being cut off.

***nbytes*** The number of bytes to be cut off from the memory buffer.

**Returns**

Completion status of operation:

- 0 = success,
- negative return value is error.

#### 2.17.3.3 void rtxMemBufFree (OSRTMEMBUF * *pMemBuf*)

This function frees the memory buffer. If memory was allocated then it will be freed. Do not use the memory buffer structure after this function is called.

**Parameters**

***pMemBuf*** A pointer to a memory buffer structure.

#### 2.17.3.4 OSOCTET* rtxMemBufGetData (const OSRTMEMBUF * *pMemBuf*, int * *length*)

This function returns the pointer to the used part of a memory buffer.

**Parameters**

***pMemBuf*** A pointer to a memory buffer structure.

***length*** The pointer to the length of the used part of the memory buffer.

**Returns**

The pointer to the used part of the memory buffer.

#### 2.17.3.5 OSOCTET* rtxMemBufGetDataExt (const OSRTMEMBUF * *pMemBuf*, OSSIZE * *length*)

This function returns the pointer to the used part of a memory buffer. The extended version returns length in a size-typed argument which is a 64-bit value on many systems.

**Parameters**

***pMemBuf*** A pointer to a memory buffer structure.

***length*** The pointer to the length of the used part of the memory buffer.

**Returns**

The pointer to the used part of the memory buffer.

#### 2.17.3.6 OSSIZE rtxMemBufGetDataLen (const OSRTMEMBUF * *pMemBuf*)

This function returns the length of the used part of a memory buffer.

**Parameters**

***pMemBuf*** A pointer to a memory buffer structure.

**Returns**

The length of the used part of the buffer.

#### 2.17.3.7 void rtxMemBufInit (OSCTXT * *pCtxt*, OSRTMEMBUF * *pMemBuf*, OSSIZE *segsize*)

This function initializes a memory buffer structure. It does not allocate memory; it sets the fields of the structure to the proper states. This function must be called before any operations with the memory buffer.

**Parameters**

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

***pMemBuf*** A pointer to the initialized memory buffer structure.

***segsize*** The number of bytes in which the memory buffer will be expanded incase it is full.

#### 2.17.3.8 void rtxMemBufInitBuffer (OSCTXT * *pCtxt*, OSRTMEMBUF * *pMemBuf*, OSOCTET * *buf*, OSSIZE *bufsize*, OSSIZE *segsize*)

This function assigns a static buffer to the memory buffer structure. It does not allocate memory; it sets the pointer to the passed buffer. If additional memory is required (for example, additional data is appended to the buffer using rtxMemBufAppend), a dynamic buffer will be allocated and all data copied to the new buffer.

**Parameters**

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

***pMemBuf*** A pointer to a memory buffer structure.

***buf*** A pointer to the buffer to be assigned.

***bufsize*** The size of the buffer.

***segsize*** The number of bytes on which the memory buffer will be expanded in case it is full.

#### 2.17.3.9 int rtxMemBufPreAllocate (OSRTMEMBUF * *pMemBuf*, OSSIZE *nbytes*)

This function allocates a buffer with a predetermined amount of space.

**Parameters**

***pMemBuf*** A pointer to a memory buffer structure.

***nbytes*** The number of bytes to be copied from pData.

**Returns**

Completion status of operation:

- 0 = success,
- negative return value is error.

#### 2.17.3.10 void rtxMemBufReset (OSRTMEMBUF * *pMemBuf*)

This function resets the memory buffer structure. It does not free memory, just sets the pointer to the beginning and the used length to zero.

**Parameters**

***pMemBuf*** A pointer to a memory buffer structure.

#### 2.17.3.11 int rtxMemBufSet (OSRTMEMBUF * *pMemBuf*, OSOCTET *value*, OSSIZE *nbytes*)

This function sets part of a memory buffer to a specified octet value. The filling is started from the end of the memory buffer. If the buffer is dynamic and full, then the buffer will be reallocated. If it is static (a static buffer was assigned by a call to rtxMemBufInitBuffer) or it is empty (no memory previously was allocated) then a new buffer will be allocated.

**Parameters**

***pMemBuf*** A pointer to a memory buffer structure.

***value*** The pointer to the buffer to be appended. The data will be copied at the end of the memory buffer.

***nbytes*** The number of bytes to be copied from pData.

**Returns**

Completion status of operation:

- 0 = success,
- negative return value is error.

#### 2.17.3.12 OSBOOL rtxMemBufSetExpandable (OSRTMEMBUF * *pMemBuf*, OSBOOL *isExpandable*)

This function sets "isExpandable" flag for the memory buffer object. By default, this flag is set to TRUE, thus, memory buffer could be expanded, even if it was initialized by static buffer (see `rtMemBufInitBuffer`). If flag is cleared and buffer is full the rtMemBufAppend/rtMemBufPreAllocate functions will return error status.

**Parameters**

***pMemBuf*** A pointer to a memory buffer structure.

***isExpandable*** TRUE, if buffer should be expandable.

**Returns**

Previous state of "isExpandable" flag.

#### 2.17.3.13 OSBOOL rtxMemBufSetUseSysMem (OSRTMEMBUF * *pMemBuf*, OSBOOL *value*)

This function sets a flag to indicate that system memory management should be used instead of the custom memory manager. This should be used if the allocated buffer must be preserved after calls to rtxMemFree or rtxMemReset.

**Parameters**

***pMemBuf*** A pointer to a memory buffer structure.

***value*** Boolean indicating system memory management to be used.

**Returns**

Previous state of "useSysMem" flag.

#### 2.17.3.14 OSSIZE rtxMemBufTrimW (OSRTMEMBUF * *pMemBuf*)

This function trims white space of the memory buffer.

**Parameters**

***pMemBuf*** A pointer to a memory buffer structure.

**Returns**

Length of trimmed buffer.

## 2.18 Print Functions

### Functions

- int rtxByteToHexChar (OSOCTET byte, char *buf, OSSIZE bufsize)
- void rtxPrintBoolean (const char *name, OSBOOL value)
- void rtxPrintDate (const char *name, const OSNumDateTime *pvalue)
- void rtxPrintTime (const char *name, const OSNumDateTime *pvalue)
- void rtxPrintDateTime (const char *name, const OSNumDateTime *pvalue)
- void **rtxPrintGYear** (const char *name, const OSNumDateTime *pvalue)
- void **rtxPrintGYearMonth** (const char *name, const OSNumDateTime *pvalue)
- void **rtxPrintGMonth** (const char *name, const OSNumDateTime *pvalue)
- void **rtxPrintGMonthDay** (const char *name, const OSNumDateTime *pvalue)
- void **rtxPrintGDay** (const char *name, const OSNumDateTime *pvalue)
- void rtxPrintInteger (const char *name, OSINT32 value)
- void rtxPrintInt64 (const char *name, OSINT64 value)
- void rtxPrintUnsigned (const char *name, OSUINT32 value)
- void rtxPrintUInt64 (const char *name, OSUINT64 value)
- void rtxPrintHexStr (const char *name, OSSIZE numocts, const OSOCTET *data)
- void rtxPrintHexBinary (const char *name, OSSIZE numocts, const OSOCTET *data)
- void rtxPrintCharStr (const char *name, const char *cstring)
- void rtxPrintUTF8CharStr (const char *name, const OSUTF8CHAR *cstring)
- void rtxPrintUnicodeCharStr (const char *name, const OSUNICHAR *str, int nchars)
- void rtxPrintReal (const char *name, OSREAL value)
- void rtxPrintNull (const char *name)
- void rtxPrintNVP (const char *name, const OSUTF8NVP *value)
- int rtxPrintFile (const char *filename)
- void rtxPrintIndent (void)
- void rtxPrintIncrIndent (void)
- void rtxPrintDecrIndent (void)
- void rtxPrintCloseBrace (void)
- void rtxPrintOpenBrace (const char *)
- void rtxHexDumpToNamedFile (const char *filename, const OSOCTET *data, OSSIZE numocts)
- void rtxHexDumpToFile (FILE *fp, const OSOCTET *data, OSSIZE numocts)
- void rtxHexDumpToFileEx (FILE *fp, const OSOCTET *data, OSSIZE numocts, int bytesPerUnit)
- void rtxHexDump (const OSOCTET *data, OSSIZE numocts)
- void rtxHexDumpEx (const OSOCTET *data, OSSIZE numocts, int bytesPerUnit)
- int rtxHexDumpToString (const OSOCTET *data, OSSIZE numocts, char *buffer, OSSIZE bufferIndex, OSSIZE bufferSize)
- int rtxHexDumpToStringEx (const OSOCTET *data, OSSIZE numocts, char *buffer, OSSIZE bufferIndex, OSSIZE bufferSize, int bytesPerUnit)
- void rtxHexDumpFileContents (const char *inFilePath)
- void rtxHexDumpFileContentsToFile (const char *inFilePath, const char *outFilePath)

### 2.18.1 Detailed Description

These functions simply print the output in a "name=value" format. The value format is obtained by calling one of the ToString functions with the given value.

## 2.18.2 Function Documentation

### 2.18.2.1 int rtxByteToHexChar (OSOCTET *byte*, char * *buf*, OSSIZE *bufsize*)

This function converts a byte value into its hex string equivalent.

**Parameters**

***byte*** Byte to format.

***buf*** Output buffer.

***bufsize*** Output buffer size.

### 2.18.2.2 void rtxHexDump (const OSOCTET * *data*, OSSIZE *numocts*)

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

**Parameters**

***data*** The pointer to a buffer to be displayed.

***numocts*** The number of octets to be displayed.

### 2.18.2.3 void rtxHexDumpEx (const OSOCTET * *data*, OSSIZE *numocts*, int *bytesPerUnit*)

This function outputs a hexadecimal dump of the current buffer contents to stdout, but it may display the dump as an array or bytes, words, or double words.

**Parameters**

***data*** The pointer to a buffer to be displayed.

***numocts*** The number of octets to be displayed.

***bytesPerUnit*** The number of bytes in one unit. May be 1 (byte), 2 (word), or 4 (double word).

### 2.18.2.4 void rtxHexDumpFileContents (const char * *inFilePath*)

This function outputs a hexadecimal dump of the contents of the named file to stdout.

**Parameters**

***inFilePath*** Name of file to be dumped.

### 2.18.2.5 void rtxHexDumpFileContentsToFile (const char * *inFilePath*, const char * *outFilePath*)

This function outputs a hexadecimal dump of the contents of the named file to a text file.

**Parameters**

***inFilePath*** Name of file to be dumped.

***outFilePath*** Name of file to which dump contents will be written.

#### 2.18.2.6 void rtxHexDumpToFile (FILE * *fp*, const OSOCTET * *data*, OSSIZE *numocts*)

This function outputs a hexadecimal dump of the current buffer contents to a file.

**Parameters**

***fp*** A pointer to FILE structure. The file should be opened for writing.

***data*** The pointer to a buffer to be displayed.

***numocts*** The number of octets to be displayed

#### 2.18.2.7 void rtxHexDumpToFileEx (FILE * *fp*, const OSOCTET * *data*, OSSIZE *numocts*, int *bytesPerUnit*)

This function outputs a hexadecimal dump of the current buffer to a file, but it may output the dump as an array of bytes, words, or double words.

**Parameters**

***fp*** A pointer to FILE structure. The file should be opened for writing.

***data*** The pointer to a buffer to be displayed.

***numocts*** The number of octets to be displayed.

***bytesPerUnit*** The number of bytes in one unit. May be 1 (byte), 2 (word), or 4 (double word).

#### 2.18.2.8 void rtxHexDumpToNamedFile (const char * *filename*, const OSOCTET * *data*, OSSIZE *numocts*)

This function outputs a hexadecimal dump of the current buffer contents to the file with the given name. The file is opened or created and then closed after the writer operation is complete.

**Parameters**

***filename*** Full path to file to which data should be output.

***data*** The pointer to a buffer to be displayed.

***numocts*** The number of octets to be displayed

#### 2.18.2.9 int rtxHexDumpToString (const OSOCTET * *data*, OSSIZE *numocts*, char * *buffer*, OSSIZE *bufferIndex*, OSSIZE *bufferSize*)

This function formats a hexadecimal dump of the current buffer contents to a string.

**Parameters**

***data*** The pointer to a buffer to be displayed.

***numocts*** The number of octets to be displayed.

***buffer*** The destination string buffer.

***bufferIndex*** The starting position in the destination buffer. The formatting of the dump will begin at this position.

***bufferSize*** The total size of the destination buffer.

**Returns**

The length of the final string.

#### 2.18.2.10 int rtxHexDumpToStringEx (const OSOCTET * *data*, OSSIZE *numocts*, char * *buffer*, OSSIZE *bufferIndex*, OSSIZE *bufferSize*, int *bytesPerUnit*)

This function formats a hexadecimal dump of the current buffer contents to a string, but it may output the dump as an array of bytes, words, or double words.

**Parameters**

***data*** The pointer to a buffer to be displayed.

***numocts*** The number of octets to be displayed.

***buffer*** The destination string buffer.

***bufferIndex*** The starting position in the destination buffer. The formatting of the dump will begin at this position.

***bufferSize*** The total size of the destination buffer.

***bytesPerUnit*** The number of bytes in one unit. May be 1 (byte), 2 (word), or 4 (double word).

**Returns**

The length of the final string.

#### 2.18.2.11 void rtxPrintBoolean (const char * *name*, OSBOOL *value*)

Prints a boolean value to stdout.

**Parameters**

***name*** The name of the variable to print.

***value*** Boolean value to print.

#### 2.18.2.12 void rtxPrintCharStr (const char * *name*, const char * *cstring*)

Prints an ASCII character string value to stdout.

**Parameters**

***name*** The name of the variable to print.

***cstring*** A pointer to the character string to be printed.

#### 2.18.2.13 void rtxPrintCloseBrace (void)

This function closes a braced region by decreasing the indent level, printing indent spaces, and printing the closing brace.

#### 2.18.2.14 void rtxPrintDate (const char * *name*, const OSNumDateTime * *pvalue*)

Prints a date value to stdout.

**Parameters**

***name*** Name of the variable to print.

***pvalue*** Pointer to a structure that holds numeric DateTime value to print.

#### 2.18.2.15 void rtxPrintDateTime (const char * *name*, const OSNumDateTime * *pvalue*)

Prints a dateTime value to stdout.

**Parameters**

***name*** Name of the variable to print.

***pvalue*** Pointer to a structure that holds numeric DateTime value to print.

#### 2.18.2.16 void rtxPrintDecrIndent (void)

This function decrements the current indentation level.

#### 2.18.2.17 int rtxPrintFile (const char * *filename*)

This function prints the contents of a text file to stdout.

**Parameters**

***filename*** The name of the text file to print.

**Returns**

Status of operation, 0 if success.

#### 2.18.2.18 void rtxPrintHexBinary (const char * *name*, OSSIZE *numocts*, const OSOCTET * *data*)

Prints an octet string value in hex binary format to stdout.

**Parameters**

***name*** The name of the variable to print.

***numocts*** The number of octets to be printed.

***data*** A pointer to the data to be printed.

#### 2.18.2.19 void rtxPrintHexStr (const char * *name*, OSSIZE *numocts*, const OSOCTET * *data*)

This function prints the value of a binary string in hex format to standard output. If the string is 32 bytes or less, it is printed on a single line with a '0x' prefix. If longer, a formatted hex dump showing both hex and ascii codes is done.

**Parameters**

***name*** The name of the variable to print.

***numocts*** The number of octets to be printed.

***data*** A pointer to the data to be printed.

#### 2.18.2.20 void rtxPrintIncrIndent (void)

This function increments the current indentation level.

#### 2.18.2.21 void rtxPrintIndent (void)

This function prints indentation spaces to stdout.

#### 2.18.2.22 void rtxPrintInt64 (const char * *name*, OSINT64 *value*)

Prints a 64-bit integer value to stdout.

**Parameters**

***name*** The name of the variable to print.

***value*** 64-bit integer value to print.

#### 2.18.2.23 void rtxPrintInteger (const char * *name*, OSINT32 *value*)

Prints an integer value to stdout.

**Parameters**

***name*** The name of the variable to print.

***value*** Integer value to print.

#### 2.18.2.24 void rtxPrintNull (const char * *name*)

Prints a NULL value to stdout.

**Parameters**

***name*** The name of the variable to print.

#### 2.18.2.25 void rtxPrintNVP (const char * *name*, const OSUTF8NVP * *value*)

Prints a name-value pair to stdout.

**Parameters**

***name*** The name of the variable to print.

***value*** A pointer to name-value pair structure to print.

#### 2.18.2.26 void rtxPrintOpenBrace (const char *)

This function opens a braced region by printing indent spaces, printing the name and opening brace, and increasing the indent level.

#### 2.18.2.27 void rtxPrintReal (const char * *name*, OSREAL *value*)

Prints a REAL (float, double, decimal) value to stdout.

**Parameters**

***name*** The name of the variable to print.

***value*** REAL value to print.

#### 2.18.2.28 void rtxPrintTime (const char * *name*, const OSNumDateTime * *pvalue*)

Prints a time value to stdout.

**Parameters**

***name*** Name of the variable to print.

***pvalue*** Pointer to a structure that holds numeric DateTime value to print.

#### 2.18.2.29 void rtxPrintUInt64 (const char * *name*, OSUINT64 *value*)

Prints an unsigned 64-bit integer value to stdout.

**Parameters**

***name*** The name of the variable to print.

***value*** Unsigned 64-bit integer value to print.

#### 2.18.2.30 void rtxPrintUnicodeCharStr (const char * *name*, const OSUNICHAR * *str*, int *nchars*)

This function prints a Unicode string to standard output. Characters in the string that are within the normal Ascii range are printed as single characters. Characters outside the Ascii range are printed as 4-byte hex codes (0xnnnn).

**Parameters**

***name*** The name of the variable to print.

***str*** Pointer to unicode sring to be printed. String is an array of C unsigned short data variables.

***nchars*** Number of characters in the string. If value is negative, string is assumed to be null-terminated (i.e. ends with a 0x0000 character).

#### 2.18.2.31 void rtxPrintUnsigned (const char * *name*, OSUINT32 *value*)

Prints an unsigned integer value to stdout.

**Parameters**

***name*** The name of the variable to print.

***value*** Unsigned integer value to print.

#### 2.18.2.32 void rtxPrintUTF8CharStr (const char * *name*, const OSUTF8CHAR * *cstring*)

Prints a UTF-8 encoded character string value to stdout.

**Parameters**

***name*** The name of the variable to print.

***cstring*** A pointer to the character string to be printed.

## 2.19 Print-To-Stream Functions

### Functions

- void rtxPrintToStreamBoolean (OSCTXT *pctxt, const char *name, OSBOOL value)
- void rtxPrintToStreamDate (OSCTXT *pctxt, const char *name, const OSNumDateTime *pvalue)
- void rtxPrintToStreamTime (OSCTXT *pctxt, const char *name, const OSNumDateTime *pvalue)
- void rtxPrintToStreamDateTime (OSCTXT *pctxt, const char *name, const OSNumDateTime *pvalue)
- void **rtxPrintToStreamGYear** (OSCTXT *pctxt, const char *name, const OSNumDateTime *pvalue)
- void **rtxPrintToStreamGYearMonth** (OSCTXT *pctxt, const char *name, const OSNumDateTime *pvalue)
- void **rtxPrintToStreamGMonth** (OSCTXT *pctxt, const char *name, const OSNumDateTime *pvalue)
- void **rtxPrintToStreamGMonthDay** (OSCTXT *pctxt, const char *name, const OSNumDateTime *pvalue)
- void **rtxPrintToStreamGDay** (OSCTXT *pctxt, const char *name, const OSNumDateTime *pvalue)
- void rtxPrintToStreamInteger (OSCTXT *pctxt, const char *name, OSINT32 value)
- void rtxPrintToStreamInt64 (OSCTXT *pctxt, const char *name, OSINT64 value)
- void rtxPrintToStreamUnsigned (OSCTXT *pctxt, const char *name, OSUINT32 value)
- void rtxPrintToStreamUInt64 (OSCTXT *pctxt, const char *name, OSUINT64 value)
- void rtxPrintToStreamHexStr (OSCTXT *pctxt, const char *name, size_t numocts, const OSOCTET *data)
- void rtxPrintToStreamHexBinary (OSCTXT *pctxt, const char *name, size_t numocts, const OSOCTET *data)
- void rtxPrintToStreamCharStr (OSCTXT *pctxt, const char *name, const char *cstring)
- void rtxPrintToStreamUTF8CharStr (OSCTXT *pctxt, const char *name, const OSUTF8CHAR *cstring)
- void rtxPrintToStreamUnicodeCharStr (OSCTXT *pctxt, const char *name, const OSUNICHAR *str, int nchars)
- void rtxPrintToStreamReal (OSCTXT *pctxt, const char *name, OSREAL value)
- void rtxPrintToStreamNull (OSCTXT *pctxt, const char *name)
- void rtxPrintToStreamNVP (OSCTXT *pctxt, const char *name, const OSUTF8NVP *value)
- int rtxPrintToStreamFile (OSCTXT *pctxt, const char *filename)
- void rtxPrintToStreamIndent (OSCTXT *pctxt)
- void rtxPrintToStreamIncrIndent (OSCTXT *pctxt)
- void rtxPrintToStreamDecrIndent (OSCTXT *pctxt)
- void rtxPrintToStreamCloseBrace (OSCTXT *pctxt)
- void rtxPrintToStreamOpenBrace (OSCTXT *pctxt, const char *)
- void rtxHexDumpToStream (OSCTXT *pctxt, const OSOCTET *data, size_t numocts)
- void rtxHexDumpToStreamEx (OSCTXT *pctxt, const OSOCTET *data, size_t numocts, int bytesPerUnit)

### 2.19.1 Detailed Description

These functions print typed data in a "name=value" format. The output is redirected to the print stream defined within the context or to a global print stream. Print streams are set using the rtxSetPrintStream or rtxSetGlobalPrintStream function.

### 2.19.2 Function Documentation

#### 2.19.2.1 void rtxHexDumpToStream (OSCTXT * *pctxt*, const OSOCTET * *data*, size_t *numocts*)

This function outputs a hexadecimal dump of the current buffer contents to a print stream.

**Parameters**

***pctxt*** A pointer to a context structure.

***data*** The pointer to a buffer to be displayed.

***numocts*** The number of octets to be displayed

#### 2.19.2.2 void rtxHexDumpToStreamEx (OSCTXT * *pctxt*, const OSOCTET * *data*, size_t *numocts*, int *bytesPerUnit*)

This function outputs a hexadecimal dump of the current buffer to a print stream, but it may output the dump as an array of bytes, words, or double words.

**Parameters**

***pctxt*** A pointer to a context structure.

***data*** The pointer to a buffer to be displayed.

***numocts*** The number of octets to be displayed.

***bytesPerUnit*** The number of bytes in one unit. May be 1 (byte), 2 (word), or 4 (double word).

#### 2.19.2.3 void rtxPrintToStreamBoolean (OSCTXT * *pctxt*, const char * *name*, OSBOOL *value*)

Prints a boolean value to a print stream.

**Parameters**

***pctxt*** A pointer to a context structure.

***name*** The name of the variable to print.

***value*** Boolean value to print.

#### 2.19.2.4 void rtxPrintToStreamCharStr (OSCTXT * *pctxt*, const char * *name*, const char * *cstring*)

Prints an ASCII character string value to a print stream.

**Parameters**

***pctxt*** A pointer to a context structure.

***name*** The name of the variable to print.

***cstring*** A pointer to the character string to be printed.

#### 2.19.2.5 void rtxPrintToStreamCloseBrace (OSCTXT * *pctxt*)

This function closes a braced region by decreasing the indent level, printing indent spaces, and printing the closing brace.

#### 2.19.2.6 void rtxPrintToStreamDate (OSCTXT * *pctxt*, const char * *name*, const OSNumDateTime * *pvalue*)

Prints a date value to a print stream.

**Parameters**

***pctxt*** A pointer to a context structure.

***name*** Name of the variable to print.

***pvalue*** Pointer to a structure that holds numeric DateTime value to print.

#### 2.19.2.7 void rtxPrintToStreamDateTime (OSCTXT * *pctxt*, const char * *name*, const OSNumDateTime * *pvalue*)

Prints a dateTime value to a print stream.

**Parameters**

***pctxt*** A pointer to a context structure.

***name*** Name of the variable to print.

***pvalue*** Pointer to a structure that holds numeric DateTime value to print.

#### 2.19.2.8 void rtxPrintToStreamDecrIndent (OSCTXT * *pctxt*)

This function decrements the current indentation level.

**Parameters**

***pctxt*** A pointer to a context data structure that holds the print stream.

#### 2.19.2.9 int rtxPrintToStreamFile (OSCTXT * *pctxt*, const char * *filename*)

This function prints the contents of a text file to a print stream.

**Parameters**

***pctxt*** A pointer to a context structure.

***filename*** The name of the text file to print.

**Returns**

Status of operation, 0 if success.

#### 2.19.2.10 void rtxPrintToStreamHexBinary (OSCTXT * *pctxt*, const char * *name*, size_t *numocts*, const OSOCTET * *data*)

Prints an octet string value in hex binary format to a print stream.

**Parameters**

***pctxt*** A pointer to a context structure.

***name*** The name of the variable to print.

***numocts*** The number of octets to be printed.

***data*** A pointer to the data to be printed.

#### 2.19.2.11 void rtxPrintToStreamHexStr (OSCTXT * *pctxt*, const char * *name*, size_t *numocts*, const OSOCTET * *data*)

This function prints the value of a binary string in hex format to standard output. If the string is 32 bytes or less, it is printed on a single line with a '0x' prefix. If longer, a formatted hex dump showing both hex and ascii codes is done.

**Parameters**

***pctxt*** A pointer to a context structure.

***name*** The name of the variable to print.

***numocts*** The number of octets to be printed.

***data*** A pointer to the data to be printed.

#### 2.19.2.12 void rtxPrintToStreamIncrIndent (OSCTXT * *pctxt*)

This function increments the current indentation level.

**Parameters**

***pctxt*** A pointer to a context data structure that holds the print stream.

#### 2.19.2.13 void rtxPrintToStreamIndent (OSCTXT * *pctxt*)

This function prints indentation spaces to a print stream.

#### 2.19.2.14 void rtxPrintToStreamInt64 (OSCTXT * *pctxt*, const char * *name*, OSINT64 *value*)

Prints a 64-bit integer value to a print stream.

**Parameters**

***pctxt*** A pointer to a context structure.

***name*** The name of the variable to print.

***value*** 64-bit integer value to print.

#### 2.19.2.15 void rtxPrintToStreamInteger (OSCTXT * *pctxt*, const char * *name*, OSINT32 *value*)

Prints an integer value to a print stream.

**Parameters**

***pctxt*** A pointer to a context structure.

***name*** The name of the variable to print.

***value*** Integer value to print.

#### 2.19.2.16 void rtxPrintToStreamNull (OSCTXT * *pctxt*, const char * *name*)

Prints a NULL value to a print stream.

**Parameters**

***pctxt*** A pointer to a context structure.

***name*** The name of the variable to print.

#### 2.19.2.17 void rtxPrintToStreamNVP (OSCTXT * *pctxt*, const char * *name*, const OSUTF8NVP * *value*)

Prints a name-value pair to a print stream.

**Parameters**

***pctxt*** A pointer to a context structure.

***name*** The name of the variable to print.

***value*** A pointer to name-value pair structure to print.

#### 2.19.2.18 void rtxPrintToStreamOpenBrace (OSCTXT * *pctxt*, const char *)

This function opens a braced region by printing indent spaces, printing the name and opening brace, and increasing the indent level.

#### 2.19.2.19 void rtxPrintToStreamReal (OSCTXT * *pctxt*, const char * *name*, OSREAL *value*)

Prints a REAL (float, double, decimal) value to a print stream.

**Parameters**

***pctxt*** A pointer to a context structure.

***name*** The name of the variable to print.

***value*** REAL value to print.

#### 2.19.2.20 void rtxPrintToStreamTime (OSCTXT * *pctxt*, const char * *name*, const OSNumDateTime * *pvalue*)

Prints a time value to a print stream.

**Parameters**

***pctxt*** A pointer to a context structure.

***name*** Name of the variable to print.

***pvalue*** Pointer to a structure that holds numeric DateTime value to print.

#### 2.19.2.21 void rtxPrintToStreamUInt64 (OSCTXT * *pctxt*, const char * *name*, OSUINT64 *value*)

Prints an unsigned 64-bit integer value to a print stream.

**Parameters**

***pctxt*** A pointer to a context structure.

***name*** The name of the variable to print.

***value*** Unsigned 64-bit integer value to print.

#### 2.19.2.22 void rtxPrintToStreamUnicodeCharStr (OSCTXT * *pctxt*, const char * *name*, const OSUNICHAR * *str*, int *nchars*)

This function prints a Unicode string to standard output. Characters in the string that are within the normal Ascii range are printed as single characters. Characters outside the Ascii range are printed as 4-byte hex codes (0xnnnn).

**Parameters**

***pctxt*** A pointer to a context structure.

***name*** The name of the variable to print.

***str*** Pointer to unicode sring to be printed. String is an array of C unsigned short data variables.

***nchars*** Number of characters in the string. If value is negative, string is assumed to be null-terminated (i.e. ends with a 0x0000 character).

#### 2.19.2.23 void rtxPrintToStreamUnsigned (OSCTXT * *pctxt*, const char * *name*, OSUINT32 *value*)

Prints an unsigned integer value to a print stream.

**Parameters**

***pctxt*** A pointer to a context structure.

***name*** The name of the variable to print.

***value*** Unsigned integer value to print.

#### 2.19.2.24 void rtxPrintToStreamUTF8CharStr (OSCTXT * *pctxt*, const char * *name*, const OSUTF8CHAR * *cstring*)

Prints a UTF-8 encoded character string value to a print stream.

**Parameters**

***pctxt*** A pointer to a context structure.

***name*** The name of the variable to print.

***cstring*** A pointer to the character string to be printed.

## 2.20 TCP/IP or UDP socket utility functions

### Defines

- #define **OSIPADDR_ANY** ((OSIPADDR)0)
- #define **OSIPADDR_LOCAL** ((OSIPADDR)0x7f000001UL)

### Typedefs

- typedef unsigned long OSIPADDR

### Functions

- int rtxSocketAccept (OSRTSOCKET socket, OSRTSOCKET *pNewSocket, OSIPADDR *destAddr, int *destPort)
- int rtxSocketAddrToStr (OSIPADDR ipAddr, char *pbuf, size_t bufsize)
- int rtxSocketBind (OSRTSOCKET socket, OSIPADDR addr, int port)
- int rtxSocketClose (OSRTSOCKET socket)
- int rtxSocketConnect (OSRTSOCKET socket, const char *host, int port)
- int rtxSocketConnectTimed (OSRTSOCKET socket, const char *host, int port, int nsecs)
- int rtxSocketCreate (OSRTSOCKET *psocket)
- int **rtxSocketCreateUDP** (OSRTSOCKET *psocket)
- int rtxSocketGetHost (const char *host, struct in_addr *inaddr)
- int rtxSocketsInit ()
- int rtxSocketListen (OSRTSOCKET socket, int maxConnection)
- int rtxSocketParseURL (char *url, char **protocol, char **address, int *port)
- int rtxSocketRecv (OSRTSOCKET socket, OSOCTET *pbuf, size_t bufsize)
- int rtxSocketRecvTimed (OSRTSOCKET socket, OSOCTET *pbuf, size_t bufsize, OSUINT32 secs)
- int rtxSocketSelect (int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout)
- int rtxSocketSend (OSRTSOCKET socket, const OSOCTET *pdata, size_t size)
- int rtxSocketSetBlocking (OSRTSOCKET socket, OSBOOL value)
- int rtxSocketStrToAddr (const char *pIPAddrStr, OSIPADDR *pIPAddr)

### 2.20.1 Typedef Documentation

#### 2.20.1.1 typedef unsigned long OSIPADDR

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.

### 2.20.2 Function Documentation

#### 2.20.2.1 int rtxSocketAccept (OSRTSOCKET *socket*, OSRTSOCKET * *pNewSocket*, OSIPADDR * *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 handle created by call to rtxSocketCreate function.

***pNewSocket*** The pointer to variable to receive the new socket 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 (0) = success, negative return value is error.

#### 2.20.2.2 int rtxSocketAddrToStr (OSIPADDR *ipAddr*, char * *pbuf*, size_t *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 (0) = success, negative return value is error.

#### 2.20.2.3 int rtxSocketBind (OSRTSOCKET *socket*, OSIPADDR *addr*, int *port*)

This function associates a local address with a socket. It is used on an unconnected socket before subsequent calls to the rtxSocketConnect or rtxSocketListen functions. See description of 'bind' socket function for further details.

**Parameters**

***socket*** The socket handle created by call to rtxSocketCreate 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 (0) = success, negative return value is error.

#### 2.20.2.4 int rtxSocketClose (OSRTSOCKET *socket*)

This function closes an existing socket.

**Parameters**

***socket*** The socket handle created by call to rtxSocketCreate or rtxSocketAccept function.

**Returns**

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

#### 2.20.2.5 int rtxSocketConnect (OSRTSOCKET *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 handle created by call to rtxSocketCreate 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 (0) = success, negative return value is error.

#### 2.20.2.6 int rtxSocketConnectTimed (OSRTSOCKET *socket*, const char * *host*, int *port*, int *nsecs*)

This function establishes a connection to a specified socket. It is similar to the rtxSocketConnect function except that it will only wait the given number of seconds to establish a connection before giving up.

**Parameters**

***socket*** The socket handle created by call to rtxSocketCreate 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.

***nsecs*** Number of seconds to wait before failing.

**Returns**

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

#### 2.20.2.7 int rtxSocketCreate (OSRTSOCKET * *psocket*)

This function creates a TCP socket.

**Parameters**

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

**Returns**

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

#### 2.20.2.8 int rtxSocketGetHost (const char * *host*, struct in_addr * *inaddr*)

This function resolves the given host name to an IP address. The resulting address is stored in the given socket address structure.

**Parameters**

***host*** Host name to resolve

***inaddr*** Socket address structure to receive resolved IP address

**Returns**

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

#### 2.20.2.9 int rtxSocketListen (OSRTSOCKET *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 rtxSocketCreate function and bound to a local address with the rtxSocketBind function, a maxConnection for incoming connections is specified with rtxSocketListen, and then the connections are accepted with the rtxSocketAccept function. See description of 'listen' socket function for further details.

**Parameters**

***socket*** The socket handle created by call to rtxSocketCreate function.

***maxConnection*** Maximum length of the queue of pending connections.

**Returns**

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

#### 2.20.2.10 int rtxSocketParseURL (char * *url*, char ** *protocol*, char ** *address*, int * *port*)

This function parses a simple URL of the form <protocol>://<address>:<port> into its individual components. It is assumed that the buffer the URL is provided in is modifiable. Null-terminators are inserted in the buffer to delimit the individual components. If the user needs to use the URL in unparsed form for any other purpose, they will need to make a copy of it before calling this function.

**Parameters**

***url*** URL to be parsed. Buffer will be altered.

***protocol*** Protocol string parsed from the URL.

***address*** IP address or domain name parsed from URL.

***port*** Optional port number. Zero if no port provided.

**Returns**

Zero if parse successful or negative error code.

#### 2.20.2.11 int rtxSocketRecv (OSRTSOCKET *socket*, OSOCTET * *pbuf*, size_t *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 handle created by call to rtxSocketCreate or rtxSocketAccept 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.

#### 2.20.2.12 int rtxSocketRecvTimed (OSRTSOCKET *socket*, OSOCTET * *pbuf*, size_t *bufsize*, OSUINT32 *secs*)

This function receives data from a connected socket on a timed basis. It is used to read incoming data on sockets. The socket must be connected before calling this function. If no data is available within the given timeout period, an error is returned. See description of 'recv' socket function for further details.

**Parameters**

***socket*** The socket handle created by call to rtxSocketCreate or rtxSocketAccept function.

***pbuf*** Pointer to the buffer for the incoming data.

***bufsize*** Length of the buffer. secs Amount of time to wait, in seconds, for data to be received.

**Returns**

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

#### 2.20.2.13 int rtxSocketSelect (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 documentation of the "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 = success, negative return value is error.

#### 2.20.2.14 int rtxSocketSend (OSRTSOCKET *socket*, const OSOCTET * *pdata*, size_t *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 handle created by call to rtxSocketCreate or rtxSocketAccept function.

***pdata*** Buffer containing the data to be transmitted.

***size*** Length of the data in pdata.

**Returns**

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

#### 2.20.2.15 int rtxSocketSetBlocking (OSRTSOCKET *socket*, OSBOOL *value*)

This function turns blocking mode for a socket on or off.

**Parameters**

***socket*** The socket handle created by call to rtxSocketCreate or rtxSocketAccept function.

***value*** Boolean value. True = turn blocking mode on.

**Returns**

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

#### 2.20.2.16 int rtxSocketsInit ()

This function initiates use of sockets by an application. This function must be called first before use sockets.

**Returns**

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

#### 2.20.2.17 int rtxSocketStrToAddr (const char * *pIPAddrStr*, OSIPADDR * *pIPAddr*)

This function converts the string with IP address to a double word representation. The converted address may be used with the rtxSocketBind 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 (0) = success, negative return value is error.

## 2.21 Input/Output Data Stream Utility Functions

### Classes

- struct OSRTSTREAM

### Defines

- #define **OSRTSTRMF_INPUT** 0x0001
- #define **OSRTSTRMF_OUTPUT** 0x0002
- #define **OSRTSTRMF_BUFFERED** 0x8000
- #define **OSRTSTRMF_UNBUFFERED** 0x4000
- #define **OSRTSTRMF_POSMARKED** 0x2000
- #define **OSRTSTRMF_FIXINMEM** 0x1000
- #define **OSRTSTRMF_BUF_INPUT** (OSRTSTRMF_INPUT|OSRTSTRMF_BUFFERED)
- #define **OSRTSTRMF_BUF_OUTPUT** (OSRTSTRMF_OUTPUT|OSRTSTRMF_BUFFERED)
- #define **OSRTSTRMID_FILE** 1
- #define **OSRTSTRMID_SOCKET** 2
- #define **OSRTSTRMID_MEMORY** 3
- #define **OSRTSTRMID_BUFFERED** 4
- #define **OSRTSTRMID_DIRECTBUF** 5
- #define **OSRTSTRMID_CTXTBUF** 6
- #define **OSRTSTRMID_ZLIB** 7
- #define **OSRTSTRMID_USER** 1000
- #define **OSRTSTRM_K_BUFSIZE** 1024
- #define **OSRTSTRM_K_INVALIDMARK** ((size_t)-1)
- #define **OSRTSTREAM_BYTEINDEX**(pctxt)
- #define **OSRTSTREAM_ID**(pctxt) ((pctxt)->pStream->id)
- #define **OSRTSTREAM_FLAGS**(pctxt) ((pctxt)->pStream->flags)

### Typedefs

- typedef long(* OSRTStreamReadProc )(struct OSRTSTREAM *pStream, OSOCTET *pbuffer, size_t bufSize)
- typedef long(* OSRTStreamBlockingReadProc )(struct OSRTSTREAM *pStream, OSOCTET *pbuffer, size_t toReadBytes)
- typedef long(* OSRTStreamWriteProc )(struct OSRTSTREAM *pStream, const OSOCTET *data, size_t numocts)
- typedef int(* OSRTStreamFlushProc )(struct OSRTSTREAM *pStream)
- typedef int(* OSRTStreamCloseProc )(struct OSRTSTREAM *pStream)
- typedef int(* OSRTStreamSkipProc )(struct OSRTSTREAM *pStream, size_t skipBytes)
- typedef int(* OSRTStreamMarkProc )(struct OSRTSTREAM *pStream, size_t readAheadLimit)
- typedef int(* OSRTStreamResetProc )(struct OSRTSTREAM *pStream)
- typedef int(* OSRTStreamGetPosProc )(struct OSRTSTREAM *pStream, size_t *ppos)
- typedef int(* OSRTStreamSetPosProc )(struct OSRTSTREAM *pStream, size_t pos)
- typedef struct OSRTSTREAM OSRTSTREAM

## Functions

- int rtxStreamClose (OSCTXT *pctxt)
- int rtxStreamFlush (OSCTXT *pctxt)
- int rtxStreamInit (OSCTXT *pctxt)
- long rtxStreamRead (OSCTXT *pctxt, OSOCTET *pbuffer, size_t bufSize)
- long rtxStreamBlockingRead (OSCTXT *pctxt, OSOCTET *pbuffer, size_t readBytes)
- int rtxStreamSkip (OSCTXT *pctxt, size_t skipBytes)
- long rtxStreamWrite (OSCTXT *pctxt, const OSOCTET *data, size_t numocts)
- int rtxStreamGetIOBytes (OSCTXT *pctxt, size_t *pPos)
- int rtxStreamMark (OSCTXT *pctxt, size_t readAheadLimit)
- int rtxStreamReset (OSCTXT *pctxt)
- OSBOOL rtxStreamMarkSupported (OSCTXT *pctxt)
- OSBOOL rtxStreamIsOpened (OSCTXT *pctxt)
- OSBOOL rtxStreamIsReadable (OSCTXT *pctxt)
- OSBOOL rtxStreamIsWritable (OSCTXT *pctxt)
- int rtxStreamRelease (OSCTXT *pctxt)
- void rtxStreamSetCapture (OSCTXT *pctxt, OSRTMEMBUF *pmembuf)
- OSRTMEMBUF * rtxStreamGetCapture (OSCTXT *pctxt)
- int rtxStreamGetPos (OSCTXT *pctxt, size_t *ppos)
- int rtxStreamSetPos (OSCTXT *pctxt, size_t pos)

### 2.21.1 Detailed Description

Stream functions are used for unbuffered stream operations. All of the operations with streams are performed using a context block to maintain state information.

These functions may be used for any input/output operations with streams. Each stream should be initialized first by call to the `rtxStreamInit` function. After initialization, the stream may be opened for reading or writing by calling one of the following functions:

- `rtxStreamFileOpen`
- `rtxStreamFileAttach`
- `rtxStreamSocketAttach`
- `rtxStreamMemoryCreate`
- `rtxStreamMemoryAttach`

### 2.21.2 Define Documentation

#### 2.21.2.1 #define OSRTSTREAM_BYTEINDEX(pctxt)

**Value:**

```
(((pctxt)->pStream->id == OSRTSTRMID_DIRECTBUF) ?                    \
 ((pctxt)->pStream->bytesProcessed + (pctxt)->buffer.byteIndex) :    \
 ((pctxt)->pStream->bytesProcessed /* was ioBytes */ ))
```

### 2.21.3 Typedef Documentation

#### 2.21.3.1 typedef struct OSRTSTREAM OSRTSTREAM

The stream control block. A user may implement a customized stream by defining read, skip, close functions for input streams and write, flush, close for output streams.

#### 2.21.3.2 typedef long(* OSRTStreamBlockingReadProc)(struct OSRTSTREAM *pStream, OSOCTET *pbuffer, size_t toReadBytes)

Stream blockingRead function pointer type. A user may implement a customized read function for specific input streams. The blockingRead function is defined in the OSRTSTREAM control structure.

#### 2.21.3.3 typedef int(* OSRTStreamCloseProc)(struct OSRTSTREAM *pStream)

Stream close function pointer type. A user may implement a customized close function for any specific input or output streams. The close function is defined in the OSRTSTREAM control structure.

#### 2.21.3.4 typedef int(* OSRTStreamFlushProc)(struct OSRTSTREAM *pStream)

Stream flush function pointer type. A user may implement a customized flush function for any specific output streams. The flush function is defined in the OSRTSTREAM control structure.

#### 2.21.3.5 typedef int(* OSRTStreamGetPosProc)(struct OSRTSTREAM *pStream, size_t *ppos)

Stream get position function pointer type. A user may implement a customized function for a specific input stream type. The mark function is defined in the OSRTSTREAM control structure.

#### 2.21.3.6 typedef int(* OSRTStreamMarkProc)(struct OSRTSTREAM *pStream, size_t readAheadLimit)

Stream mark function pointer type. A user may implement a customized function for a specific input stream type. The mark function is defined in the OSRTSTREAM control structure.

#### 2.21.3.7 typedef long(* OSRTStreamReadProc)(struct OSRTSTREAM *pStream, OSOCTET *pbuffer, size_t bufSize)

Stream read function pointer type. A user may implement a customized read function for specific input streams. The read function is defined in the OSRTSTREAM control structure.

#### 2.21.3.8 typedef int(* OSRTStreamResetProc)(struct OSRTSTREAM *pStream)

Stream reset function pointer type. A user may implement a customized function for a specific input stream type. The reset function is defined in the OSRTSTREAM control structure.

#### 2.21.3.9 typedef int(* OSRTStreamSetPosProc)(struct OSRTSTREAM *pStream, size_t pos)

Stream set position function pointer type. A user may implement a customized function for a specific input stream type. The mark function is defined in the OSRTSTREAM control structure.

#### 2.21.3.10 typedef int(* OSRTStreamSkipProc)(struct OSRTSTREAM *pStream, size_t skipBytes)

Stream skip function pointer type. A user may implement a customized function for a specific input stream type. The skip function is defined in the OSRTSTREAM control structure.

#### 2.21.3.11 typedef long(* OSRTStreamWriteProc)(struct OSRTSTREAM *pStream, const OSOCTET *data, size_t numocts)

Stream write function pointer type. A user may implement a customized write function for any specific output streams. The write function is defined in the OSRTSTREAM control structure.

### 2.21.4 Function Documentation

#### 2.21.4.1 long rtxStreamBlockingRead (OSCTXT * *pctxt*, OSOCTET * *pbuffer*, size_t *readBytes*)

This function reads up to 'bufsize' bytes of data from the input stream into an array of octets. An attempt is made to read as many as bufsize octets, but a smaller number may be read, possibly zero. The number of octets actually read is returned as an integer. This functions blocks until input data is available, end of file is detected, or another error is occurred.

**Parameters**

***pctxt*** Pointer to a context structure variable which has been initialized for stream operations via a call to rtxStreamInit.

***pbuffer*** Pointer to a buffer to receive data.

***readBytes*** Number of bytes to read.

**Returns**

The total number of octets read into the buffer, or negative value with error code if any error is occurred.

#### 2.21.4.2 int rtxStreamClose (OSCTXT * *pctxt*)

This function closes the input or output stream and releases any system resources associated with the stream. For output streams this function also flushes all internal buffers to the stream.

**Parameters**

***pctxt*** Pointer to a context structure variable which has been initialized for stream operations via a call to `rtxStreamInit`.

#### 2.21.4.3 int rtxStreamFlush (OSCTXT * *pctxt*)

This function flushes the output stream and forces any buffered output octets to be written out.

**Parameters**

***pctxt*** Pointer to a context structure variable which has been initialized for stream operations via a call to `rtxStreamInit`.

**Returns**

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

#### 2.21.4.4 OSRTMEMBUF* rtxStreamGetCapture (OSCTXT * *pctxt*)

This function returns the capture buffer currently assigned to the stream.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

**Returns**

Pointer to memory buffer that was previously assigned as a capture buffer to the stream.

#### 2.21.4.5 int rtxStreamGetIOBytes (OSCTXT * *pctxt*, size_t * *pPos*)

This function returns the number of processed octets. If the stream was opened as an input stream, then it returns the total number of read octets. If the stream was opened as an output stream, then it returns the total number of written octets. Otherwise, this function returns an error code.

**Parameters**

***pctxt*** Pointer to a context structure variable which has been initialized for stream operations via a call to `rtxStreamInit`.

***pPos*** Pointer to argument to receive total number of processed octets.

**Returns**

The total number of processed octets or error code (negative value).

#### 2.21.4.6 int rtxStreamGetPos (OSCTXT * *pctxt*, size_t * *ppos*)

Get the current position in input stream.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

***ppos*** Pointer to a variable to receive position.

**Returns**

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

#### 2.21.4.7 int rtxStreamInit (OSCTXT * *pctxt*)

This function initializes a stream part of the context block. This function should be called first before any operation with a stream.

**Parameters**

***pctxt*** Pointer to context structure variable, for which stream to be initialized.

**Returns**

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

#### 2.21.4.8 OSBOOL rtxStreamIsOpened (OSCTXT * *pctxt*)

Tests if this stream opened (for reading or writing).

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

**Returns**

TRUE if this stream is opened for reading or writing; FALSE otherwise.

#### 2.21.4.9 OSBOOL rtxStreamIsReadable (OSCTXT * *pctxt*)

Tests if this stream opened for reading.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

**Returns**

TRUE if this stream is opened for reading; FALSE otherwise.

#### 2.21.4.10 OSBOOL rtxStreamIsWritable (OSCTXT * *pctxt*)

Tests if this stream opened for writing.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

**Returns**

TRUE if this stream is opened for writing; FALSE otherwise.

#### 2.21.4.11 int rtxStreamMark (OSCTXT * *pctxt*, size_t *readAheadLimit*)

Marks the current position in this input stream. A subsequent call to the rtxStreamReset function repositions this stream at the last marked position so that subsequent reads re-read the same bytes. The `readAheadLimit` argument tells this input stream to allow many bytes to be read before the mark position gets invalidated.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

***readAheadLimit*** The maximum limit of bytes that can be read before the mark position becomes invalid.

**Returns**

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

#### 2.21.4.12 OSBOOL rtxStreamMarkSupported (OSCTXT * *pctxt*)

Tests if this input stream supports the mark and reset methods. Whether or not mark and reset are supported is an invariant property of a particular input stream instance. By default, it returns FALSE.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

**Returns**

TRUE if this stream instance supports the mark and reset methods; FALSE otherwise.

#### 2.21.4.13 long rtxStreamRead (OSCTXT * *pctxt*, OSOCTET * *pbuffer*, size_t *bufSize*)

This function reads up to 'bufsize' bytes of data from the input stream into an array of octets. An attempt is made to read as many as bufsize octets, but a smaller number may be read, possibly zero. The number of octets actually read is returned as an integer. This functions blocks until input data is available, end of file is detected, or another error is occurred.

**Parameters**

***pctxt*** Pointer to a context structure variable which has been initialized for stream operations via a call to rtxStreamInit.

***pbuffer*** Pointer to a buffer to receive data.

***bufSize*** Size of the buffer.

**Returns**

The total number of octets read into the buffer, or negative value with error code if any error is occurred.

#### 2.21.4.14 int rtxStreamRelease (OSCTXT * *pctxt*)

This function releases the stream's resources. If it is opened for reading or writing it will be closed.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

**Returns**

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

#### 2.21.4.15 int rtxStreamReset (OSCTXT * *pctxt*)

Repositions this stream to the position recorded by the last call to the rtxStreamMark function.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

**Returns**

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

#### 2.21.4.16 void rtxStreamSetCapture (OSCTXT * *pctxt*, OSRTMEMBUF * *pmembuf*)

This function sets a capture buffer for the stream. This is used to record all data read from the stream.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

***pmembuf*** Pointer to an initialized memory buffer structure. This argument may be set to NULL to disable capture if previously set.

#### 2.21.4.17 int rtxStreamSetPos (OSCTXT * *pctxt*, size_t *pos*)

Set the current position in input stream.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

***pos*** Stream position.

**Returns**

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

#### 2.21.4.18 int rtxStreamSkip (OSCTXT * *pctxt*, size_t *skipBytes*)

This function skips over and discards the specified amount of data octets from this input stream.

**Parameters**

***pctxt*** Pointer to a context structure variable which has been initialized for stream operations via a call to rtxStreamInit.

***skipBytes*** The number of octets to be skipped.

**Returns**

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

#### 2.21.4.19 long rtxStreamWrite (OSCTXT * *pctxt*, const OSOCTET * *data*, size_t *numocts*)

This function writes the specified amount of octets from the specified array to the output stream.

**Parameters**

***pctxt*** Pointer to a context structure variable which has been initialized for stream operations via a call to rtxStreamInit.

***data*** The pointer to data to be written.

***numocts*** The number of octets to write.

**Returns**

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

## 2.22 File stream functions.

### Functions

- int rtxStreamFileAttach (OSCTXT *pctxt, FILE *pFile, OSUINT16 flags)
- int rtxStreamFileOpen (OSCTXT *pctxt, const char *pFilename, OSUINT16 flags)
- int rtxStreamFileCreateReader (OSCTXT *pctxt, const char *pFilename)
- int rtxStreamFileCreateWriter (OSCTXT *pctxt, const char *pFilename)

### 2.22.1 Detailed Description

File stream functions are used for stream operations with files.

### 2.22.2 Function Documentation

#### 2.22.2.1 int rtxStreamFileAttach (OSCTXT * *pctxt*, FILE * *pFile*, OSUINT16 *flags*)

Attaches the existing file structure pointer to the stream. The file should be already opened either for the reading or writing. The 'flags' parameter specifies the access mode for the stream - input or output.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

***pFile*** Pointer to FILE structure. File should be already opened either for the writing or reading.

***flags*** Specifies the access mode for the stream:

- OSRTSTRMF_INPUT = input (reading) stream;
- OSRTSTRMF_OUTPUT = output (writing) stream.

**Returns**

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

#### 2.22.2.2 int rtxStreamFileCreateReader (OSCTXT * *pctxt*, const char * *pFilename*)

This function creates an input file stream using the specified file name.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

***pFilename*** Pointer to null-terminated string that contains the name of file.

**Returns**

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

#### 2.22.2.3 int rtxStreamFileCreateWriter (OSCTXT * *pctxt*, const char * *pFilename*)

This function creates an output file stream using the file name.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

***pFilename*** Pointer to null-terminated string that contains the name of file.

**Returns**

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

#### 2.22.2.4 int rtxStreamFileOpen (OSCTXT * *pctxt*, const char * *pFilename*, OSUINT16 *flags*)

Opens a file stream. The 'flags' parameter specifies the access mode for the stream - input or output.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

***pFilename*** Pointer to null-terminated string that contains the name of file.

***flags*** Specifies the access mode for the stream:

- OSRTSTRMF_INPUT = input (reading) stream;
- OSRTSTRMF_OUTPUT = output (writing) stream.

**Returns**

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

## 2.23 Memory stream functions.

### Functions

- int rtxStreamMemoryCreate (OSCTXT *pctxt, OSUINT16 flags)
- int rtxStreamMemoryAttach (OSCTXT *pctxt, OSOCTET *pMemBuf, size_t bufSize, OSUINT16 flags)
- OSOCTET * rtxStreamMemoryGetBuffer (OSCTXT *pctxt, size_t *pSize)
- int rtxStreamMemoryCreateReader (OSCTXT *pctxt, OSOCTET *pMemBuf, size_t bufSize)
- int rtxStreamMemoryCreateWriter (OSCTXT *pctxt, OSOCTET *pMemBuf, size_t bufSize)
- int rtxStreamMemoryResetWriter (OSCTXT *pctxt)

### 2.23.1 Detailed Description

Memory stream functions are used for memory stream operations.

### 2.23.2 Function Documentation

#### 2.23.2.1 int rtxStreamMemoryAttach (OSCTXT * *pctxt*, OSOCTET * *pMemBuf*, size_t *bufSize*, OSUINT16 *flags*)

Opens a memory stream using the specified memory buffer. The 'flags' parameter specifies the access mode for the stream - input or output.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

***pMemBuf*** The pointer to the buffer.

***bufSize*** The size of the buffer.

***flags*** Specifies the access mode for the stream:

- OSRTSTRMF_INPUT = input (reading) stream;
- OSRTSTRMF_OUTPUT = output (writing) stream.

**Returns**

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

#### 2.23.2.2 int rtxStreamMemoryCreate (OSCTXT * *pctxt*, OSUINT16 *flags*)

Opens a memory stream. A memory buffer will be created by this function. The 'flags' parameter specifies the access mode for the stream - input or output.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

***flags*** Specifies the access mode for the stream:

- OSRTSTRMF_INPUT = input (reading) stream;
- OSRTSTRMF_OUTPUT = output (writing) stream.

**Returns**

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

#### 2.23.2.3 int rtxStreamMemoryCreateReader (OSCTXT * *pctxt*, OSOCTET * *pMemBuf*, size_t *bufSize*)

This function creates an input memory stream using the specified buffer.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

***pMemBuf*** The pointer to the buffer

***bufSize*** The size of the buffer

**Returns**

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

#### 2.23.2.4 int rtxStreamMemoryCreateWriter (OSCTXT * *pctxt*, OSOCTET * *pMemBuf*, size_t *bufSize*)

This function creates an output memory stream using the specified buffer. If `pMemBuf` or `bufSize` is NULL then new buffer will be allocated.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

***pMemBuf*** The pointer to the buffer. Can be NULL - new buffer will be allocated in this case.

***bufSize*** The size of the buffer. Can be 0 - new buffer will be allocated in this case.

**Returns**

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

#### 2.23.2.5 OSOCTET* rtxStreamMemoryGetBuffer (OSCTXT * *pctxt*, size_t * *pSize*)

This function returns the memory buffer and its size for the given memory stream.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

***pSize*** The pointer to size_t to receive the size of buffer.

**Returns**

The pointer to memory buffer. NULL, if error occurred.

#### 2.23.2.6 int rtxStreamMemoryResetWriter (OSCTXT * *pctxt*)

This function resets the output memory stream internal buffer to allow it to be overwritten with new data. Memory for the buffer is not freed.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

**Returns**

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

## 2.24 Socket stream functions.

### Functions

- int rtxStreamSocketAttach (OSCTXT *pctxt, OSRTSOCKET socket, OSUINT16 flags)
- int rtxStreamSocketClose (OSCTXT *pctxt)
- int rtxStreamSocketCreateWriter (OSCTXT *pctxt, const char *host, int port)
- int rtxStreamSocketSetOwnership (OSCTXT *pctxt, OSBOOL ownSocket)
- int rtxStreamSocketSetReadTimeout (OSCTXT *pctxt, OSUINT32 nsecs)

### 2.24.1 Detailed Description

Socket stream functions are used for socket stream operations.

### 2.24.2 Function Documentation

#### 2.24.2.1 int rtxStreamSocketAttach (OSCTXT * *pctxt*, OSRTSOCKET *socket*, OSUINT16 *flags*)

Attaches the existing socket handle to the stream. The socket should be already opened and connected. The 'flags' parameter specifies the access mode for the stream - input or output.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

***socket*** The socket handle created by `rtxSocketCreate`.

***flags*** Specifies the access mode for the stream:

- OSRTSTRMF_INPUT = input (reading) stream;
- OSRTSTRMF_OUTPUT = output (writing) stream.

**Returns**

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

#### 2.24.2.2 int rtxStreamSocketClose (OSCTXT * *pctxt*)

This function closes a socket stream.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

**Returns**

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

#### 2.24.2.3 int rtxStreamSocketCreateWriter (OSCTXT * *pctxt*, const char * *host*, int *port*)

This function opens a socket stream for writing.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

***host*** Name of host or IP address to which to connect.

***port*** Port number to which to connect.

**Returns**

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

#### 2.24.2.4 int rtxStreamSocketSetOwnership (OSCTXT * *pctxt*, OSBOOL *ownSocket*)

This function transfers ownership of the socket to or from the stream instance. The socket will be closed and deleted when the stream is closed or goes out of scope. By default stream socket owns the socket.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

***ownSocket*** Boolean value.

**Returns**

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

#### 2.24.2.5 int rtxStreamSocketSetReadTimeout (OSCTXT * *pctxt*, OSUINT32 *nsecs*)

This function sets the read timeout value to the given number of seconds. Any read operation attempted on the stream will timeout after this period of time if no data is received.

**Parameters**

***pctxt*** Pointer to a context structure variable that has been initialized for stream operations.

***nsecs*** Number of seconds to wait before timing out.

**Returns**

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

## 2.25 Doubly-Linked List Utility Functions

### Classes

- struct OSRTDListNode
- struct OSRTDList
- struct OSRTDListBuf
- struct OSRTDListUTF8StrNode

### Typedefs

- typedef struct OSRTDListNode **OSRTDListNode**
- typedef struct OSRTDList **OSRTDList**
- typedef struct OSRTDListBuf **OSRTDListBuf**
- typedef struct OSRTDListUTF8StrNode **OSRTDListUTF8StrNode**
- typedef int(∗ **PEqualsFunc** )(const void ∗a, const void ∗b, const void ∗sortCtxt)

### Functions

- void rtxDListInit (OSRTDList ∗pList)
- OSRTDListNode ∗ rtxDListAppend (struct OSCTXT ∗pctxt, OSRTDList ∗pList, void ∗pData)
- OSRTDListNode ∗ rtxDListAppendCharArray (struct OSCTXT ∗pctxt, OSRTDList ∗pList, size_t length, char ∗pData)
- OSRTDListNode ∗ rtxDListAppendNode (OSRTDList ∗pList, OSRTDListNode ∗pListNode)
- OSRTDListNode ∗ rtxDListInsert (struct OSCTXT ∗pctxt, OSRTDList ∗pList, OSSIZE idx, void ∗pData)
- OSRTDListNode ∗ **rtxDListInsertNode** (OSRTDList ∗pList, OSSIZE idx, OSRTDListNode ∗pListNode)
- OSRTDListNode ∗ rtxDListInsertBefore (struct OSCTXT ∗pctxt, OSRTDList ∗pList, OSRTDListNode ∗node, void ∗pData)
- OSRTDListNode ∗ rtxDListInsertAfter (struct OSCTXT ∗pctxt, OSRTDList ∗pList, OSRTDListNode ∗node, void ∗pData)
- OSRTDListNode ∗ rtxDListFindByIndex (const OSRTDList ∗pList, OSSIZE idx)
- OSRTDListNode ∗ rtxDListFindByData (const OSRTDList ∗pList, void ∗data)
- int rtxDListFindIndexByData (const OSRTDList ∗pList, void ∗data)
- void rtxDListFreeNode (struct OSCTXT ∗pctxt, OSRTDList ∗pList, OSRTDListNode ∗node)
- void rtxDListRemove (OSRTDList ∗pList, OSRTDListNode ∗node)
- void rtxDListFreeNodes (struct OSCTXT ∗pctxt, OSRTDList ∗pList)
- void rtxDListFreeAll (struct OSCTXT ∗pctxt, OSRTDList ∗pList)
- int rtxDListToArray (struct OSCTXT ∗pctxt, OSRTDList ∗pList, void ∗∗ppArray, OSSIZE ∗pElemCount, OSSIZE elemSize)
- int rtxDListAppendArray (struct OSCTXT ∗pctxt, OSRTDList ∗pList, void ∗pArray, OSSIZE numElements, OSSIZE elemSize)
- int rtxDListAppendArrayCopy (struct OSCTXT ∗pctxt, OSRTDList ∗pList, const void ∗pArray, OSSIZE numElements, OSSIZE elemSize)
- int rtxDListToUTF8Str (struct OSCTXT ∗pctxt, OSRTDList ∗pList, OSUTF8CHAR ∗∗ppstr, char sep)
- OSRTDListNode ∗ **rtxDListInsertSorted** (struct OSCTXT ∗pctxt, OSRTDList ∗pList, void ∗pData, PEqualsFunc equalsFunc, void ∗sortCtxt)
- OSRTDListNode ∗ **rtxDListInsertNodeSorted** (OSRTDList ∗pList, OSRTDListNode ∗pListNode, PEqualsFunc equalsFunc, void ∗sortCtxt)

### 2.25.1 Detailed Description

The doubly-linked list utility functions provide common routines for managing linked lists. These lists are used to model XSD list and repeating element types within the generated code. This list type contains forward and backward pointers allowing the list to be traversed in either direction.

### 2.25.2 Function Documentation

#### 2.25.2.1 OSRTDListNode* rtxDListAppend (struct OSCTXT * *pctxt*, OSRTDList * *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 an object of any type. The `rtxMemAlloc` function is used to allocate memory for the list node structure; therefore, all internal list memory will be released whenever `rtxMemFree` 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 will be appended.

***pData*** A pointer to the 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.

#### 2.25.2.2 int rtxDListAppendArray (struct OSCTXT * *pctxt*, OSRTDList * *pList*, void * *pArray*, OSSIZE *numElements*, OSSIZE *elemSize*)

This function appends pointers to items in the given array to a doubly linked list structure. The array is assumed to hold an array of values as opposed to pointers. The actual address of each item in the array is stored - a copy of each item is not made.

**Parameters**

***pctxt*** A pointer to a context structure.

***pList*** A pointer to the linked list structure onto which the array items will be appended.

***pArray*** A pointer to the source array to be converted.

***numElements*** The number of elements in the array.

***elemSize*** The size of one element in the array. Use the `sizeof()` operator to pass this parameter.

**Returns**

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

#### 2.25.2.3 int rtxDListAppendArrayCopy (struct OSCTXT * *pctxt*, OSRTDList * *pList*, const void * *pArray*, OSSIZE *numElements*, OSSIZE *elemSize*)

This function appends a copy of each item in the given array to a doubly linked list structure. In this case, the `rtxMemAlloc` function is used to allocate memory for each item and a copy is made.

**Parameters**

***pctxt*** A pointer to a context structure.

***pList*** A pointer to the linked list structure onto which the array items will be appended.

***pArray*** A pointer to the source array to be converted.

***numElements*** The number of elements in the array.

***elemSize*** The size of one element in the array. Use the `sizeof()` operator to pass this parameter.

**Returns**

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

#### 2.25.2.4 OSRTDListNode* rtxDListAppendCharArray (struct OSCTXT * *pctxt*, OSRTDList * *pList*, size_t *length*, char * *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 an object of any type. The `rtxMemAlloc` function is used to allocate memory for the list node structure; therefore, all internal list memory will be released whenever `rtxMemFree` is called. The array passed in is copied and a pointer to the copy is stored in the list.

**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 will be appended.

***length*** The size of the character array to be appended.

***pData*** A pointer to the character array.

**Returns**

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

#### 2.25.2.5 OSRTDListNode* rtxDListAppendNode (OSRTDList * *pList*, OSRTDListNode * *pListNode*)

This function appends an `OSRTDListNode` to the linked list structure. The node data is a void pointer that can point to an object of any type. The `rtxMemAlloc` function is used to allocate memory for the list node structure; therefore, all internal list memory will be released whenever `rtxMemFree` is called. The pointer to the data item itself is stored in the node structure - a copy is not made.

**Parameters**

***pList*** A pointer to a linked list structure onto which the data item will be appended.

***pListNode*** A pointer to the node to be appended to the list.

**Returns**

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

#### 2.25.2.6 OSRTDListNode* rtxDListFindByData (const OSRTDList * *pList*, void * *data*)

This function will return the node pointer of the given data item within the list or NULL if the item is not found.

**Parameters**

***pList*** A pointer to a linked list structure.

***data*** Pointer to the data item to search for. Note that comparison of pointer values is done; not the items pointed at by the pointers.

**Returns**

A pointer to an allocated linked list node structure.

#### 2.25.2.7 OSRTDListNode* rtxDListFindByIndex (const OSRTDList * *pList*, OSSIZE *idx*)

This function will return the node pointer of the indexed entry in the list.

**Parameters**

***pList*** A pointer to a linked list structure.

***idx*** Zero-based index into list where the specified item is located. If the list contains fewer items then the index, NULL is returned.

**Returns**

A pointer to an allocated linked list node structure. To get the actual data item, the `data` member variable pointer within this structure must be dereferenced.

#### 2.25.2.8 int rtxDListFindIndexByData (const OSRTDList * *pList*, void * *data*)

This function will return the index of the given data item within the list or -1 if the item is not found.

**Parameters**

***pList*** A pointer to a linked list structure.

***data*** Pointer to the data item to search for. Note that comparison of pointer values is done; not the items pointed at by the pointers.

**Returns**

Index of item within the list or -1 if not found.

#### 2.25.2.9 void rtxDListFreeAll (struct OSCTXT * *pctxt*, OSRTDList * *pList*)

This function will free all of the dynamic memory used to hold the list node pointers and the data items. In this case, it is assumed that the `rtxMemAlloc` function was used to allocate memory for the data items.

**Parameters**

***pctxt*** A pointer to a context structure.

***pList*** A pointer to a linked list structure.

#### 2.25.2.10 void rtxDListFreeNode (struct OSCTXT * *pctxt*, OSRTDList * *pList*, OSRTDListNode * *node*)

This function will remove the given node from the list and free memory. The data memory is not freed. It might be released when the `rtxMemFree` or `rtFreeContext` function is called with this context.

**Parameters**

***pctxt*** A pointer to a context structure.

***pList*** A pointer to a linked list structure.

***node*** Pointer to the list node to be removed.

#### 2.25.2.11 void rtxDListFreeNodes (struct OSCTXT * *pctxt*, OSRTDList * *pList*)

This function will free all of the dynamic memory used to hold the list node pointers. It does not free the data items because it is unknown how the memory was allocated for these items.

**Parameters**

***pctxt*** A pointer to a context structure.

***pList*** A pointer to a linked list structure.

#### 2.25.2.12 void rtxDListInit (OSRTDList * *pList*)

This function initializes a doubly linked list structure. It sets the number of elements to zero and sets all internal pointer values to NULL. A doubly linked-list structure is described by the `OSRTDList` type. Nodes of the list are of type `OSRTDListNode`.

Memory for the structures is allocated using the `rtxMemAlloc` run-time function and is maintained within the context structure that is a required parameter to all rtDList functions. This memory is released when `rtxMemFree` 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.

#### 2.25.2.13 OSRTDListNode* rtxDListInsert (struct OSCTXT * *pctxt*, OSRTDList * *pList*, OSSIZE *idx*, void * *pData*)

This function inserts an item into the linked list structure. The data item is passed into the function as a void pointer that can point to an object of any type. The `rtxMemAlloc` function is used to allocate memory for the list node structure; therefore, all internal list memory will be released when the `rtxMemFree` function is called.

**Parameters**

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

***pList*** A pointer to a linked list structure into which the data item is to be inserted.

***idx*** Zero-based index into list where the specified item is to be inserted.

***pData*** A pointer to the data item to be inserted to the list.

**Returns**

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

#### 2.25.2.14 OSRTDListNode* rtxDListInsertAfter (struct OSCTXT * *pctxt*, OSRTDList * *pList*, OSRTDListNode * *node*, void * *pData*)

This function inserts an item into the linked list structure after the specified element. The `rtxMemAlloc` function is used to allocate memory for the list node structure; therefore, all internal list memory will be released when the `rtxMemFree` function is called.

**Parameters**

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

***pList*** A pointer to a linked list structure into which the data item is to be inserted.

***node*** The position in the list where the item is to be inserted. The item will be inserted after this node or added as the head element if node is null.

***pData*** A pointer to the data item to be inserted to the list.

**Returns**

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

#### 2.25.2.15 OSRTDListNode* rtxDListInsertBefore (struct OSCTXT * *pctxt*, OSRTDList * *pList*, OSRTDListNode * *node*, void * *pData*)

This function inserts an item into the linked list structure before the specified element. The `rtxMemAlloc` function is used to allocate memory for the list node structure; therefore, all internal list memory will be released when the `rtxMemFree` function is called.

**Parameters**

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

***pList*** A pointer to a linked list structure into which the data item is to be inserted.

***node*** The position in the list where the item is to be inserted. The item will be inserted before this node or appended to the list if node is null.

***pData*** A pointer to the data item to be inserted to the list.

**Returns**

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

#### 2.25.2.16 void rtxDListRemove (OSRTDList * *pList*, OSRTDListNode * *node*)

This function will remove the given node from the list. The node memory is not freed. It will be released when the `rtxMemFree` or `rtFreeContext` function is called with this context.

**Parameters**

***pList*** A pointer to a linked list structure.

***node*** Pointer to the list node to be removed.

#### 2.25.2.17 int rtxDListToArray (struct OSCTXT * *pctxt*, OSRTDList * *pList*, void ** *ppArray*, OSSIZE * *pElemCount*, OSSIZE *elemSize*)

This function converts a doubly linked list to an array.

**Parameters**

***pctxt*** A pointer to a context structure.

***pList*** A pointer to a linked list structure.

***ppArray*** A pointer to a pointer to the destination array.

***pElemCount*** A pointer to the number of elements already allocated in `ppArray`. If pElements is NULL, or pElements is less than the number of nodes in the list, then a new array is allocated and the pointer is stored in `ppArray`. Memory is allocated via calls to the `rtxMemAlloc` function.

***elemSize*** The size of one element in the array. Use the `sizeof()` operator to pass this parameter.

**Returns**

The number of elements in the returned array.

#### 2.25.2.18 int rtxDListToUTF8Str (struct OSCTXT * *pctxt*, OSRTDList * *pList*, OSUTF8CHAR ** *ppstr*, char *sep*)

This function concatanates all of the components in the given list to form a UTF-8 string. The list is assumed to contain null-terminated character string components. The given separator chraacter is inserted after each list component. The `rtxMemAlloc` function is used to allocate memory for the output string.

**Parameters**

***pctxt*** A pointer to a context structure.

***pList*** A pointer to the linked list structure onto which the array items will be appended.

***ppstr*** A pointer to a char pointer to hold output string.

***sep*** Separator character to add between string components.

**Returns**

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

## 2.26 Linked List Utility Functions

### Classes

- struct _OSRTSListNode
- struct _OSRTSList

### Typedefs

- typedef struct _OSRTSListNode **OSRTSListNode**
- typedef struct _OSRTSList **OSRTSList**

### Functions

- void rtxSListInit (OSRTSList *pList)
- void rtxSListInitEx (OSCTXT *pctxt, OSRTSList *pList)
- void rtxSListFree (OSRTSList *pList)
- void rtxSListFreeAll (OSRTSList *pList)
- OSRTSList * rtxSListCreate (void)
- OSRTSList * rtxSListCreateEx (OSCTXT *pctxt)
- OSRTSListNode * rtxSListAppend (OSRTSList *pList, void *pData)
- OSBOOL rtxSListFind (OSRTSList *pList, void *pData)
- void rtxSListRemove (OSRTSList *pList, void *pData)

### 2.26.1 Detailed Description

Singly linked list structures have only a single link pointer and can therefore only be traversed in a single direction (forward). The node structures consume less memory than those of a doubly linked list.

Another difference between the singly linked list implementation and doubly linked lists is that the singly linked list uses conventional memory allocation functions (C malloc and free) for less storage instead of the rtxMem functions. Therefore, it is not a requirement to have an initialized context structure to work with these lists. However, performance may suffer if the lists become large due to the use of non-optimized memory management.

### 2.26.2 Function Documentation

#### 2.26.2.1 OSRTSListNode* rtxSListAppend (OSRTSList * *pList*, void * *pData*)

This function appends an item to a linked list structure. The data item is passed into the function as a void parameter that can point to an object of any type.

**Parameters**

***pList*** A pointer to a linked list onto which the data item is to be appended.

***pData*** A pointer to a data item to be appended to the list.

**Returns**

A pointer to the allocated linked list structure.

#### 2.26.2.2 OSRTSList* rtxSListCreate (void)

This function creates a new linked list structure. It allocates memory for the structure and calls rtxSListInit to initialize the structure.

**Returns**

A pointer to the allocated linked list structure.

#### 2.26.2.3 OSRTSList* rtxSListCreateEx (OSCTXT * *pctxt*)

The rtxSListAppend function appends an item to linked list structure. The data is passed into the function as a void that can point to an object of any 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.

**Returns**

A pointer to the allocated linked list structure.

#### 2.26.2.4 OSBOOL rtxSListFind (OSRTSList * *pList*, void * *pData*)

This function finds an item in the linked list structure. The data item is passed into the function as a void pointer that can point to an object of any type. If the appropriate node is found in the list this function returns TRUE, otherwise FALSE.

**Parameters**

***pList*** A pointer to a linked list onto which the data item is to be appended.

***pData*** A pointer to a data item to be appended to the list.

**Returns**

TRUE, if the node is found, otherwise FALSE.

#### 2.26.2.5 void rtxSListFree (OSRTSList * *pList*)

This function removes all nodes from the linked list structure and releases memory that was allocated for storing node structures (OSRTSListNode). The data will not be freed.

**Parameters**

***pList*** A pointer to a linked list onto which the data item is to be appended.

#### 2.26.2.6 void rtxSListFreeAll (OSRTSList * *pList*)

This function removes all nodes from the linked list structure and releases memory that was allocated for storing node structures (OSRTSListNode). It also frees the data structures which are assumed to have been allocated using the rtxMemAlloc function.

**Parameters**

***pList*** A pointer to a linked list onto which the data item is to be appended.

#### 2.26.2.7 void rtxSListInit (OSRTSList * *pList*)

This function initializes a singly linked list structure. It sets the number of elements to zero and sets all internal pointer values to NULL.

**Parameters**

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

#### 2.26.2.8 void rtxSListInitEx (OSCTXT * *pctxt*, OSRTSList * *pList*)

This function is similar to rtxSListInit but it also sets the pctxt (pointer to a context structure member of OSRTSList structure). This context will be used for further memory allocations; otherwise the standard malloc is used.

**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 to be initialized.

#### 2.26.2.9 void rtxSListRemove (OSRTSList * *pList*, void * *pData*)

This function finds an item in the linked list structure and removes it from the list. The data item is passed into the function as a void pointer that can point to an object of any type.

**Parameters**

***pList*** A pointer to a linked list onto which the data item is to be appended.

***pData*** A pointer to a data item to be appended to the list.

## 2.27 Stack Utility Functions

### Classes

- struct _OSRTStack

### Defines

- #define rtxStackIsEmpty(stack) (OSBOOL)((stack).dlist.count == 0)

### Typedefs

- typedef struct _OSRTStack **OSRTStack**

### Functions

- OSRTStack * rtxStackCreate (OSCTXT *pctxt)
- void rtxStackInit (OSCTXT *pctxt, OSRTStack *pStack)
- void * rtxStackPop (OSRTStack *pStack)
- int rtxStackPush (OSRTStack *pStack, void *pData)
- void * rtxStackPeek (OSRTStack *pStack)

### 2.27.1 Detailed Description

This is a simple stack structure with supporting push and pop functions. Different initialization routines are provided to permit different memory management schemes.

### 2.27.2 Define Documentation

#### 2.27.2.1 #define rtxStackIsEmpty(stack) (OSBOOL)((stack).dlist.count == 0)

This macro tests if the stack is empty.

**Parameters**

***stack*** Stack structure variable to be tested.

### 2.27.3 Function Documentation

#### 2.27.3.1 OSRTStack* rtxStackCreate (OSCTXT * *pctxt*)

This function creates a new stack structure. It allocates memory for the structure and calls rtxStackInit to initialize the structure.

**Parameters**

***pctxt*** A pointer to the context with which the stack is associated.

**Returns**

A pointer to an allocated stack structure.

#### 2.27.3.2 void rtxStackInit (OSCTXT * *pctxt*, OSRTStack * *pStack*)

This function initializes a stack structure. It sets the number of elements to zero and sets all internal pointer values to NULL.

**Parameters**

***pctxt*** A pointer to the context with which the stack is associated.

***pStack*** A pointer to a stack structure to be initialized.

#### 2.27.3.3 void* rtxStackPeek (OSRTStack * *pStack*)

This functions returns the data item on the top of the stack.

**Parameters**

***pStack*** A pointer to the structure onto which the data item is to be pushed. The pointer updated to the stack structure

**Returns**

Pointer to data item at top of stack or NULL if stack empty.

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

#### 2.27.3.4 void* rtxStackPop (OSRTStack * *pStack*)

This function pops an item off the stack.

**Parameters**

***pStack*** The pointer to the stack structure from which the value is to be popped. Pointer to the updated stack structure.

**Returns**

The pointer to the item popped from the stack

#### 2.27.3.5 int rtxStackPush (OSRTStack * *pStack*, void * *pData*)

This function pushes an item onto the stack.

**Parameters**

***pStack*** A pointer to the structure onto which the data item is to be pushed. The pointer updated to the stack structure

***pData*** A pointer to the data item to be pushed on the stack.

**Returns**

Completion status of operation:

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

## 2.28 Pattern matching functions

### Functions

- OSBOOL rtxMatchPattern (OSCTXT *pctxt, const OSUTF8CHAR *text, const OSUTF8CHAR *pattern)
- OSBOOL **rtxMatchPattern2** (OSCTXT *pctxt, const OSUTF8CHAR *pattern)
- void rtxFreeRegexpCache (OSCTXT *pctxt)

### 2.28.1 Detailed Description

These functions handle pattern matching which is required to to process XML schema pattern constraints.

### 2.28.2 Function Documentation

#### 2.28.2.1 void rtxFreeRegexpCache (OSCTXT * *pctxt*)

This function frees the memory associated with the regular expression cache. The regular expression cache is designed to use memory that survives calls to rtxMemFree and rtxMemReset, therefore it is necessary to call this function to free that memory. (Note that `rtxFreeContext` invokes this.)

#### 2.28.2.2 OSBOOL rtxMatchPattern (OSCTXT * *pctxt*, const OSUTF8CHAR * *text*, const OSUTF8CHAR * *pattern*)

This function compares the given string to the given pattern. It returns true if match, false otherwise.

**Parameters**

***pctxt*** Pointer to context structure.

***text*** Text to be matched.

***pattern*** Regular expression.

**Returns**

Boolean result.

## 2.29 Diagnostic trace functions

### Classes

- struct OSRTPrintStream

### Defines

- #define **RTDIAG1**(pctxt, msg)
- #define **RTDIAG2**(pctxt, msg, a)
- #define **RTDIAG3**(pctxt, msg, a, b)
- #define **RTDIAG4**(pctxt, msg, a, b, c)
- #define **RTDIAG5**(pctxt, msg, a, b, c, d)
- #define **RTDIAGU**(pctxt, ucstr)
- #define **RTHEXDUMP**(pctxt, buffer, numocts)
- #define **RTDIAGCHARS**(pctxt, buf, nchars)
- #define **RTDIAGSTRM2**(pctxt, msg)
- #define **RTDIAGSTRM3**(pctxt, msg, a)
- #define **RTDIAGSTRM4**(pctxt, msg, a, b)
- #define **RTDIAGSTRM5**(pctxt, msg, a, b, c)
- #define **RTHEXDUMPSTRM**(pctxt, buffer, numocts)
- #define **RTDIAGSCHARS**(pctxt, buf, nchars)

### Typedefs

- typedef void(∗ rtxPrintCallback )(void ∗pPrntStrmInfo, const char ∗fmtspec, va_list arglist)
- typedef struct OSRTPrintStream OSRTPrintStream

### Enumerations

- enum **OSRTDiagTraceLevel** { **OSRTDiagError**, **OSRTDiagWarning**, **OSRTDiagInfo**, **OSRTDiagDebug** }

### Functions

- int rtxSetPrintStream (OSCTXT ∗pctxt, rtxPrintCallback myCallback, void ∗pStrmInfo)
- int rtxSetGlobalPrintStream (rtxPrintCallback myCallback, void ∗pStrmInfo)
- int rtxPrintToStream (OSCTXT ∗pctxt, const char ∗fmtspec,...)
- int rtxDiagToStream (OSCTXT ∗pctxt, const char ∗fmtspec, va_list arglist)
- int rtxPrintStreamRelease (OSCTXT ∗pctxt)
- void rtxPrintStreamToStdoutCB (void ∗pPrntStrmInfo, const char ∗fmtspec, va_list arglist)
- void rtxPrintStreamToFileCB (void ∗pPrntStrmInfo, const char ∗fmtspec, va_list arglist)
- OSBOOL rtxDiagEnabled (OSCTXT ∗pctxt)
- OSBOOL rtxSetDiag (OSCTXT ∗pctxt, OSBOOL value)
- OSBOOL rtxSetGlobalDiag (OSBOOL value)
- void rtxDiagPrint (OSCTXT ∗pctxt, const char ∗fmtspec,...)
- void rtxDiagStream (OSCTXT ∗pctxt, const char ∗fmtspec,...)
- void rtxDiagHexDump (OSCTXT ∗pctxt, const OSOCTET ∗data, size_t numocts)
- void rtxDiagStreamHexDump (OSCTXT ∗pctxt, const OSOCTET ∗data, size_t numocts)
- void rtxDiagPrintChars (OSCTXT ∗pctxt, const char ∗data, size_t nchars)

- void rtxDiagStreamPrintChars (OSCTXT *pctxt, const char *data, size_t nchars)
- void rtxDiagStreamPrintBits (OSCTXT *pctxt, const char *descr, const OSOCTET *data, size_t bitIndex, size_t nbits)
- void rtxDiagSetTraceLevel (OSCTXT *pctxt, OSRTDiagTraceLevel level)
- OSBOOL rtxDiagTraceLevelEnabled (OSCTXT *pctxt, OSRTDiagTraceLevel level)

## Variables

- OSRTPrintStream g_PrintStream

### 2.29.1 Detailed Description

These functions add diagnostic tracing to the generated code to assist in finding where a problem might occur. Calls to these macros and functions are added when the `-trace` command-line argument is used. The diagnostic message can be turned on and off at both C compile and run-time. To C compile the diagnostics in, the _TRACE macro must be defined (-D_TRACE). To turn the diagnostics on at run-time, the `rtxSetDiag` function must be invoked with the `value` argument set to TRUE.

### 2.29.2 Typedef Documentation

#### 2.29.2.1 typedef struct OSRTPrintStream OSRTPrintStream

Structure to hold information about a global PrintStream.

#### 2.29.2.2 typedef void(* rtxPrintCallback)(void *pPrntStrmInfo, const char *fmtspec, va_list arglist)

Callback function definition for print stream.

### 2.29.3 Function Documentation

#### 2.29.3.1 OSBOOL rtxDiagEnabled (OSCTXT * *pctxt*)

This function is used to determine if diagnostic tracing is currently enabled for the specified context. It returns true if enabled, false otherwise.

**Parameters**

***pctxt*** Pointer to context structure.

**Returns**

Boolean result.

#### 2.29.3.2 void rtxDiagHexDump (OSCTXT * *pctxt*, const OSOCTET * *data*, size_t *numocts*)

This function is used to print a diagnostics hex dump of a section of memory.

**Parameters**

***pctxt*** Pointer to context structure.

***data*** Start address of memory to dump.

***numocts*** Number of bytes to dump.

#### 2.29.3.3 void rtxDiagPrint (OSCTXT * *pctxt*, const char * *fmtspec*, ...)

This function is used to print a diagnostics message to `stdout`. Its parameter specification is similar to that of the C runtime `printf` method. The `fmtspec` argument may contain % style modifiers into which variable arguments are substituted. This function is called in the generated code via the RTDIAG macros to allow diagnostic trace call to easily be compiled out of the object code.

**Parameters**

***pctxt*** Pointer to context structure.

***fmtspec*** A printf-like format specification string describing the message to be printed (for example, "string %s, ivalue %d\n"). A special character sequence (~L) may be used at the beginning of the string to select a trace level (L would be replaced with E for Error, W for warning, I for info, or D for debug).

**...** Variable list of parameters to be substituted into the format string.

#### 2.29.3.4 void rtxDiagPrintChars (OSCTXT * *pctxt*, const char * *data*, size_t *nchars*)

This function is used to print a given number of characters to standard output. The buffer containing the characters does not need to be null-terminated.

**Parameters**

***pctxt*** Pointer to context structure.

***data*** Start address of character string.

***nchars*** Number of characters to dump (this assumes 1-byte chars).

#### 2.29.3.5 void rtxDiagSetTraceLevel (OSCTXT * *pctxt*, OSRTDiagTraceLevel *level*)

This function is used to set the maximum trace level for diagnostic trace messages. Values are ERROR, WARNING, INFO, or DEBUG. The special string start sequence (~L) described in rtxDiagPrint function documentation is used to set a message level to be compared with the trace level.

**Parameters**

***pctxt*** Pointer to context structure.

***level*** Trace level to be set.

#### 2.29.3.6 void rtxDiagStream (OSCTXT * *pctxt*, const char * *fmtspec*, ...)

This function conditionally outputs diagnostic trace messages to an output stream defined within the context. A code generator embeds calls to this function into the generated source code when the -trace option is specified on the command line (note: it may embed the macro version of these calls - RTDIAGSTREAMx - so that these calls can be compiled out of the final code.

**See also**

rtxDiagPrint

#### 2.29.3.7 void rtxDiagStreamHexDump (OSCTXT * *pctxt*, const OSOCTET * *data*, size_t *numocts*)

This function is used to print a diagnostics hex dump of a section of memory to a print stream.

**Parameters**

***pctxt*** Pointer to context structure.

***data*** Start address of memory to dump.

***numocts*** Number of bytes to dump.

#### 2.29.3.8 void rtxDiagStreamPrintBits (OSCTXT * *pctxt*, const char * *descr*, const OSOCTET * *data*, size_t *bitIndex*, size_t *nbits*)

This function is used to print a given number of bits as '1' or '0' values to a print stream.

**Parameters**

***pctxt*** Pointer to context structure.

***descr*** Descriptive text to print before bits

***data*** Start address of binary data.

***bitIndex*** Zero-based offset to first bit to be printed

***nbits*** Number of bits to dump

#### 2.29.3.9 void rtxDiagStreamPrintChars (OSCTXT * *pctxt*, const char * *data*, size_t *nchars*)

This function is used to print a given number of characters to a print stream. The buffer containing the characters does not need to be null-terminated.

**Parameters**

***pctxt*** Pointer to context structure.

***data*** Start address of character string.

***nchars*** Number of characters to dump (this assumes 1-byte chars).

#### 2.29.3.10 int rtxDiagToStream (OSCTXT * *pctxt*, const char * *fmtspec*, va_list *arglist*)

Diagnostics print-to-stream function. This is the same as the `rtxPrintToStream` function except that it checks if diagnostic tracing is enabled before invoking the callback function.

**Parameters**

***pctxt*** Pointer to context to be used.

***fmtspec*** A printf-like format specification string describing the message to be printed (for example, "string %s, ivalue %d\n").

***arglist*** A variable list of arguments passed as va_list

**Returns**

Completion status, 0 on success, negative value on failure

#### 2.29.3.11 OSBOOL rtxDiagTraceLevelEnabled (OSCTXT * *pctxt*, OSRTDiagTraceLevel *level*)

This function tests if a given trace level is enabled.

**Parameters**

***pctxt*** Pointer to context structure.

***level*** Trace level to check.

**Returns**

True if enabled.

#### 2.29.3.12 int rtxPrintStreamRelease (OSCTXT * *pctxt*)

This function releases the memory held by PrintStream in the context

**Parameters**

***pctxt*** Pointer to a context for which the memory has to be released.

**Returns**

Completion status, 0 on success, negative value on failure

#### 2.29.3.13 void rtxPrintStreamToFileCB (void * *pPrntStrmInfo*, const char * *fmtspec*, va_list *arglist*)

Standard callback function for use with print-to-stream for writing to a file.

**Parameters**

***pPrntStrmInfo*** User-defined information for use by the callback function. This is supplied by the user at the time the callback is registered. This parameter should be set to the file pointer (FILE*) to which data is to be written.

***fmtspec*** Format specification of the data to be printed. This is supplied by the print-to-stream utility.

***arglist*** Variable argument list. This is supplied by the print-to-stream utility.

#### 2.29.3.14 void rtxPrintStreamToStdoutCB (void * *pPrntStrmInfo*, const char * *fmtspec*, va_list *arglist*)

Standard callback function for use with print-to-stream for writing to stdout.

**Parameters**

***pPrntStrmInfo*** User-defined information for use by the callback function. This is supplied by the user at the time the callback is registered. In this case, no user-defined information is required, so the argument can be set to NULL when the callback is registered.

***fmtspec*** Format specification of the data to be printed. This is supplied by the print-to-stream utility.

***arglist*** Variable argument list. This is supplied by the print-to-stream utility.

#### 2.29.3.15 int rtxPrintToStream (OSCTXT * *pctxt*, const char * *fmtspec*, ...)

Print-to-stream function which in turn calls the user registered callback function of the context for printing. If no callback function is registered it prints to standard output by default.

**Parameters**

***pctxt*** Pointer to context to be used.

***fmtspec*** A printf-like format specification string describing the message to be printed (for example, "string %s, ivalue %d\n").

**...** A variable list of arguments.

**Returns**

Completion status, 0 on success, negative value on failure

#### 2.29.3.16 OSBOOL rtxSetDiag (OSCTXT * *pctxt*, OSBOOL *value*)

This function is used to turn diagnostic tracing on or off at run-time on a per-context basis. Code generated using ASN1C or XBinder or a similar code generator must use the -trace command line option to enable diagnostic messages. The generated code must then be C compiled with _TRACE defined for the code to be present.

**Parameters**

***pctxt*** Pointer to context structure.

***value*** Boolean switch: TRUE turns tracing on, FALSE off.

**Returns**

Prior setting of the diagnostic trace switch in the context.

#### 2.29.3.17 OSBOOL rtxSetGlobalDiag (OSBOOL *value*)

This function is used to turn diagnostic tracing on or off at run-time on a global basis. It is similar to rtxSetDiag except tracing is enabled within all contexts.

**Parameters**

***value*** Boolean switch: TRUE turns tracing on, FALSE off.

**Returns**

Prior setting of the diagnostic trace switch in the context.

#### 2.29.3.18 int rtxSetGlobalPrintStream (rtxPrintCallback *myCallback*, void * *pStrmInfo*)

This function is for setting the callback function for a PrintStream. This version of the function sets a callback at the global level.

**Parameters**

***myCallback*** Pointer to a callback print function.

***pStrmInfo*** Pointer to user defined PrintInfo structure where users can store information required by the callback function across calls. Ex. An open File handle for callbak function which directs stream to a file.

**Returns**

Completion status, 0 on success, negative value on failure

#### 2.29.3.19 int rtxSetPrintStream (OSCTXT * *pctxt*, rtxPrintCallback *myCallback*, void * *pStrmInfo*)

This function is for setting the callback function for a PrintStream. Once a callback function is set, then all print and debug output ia sent to the defined callback function.

**Parameters**

***pctxt*** Pointer to a context in which callback print function will be set

***myCallback*** Pointer to a callback print function.

***pStrmInfo*** Pointer to user defined PrintInfo structure where users can store information required by the callback function across calls. Ex. An open File handle for callbak function which directs stream to a file.

**Returns**

Completion status, 0 on success, negative value on failure

### 2.29.4 Variable Documentation

#### 2.29.4.1 OSRTPrintStream g_PrintStream

Global PrintStream

## 2.30 Error Formatting and Print Functions

### Defines

- #define LOG_RTERR(pctxt, stat) rtxErrSetData(pctxt,stat,__FILE__,__LINE__)
- #define **LOG_RTERRNEW**(pctxt, stat) rtxErrSetNewData(pctxt,stat,__FILE__,__LINE__)
- #define OSRTASSERT(condition) if (!(condition)) { rtxErrAssertionFailed(#condition,__LINE__,__FILE__); }
- #define OSRTCHECKPARAM(condition) if (condition) { /* do nothing */ }
- #define **LOG_RTERR1**(pctxt, stat, a) (a,LOG_RTERR (pctxt, stat),stat)
- #define **LOG_RTERRNEW1**(pctxt, stat, a) (a,LOG_RTERRNEW (pctxt, stat),stat)
- #define **LOG_RTERR2**(pctxt, stat, a, b) (a,b,LOG_RTERR (pctxt, stat),stat)
- #define **LOG_RTERRNEW2**(pctxt, stat, a, b) (a,b,LOG_RTERRNEW (pctxt, stat),stat)

### Typedefs

- typedef int(* **OSErrCbFunc** )(const char *ptext, void *cbArg_p)

### Functions

- OSBOOL rtxErrAddCtxtBufParm (OSCTXT *pctxt)
- OSBOOL rtxErrAddDoubleParm (OSCTXT *pctxt, double errParm)
- OSBOOL rtxErrAddErrorTableEntry (const char *const *ppStatusText, OSINT32 minErrCode, OSINT32 maxErrCode)
- OSBOOL rtxErrAddElemNameParm (OSCTXT *pctxt)
- OSBOOL rtxErrAddIntParm (OSCTXT *pctxt, int errParm)
- OSBOOL rtxErrAddInt64Parm (OSCTXT *pctxt, OSINT64 errParm)
- OSBOOL rtxErrAddStrParm (OSCTXT *pctxt, const char *pErrParm)
- OSBOOL rtxErrAddStrnParm (OSCTXT *pctxt, const char *pErrParm, size_t nchars)
- OSBOOL rtxErrAddUIntParm (OSCTXT *pctxt, unsigned int errParm)
- OSBOOL rtxErrAddUInt64Parm (OSCTXT *pctxt, OSUINT64 errParm)
- void rtxErrAssertionFailed (const char *conditionText, int lineNo, const char *fileName)
- const char * rtxErrFmtMsg (OSRTErrInfo *pErrInfo, char *bufp, size_t bufsiz)
- void rtxErrFreeParms (OSCTXT *pctxt)
- char * rtxErrGetText (OSCTXT *pctxt, char *pBuf, size_t *pBufSize)
- char * rtxErrGetTextBuf (OSCTXT *pctxt, char *pbuf, size_t bufsiz)
- OSRTErrInfo * rtxErrNewNode (OSCTXT *pctxt)
- void rtxErrInit ()
- int rtxErrReset (OSCTXT *pctxt)
- void rtxErrLogUsingCB (OSCTXT *pctxt, OSErrCbFunc cb, void *cbArg_p)
- void rtxErrPrint (OSCTXT *pctxt)
- void rtxErrPrintElement (OSRTErrInfo *pErrInfo)
- int rtxErrSetData (OSCTXT *pctxt, int status, const char *module, int lineno)
- int rtxErrSetNewData (OSCTXT *pctxt, int status, const char *module, int lineno)
- int rtxErrGetFirstError (const OSCTXT *pctxt)
- int rtxErrGetLastError (const OSCTXT *pctxt)
- OSSIZE rtxErrGetErrorCnt (const OSCTXT *pctxt)
- int rtxErrGetStatus (const OSCTXT *pctxt)
- int rtxErrResetLastErrors (OSCTXT *pctxt, int errorsToReset)
- int rtxErrCopy (OSCTXT *pDestCtxt, const OSCTXT *pSrcCtxt)
- int rtxErrAppend (OSCTXT *pDestCtxt, const OSCTXT *pSrcCtxt)
- int rtxErrInvUIntOpt (OSCTXT *pctxt, OSUINT32 ident)

### 2.30.1 Detailed Description

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

### 2.30.2 Define Documentation

#### 2.30.2.1 #define LOG_RTERR(pctxt, stat) rtxErrSetData(pctxt,stat,__FILE__,__LINE__)

This macro is used to log a run-time error in the context. It calls the `rtxErrSetData` function to set the status and error parameters. The C built-in __FILE__ and __LINE__ macros are used to record the position in the source file of the error.

**Parameters**

***pctxt*** A pointer to a context structure.

***stat*** Error status value from `rtxErrCodes.h`

#### 2.30.2.2 #define OSRTASSERT(condition) if (!(condition)) { rtxErrAssertionFailed(#condition,__LINE__,__FILE__); }

This macro is used to check an assertion. This is a condition that is expected to be true. The `rtxErrAssertionFailed` function is called if the condition is not true. The C built-in __FILE__ and __LINE__ macros are used to record the position in the source file of the failure.

**Parameters**

***condition*** Condition to check (for example, "(ptr != NULL)")

#### 2.30.2.3 #define OSRTCHECKPARAM(condition) if (condition) { /* do nothing */ }

This macro check a condition but takes no action. Its main use is to supress VC++ level 4 "argument not used" warnings.

**Parameters**

***condition*** Condition to check (for example, "(ptr != NULL)")

### 2.30.3 Function Documentation

#### 2.30.3.1 OSBOOL rtxErrAddCtxtBufParm (OSCTXT * *pctxt*)

This function adds the contents of the context buffer to the error information structure in the context. The buffer contents are assumed to contain only printable characters.

**Parameters**

***pctxt*** A pointer to a context structure.

**Returns**

The status of the operation (TRUE if the parameter was sucessfully added).

#### 2.30.3.2 OSBOOL rtxErrAddDoubleParm (OSCTXT * *pctxt*, double *errParm*)

This function adds a double parameter to an error information structure.

**Parameters**

***pctxt*** A pointer to a context structure.
***errParm*** The double error parameter.

**Returns**

The status of the operation (TRUE if the parameter was sucessfully added).

#### 2.30.3.3 OSBOOL rtxErrAddElemNameParm (OSCTXT * *pctxt*)

This function adds an element name parameter to the context error information structure. The element name is obtained from the context element name stack. If the stack is empty, a question mark characetr (?) is inserted for the name.

**Parameters**

***pctxt*** A pointer to a context structure.

**Returns**

The status of the operation (TRUE if the parameter was sucessfully added).

#### 2.30.3.4 OSBOOL rtxErrAddErrorTableEntry (const char *const * *ppStatusText*, OSINT32 *minErrCode*, OSINT32 *maxErrCode*)

This function adds a set of error codes to the global error table. It is called within context initialization functions to add errors defined for a specific domain (for example, ASN.1 encoding/decoding) to be added to the global list of errors.

**Parameters**

***ppStatusText*** Pointer to table of error status text messages.
***minErrCode*** Minimum error status code.
***maxErrCode*** Maximum error status code.

**Returns**

The status of the operation (TRUE if entry sucessfully added to table).

#### 2.30.3.5 OSBOOL rtxErrAddInt64Parm (OSCTXT * *pctxt*, OSINT64 *errParm*)

This function adds a 64-bit 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**

***pctxt*** A pointer to a context structure.
***errParm*** The 64-bit integer error parameter.

**Returns**

The status of the operation (TRUE if the parameter was sucessfully added).

#### 2.30.3.6 OSBOOL rtxErrAddIntParm (OSCTXT * *pctxt*, 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**

***pctxt*** A pointer to a context structure.

***errParm*** The integer error parameter.

**Returns**

The status of the operation (TRUE if the parameter was sucessfully added).

#### 2.30.3.7 OSBOOL rtxErrAddStrnParm (OSCTXT * *pctxt*, const char * *pErrParm*, size_t *nchars*)

This function adds a given number of characters from a character string parameter to an error information structure.

**Parameters**

***pctxt*** A pointer to a context structure.

***pErrParm*** The character string error parameter.

***nchars*** Number of characters to add from pErrParm.

**Returns**

The status of the operation (TRUE if the parameter was sucessfully added).

#### 2.30.3.8 OSBOOL rtxErrAddStrParm (OSCTXT * *pctxt*, const char * *pErrParm*)

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

**Parameters**

***pctxt*** A pointer to a context structure.

***pErrParm*** The character string error parameter.

**Returns**

The status of the operation (TRUE if the parameter was sucessfully added).

#### 2.30.3.9 OSBOOL rtxErrAddUInt64Parm (OSCTXT * *pctxt*, OSUINT64 *errParm*)

This function adds an unsigned 64-bit integer parameter to an error information structure.

**Parameters**

***pctxt*** A pointer to a context structure.

***errParm*** The unsigned 64-bit integer error parameter.

**Returns**

The status of the operation (TRUE if the parameter was sucessfully added).

#### 2.30.3.10 OSBOOL rtxErrAddUIntParm (OSCTXT * *pctxt*, unsigned int *errParm*)

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

**Parameters**

***pctxt*** A pointer to a context structure.

***errParm*** The unsigned integer error parameter.

**Returns**

The status of the operation (TRUE if the parameter was sucessfully added).

#### 2.30.3.11 int rtxErrAppend (OSCTXT * *pDestCtxt*, const OSCTXT * *pSrcCtxt*)

This function appends error information from one context into another. Error information is added to what previously existed in the destination context.

**Parameters**

***pDestCtxt*** Pointer to destination context structure to receive the copied error information.

***pSrcCtxt*** Pointer to source context from which error information will be copied.

**Returns**

Completion status of operation:

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

#### 2.30.3.12 void rtxErrAssertionFailed (const char * *conditionText*, int *lineNo*, const char * *fileName*)

This function is used to record an assertion failure. It is used in conjunction with the `OTRTASSERT` macro. It outputs information on the condition to `stderr` and then calls `exit` to terminate the program.

**Parameters**

***conditionText*** The condition that failed (for example, ptr != NULL)

***lineNo*** Line number in the program of the failure.

***fileName*** Name of the C source file in which the assertion failure occurred.

#### 2.30.3.13 int rtxErrCopy (OSCTXT * *pDestCtxt*, const OSCTXT * *pSrcCtxt*)

This function copies error information from one context into another. Any error information that may have existed in the destination context prior to the operation is lost.

**Parameters**

***pDestCtxt*** Pointer to destination context structure to receive the copied error information.

***pSrcCtxt*** Pointer to source context from which error information will be copied.

**Returns**

Completion status of operation:

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

#### 2.30.3.14 const char* rtxErrFmtMsg (OSRTErrInfo * *pErrInfo*, char * *bufp*, size_t *bufsiz*)

This function formats a given error structure from the context into a finished status message including substituted parameters.

**Parameters**

***pErrInfo*** Pointer to error information structure.

***bufp*** Pointer to a destination buffer to receive text.

***bufsiz*** Size of the buffer.

**Returns**

Pointer to the buffer (bufp).

#### 2.30.3.15 void rtxErrFreeParms (OSCTXT * *pctxt*)

This function is used to free dynamic memory that was used in the recording of error parameters. After an error is logged, this function should be called to prevent memory leaks.

**Parameters**

***pctxt*** A pointer to a context structure.

#### 2.30.3.16 OSSIZE rtxErrGetErrorCnt (const OSCTXT * *pctxt*)

This function returns the total number of error records.

**Parameters**

***pctxt*** A pointer to a context structure.

**Returns**

The total number of error records in the context.

#### 2.30.3.17 int rtxErrGetFirstError (const OSCTXT * *pctxt*)

This function returns the error code, stored in the first error record.

**Parameters**

***pctxt*** A pointer to a context structure.

**Returns**

The first status code; zero if no error records exist.

#### 2.30.3.18 int rtxErrGetLastError (const OSCTXT * *pctxt*)

This function returns the error code, stored in the last error record.

**Parameters**

***pctxt*** A pointer to a context structure.

**Returns**

The last status code; zero if no error records exist.

#### 2.30.3.19 int rtxErrGetStatus (const OSCTXT * *pctxt*)

This function returns the status value from the context. It examines the error list to see how many errors were logged. If none, OK (zero) is returned; if one, then the status value in the single error record is returned; if more than one, the special code RTERR_MULTIPLE is returned to indicate that multiple errors occurred.

**Parameters**

***pctxt*** A pointer to a context structure.

**Returns**

Status code corresponding to errors in the context.

#### 2.30.3.20 char* rtxErrGetText (OSCTXT * *pctxt*, char * *pBuf*, size_t * *pBufSize*)

This function returns error text in a memory buffer. If buffer pointer and buffer size are specified in parameters (not NULL) then error text will be copied in the passed buffer. Otherwise, this function allocates memory using the 'rtxMemAlloc' function. This memory is automatically freed at the time the 'rtxMemFree' or 'rtxFreeContext' functions are called. The calling function may free the memory by using 'rtxMemFreePtr' function.

**Parameters**

***pctxt*** A pointer to a context structure.

***pBuf*** A pointer to a destination buffer to obtain the error text. If NULL, dynamic buffer will be allocated.

***pBufSize*** A pointer to buffer size. If pBuf is NULL and this parameter is not, it will receive the size of allocated dynamic buffer.

**Returns**

A pointer to a buffer with error text. If pBuf is not NULL, the return pointer will be equal to it. Otherwise, returns newly allocated buffer with error text. NULL, if error occurred.

#### 2.30.3.21 char* rtxErrGetTextBuf (OSCTXT * *pctxt*, char * *pbuf*, size_t *bufsiz*)

This function returns error text in the given fixed-size memory buffer. If the text will not fit in the buffer, it is truncated.

**Parameters**

***pctxt*** A pointer to a context structure.

***pbuf*** Pointer to a destination buffer to receive text.

***bufsiz*** Size of the buffer.

**Returns**

Pointer to the buffer (pbuf).

#### 2.30.3.22 void rtxErrInit ()

This function is a one-time initialization function that must be called before any other error processing functions can be called. It adds the common error status text codes to the global error table.

#### 2.30.3.23 int rtxErrInvUIntOpt (OSCTXT * *pctxt*, OSUINT32 *ident*)

This function create an 'invalid option' error (RTERR_INVOPT) in the context using an unsigned integer parameter.

**Parameters**

***pctxt*** Pointer to context structure.

***ident*** Identifier which was found to be invalid.

#### 2.30.3.24 void rtxErrLogUsingCB (OSCTXT * *pctxt*, OSErrCbFunc *cb*, void * *cbArg_p*)

This function allows error information to be logged using a user-defined callback routine. The callback routine is invoked IMMEDIATELY; it is not a "callback" in the ordinary use of that word. The callback routine is invoked with error information in the context allowing the user to do application-specific handling (for example, it can be written to an error log or a Window display). After invoking the callback, this method will invoked rtxErrFreeParms to free memory associated with the error information.

The prototype of the callback function to be passed is as follows:

*int cb (const char* ptext, void* cbArg_p);*

where *ptext* is a pointer to the formatted error text string and *cbArg_p* is a pointer to a user-defined callback argument.

**Parameters**

***pctxt*** A pointer to a context structure.

***cb*** Callback function pointer.

***cbArg_p*** Pointer to a user-defined argument that is passed to the callback function.

#### 2.30.3.25 OSRTErrInfo* rtxErrNewNode (OSCTXT * *pctxt*)

This function creates a new empty error record for the passed context. `rtxErrSetData` function call with bAlloc-New = FALSE should be used to set the data for this node.

**Parameters**

***pctxt*** A pointer to a context structure.

**Returns**

A pointer to a newly allocated error record; NULL, if error occurred.

#### 2.30.3.26 void rtxErrPrint (OSCTXT * *pctxt*)

This function is used to print the error information stored in the context to the standard output device. Parameter substitution is done so that recorded parameters are added to the output message text.

**Parameters**

***pctxt*** A pointer to a context structure.

#### 2.30.3.27 void rtxErrPrintElement (OSRTErrInfo * *pErrInfo*)

This function is used to print the error information stored in the error information element to the standard output device. Parameter substitution is done so that recorded parameters are added to the output message text.

**Parameters**

***pErrInfo*** A pointer to an error information element.

#### 2.30.3.28 int rtxErrReset (OSCTXT * *pctxt*)

This function is used to reset the error state recorded in the context to successful. It is used in the generated code in places where automatic error correction can be done.

**Parameters**

***pctxt*** A pointer to a context structure.

#### 2.30.3.29 int rtxErrResetLastErrors (OSCTXT * *pctxt*, int *errorsToReset*)

This function resets last 'errorsToReset' errors in the context.

**Parameters**

***pctxt*** A pointer to a context structure.

***errorsToReset*** A number of errors to reset, starting from the last record.

**Returns**

Completion status of operation:

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

#### 2.30.3.30 int rtxErrSetData (OSCTXT * *pctxt*, int *status*, const char * *module*, int *lineno*)

This function is used to record an error in the context structure. It is typically called via the `LOG_RTERR` macro in the generated code to trap error conditions.

**Parameters**

***pctxt*** A pointer to a context structure.

***status*** The error status code from rtxErrCodes.h

***module*** The C source file in which the error occurred.

***lineno*** The line number within the source file of the error.

**Returns**

The status code that was passed in.

#### 2.30.3.31 int rtxErrSetNewData (OSCTXT * *pctxt*, int *status*, const char * *module*, int *lineno*)

This function is used to record an error in the context structure. It is typically called via the LOG_RTERRNEW macro in the generated code to trap error conditions. It always allocates new error record.

**Parameters**

***pctxt*** A pointer to a context structure.

***status*** The error status code from rtxErrCodes.h

***module*** The C source file in which the error occurred.

***lineno*** The line number within the source file of the error.

**Returns**

The status code that was passed in.

## 2.31 Run-time error status codes.

### Defines

- #define RT_OK 0
- #define RT_OK_FRAG 2
- #define RTERR_BUFOVFLW -1
- #define RTERR_ENDOFBUF -2
- #define RTERR_IDNOTFOU -3
- #define RTERR_INVENUM -4
- #define RTERR_SETDUPL -5
- #define RTERR_SETMISRQ -6
- #define RTERR_NOTINSET -7
- #define RTERR_SEQOVFLW -8
- #define RTERR_INVOPT -9
- #define RTERR_NOMEM -10
- #define RTERR_INVHEXS -11
- #define RTERR_INVREAL -12
- #define RTERR_STROVFLW -13
- #define RTERR_BADVALUE -14
- #define RTERR_TOODEEP -15
- #define RTERR_CONSVIO -16
- #define RTERR_ENDOFFILE -17
- #define RTERR_INVUTF8 -18
- #define RTERR_OUTOFBND -19
- #define RTERR_INVPARAM -20
- #define RTERR_INVFORMAT -21
- #define RTERR_NOTINIT -22
- #define RTERR_TOOBIG -23
- #define RTERR_INVCHAR -24
- #define RTERR_XMLSTATE -25
- #define RTERR_XMLPARSE -26
- #define RTERR_SEQORDER -27
- #define RTERR_FILNOTFOU -28
- #define RTERR_READERR -29
- #define RTERR_WRITEERR -30
- #define RTERR_INVBASE64 -31
- #define RTERR_INVSOCKET -32
- #define RTERR_INVATTR -33
- #define RTERR_REGEXP -34
- #define RTERR_PATMATCH -35
- #define RTERR_ATTRMISRQ -36
- #define RTERR_HOSTNOTFOU -37
- #define RTERR_HTTPERR -38
- #define RTERR_SOAPERR -39
- #define RTERR_EXPIRED -40
- #define RTERR_UNEXPELEM -41
- #define RTERR_INVOCCUR -42
- #define RTERR_INVMSGBUF -43
- #define RTERR_DECELEMFAIL -44

- #define RTERR_DECATTRFAIL -45
- #define RTERR_STRMINUSE -46
- #define RTERR_NULLPTR -47
- #define RTERR_FAILED -48
- #define RTERR_ATTRFIXEDVAL -49
- #define RTERR_MULTIPLE -50
- #define RTERR_NOTYPEINFO -51
- #define RTERR_ADDRINUSE -52
- #define RTERR_CONNRESET -53
- #define RTERR_UNREACHABLE -54
- #define RTERR_NOCONN -55
- #define RTERR_CONNREFUSED -56
- #define RTERR_INVSOCKOPT -57
- #define RTERR_SOAPFAULT -58
- #define RTERR_MARKNOTSUP -59
- #define RTERR_NOTSUPP -60
- #define RTERR_UNBAL -61
- #define RTERR_EXPNAME -62
- #define RTERR_UNICODE -63
- #define RTERR_INVBOOL -64
- #define RTERR_INVNULL -65
- #define RTERR_INVLEN -66
- #define RTERR_UNKNOWNIE -67
- #define RTERR_NOTALIGNED -68
- #define RTERR_EXTRDATA -69
- #define RTERR_INVMAC -70
- #define RTERR_NOSECPARAMS -71

### 2.31.1 Detailed Description

This is a list of status codes that can be returned by the common run-time functions and generated code. In many cases, additional information and parameters for the different errors are stored in the context structure at the time the error in raised. This additional information can be output using the `rtxErrPrint` or `rtxErrLogUsingCB` run-time functions.

### 2.31.2 Define Documentation

#### 2.31.2.1 #define RT_OK 0

Normal completion status.

#### 2.31.2.2 #define RT_OK_FRAG 2

Message fragment return status. This is returned when a part of a message is successfully decoded. The application should continue to invoke the decode function until a zero status is returned.

#### 2.31.2.3 #define RTERR_ADDRINUSE -52

Address already in use. This status code is returned when an attempt is made to bind a socket to an address that is already in use.

#### 2.31.2.4 #define RTERR_ATTRFIXEDVAL -49

Attribute fixed value mismatch. The attribute contained a value that was different than the fixed value defined in the schema for the attribute.

#### 2.31.2.5 #define RTERR_ATTRMISRQ -36

Missing required attribute. This status code is returned by the decoder when an XML instance is missing a required attribute value as defined in the XML schema.

#### 2.31.2.6 #define RTERR_BADVALUE -14

Bad value. This status code is returned anywhere where an API is expecting a value to be within a certain range and it not within this range. An example is the encoding or decoding date values when the month or day value is not within the legal range (1-12 for month and 1 to whatever the max days is for a given month).

#### 2.31.2.7 #define RTERR_BUFOVFLW -1

Encode buffer overflow. This status code is returned when encoding into a static buffer and there is no space left for the item currently being encoded.

#### 2.31.2.8 #define RTERR_CONNREFUSED -56

Connection refused. This status code is returned when an attempt to communicate on an open socket is refused by the host.

#### 2.31.2.9 #define RTERR_CONNRESET -53

Remote connection was reset. This status code is returned when the connection is reset by the remote host (via explicit command or a crash.

#### 2.31.2.10 #define RTERR_CONSVIO -16

Constraint violation. This status code is returned when constraints defined the schema are violated. These include XSD facets such as min/maxOccurs, min/maxLength, patterns, etc.. Also ASN.1 value range, size, and permitted alphabet constraints.

#### 2.31.2.11 #define RTERR_DECATTRFAIL -45

Attribute decode failed. This status code and parameters are added to the failure status by the decoder to allow the specific attribute on which a decode error was detected to be identified.

#### 2.31.2.12 #define RTERR_DECELEMFAIL -44

Element decode failed. This status code and parameters are added to the failure status by the decoder to allow the specific element on which a decode error was detected to be identified.

#### 2.31.2.13 #define RTERR_ENDOFBUF -2

Unexpected end-of-buffer. This status code is returned when decoding and the decoder expects more data to be available but instead runs into the end of the decode buffer.

#### 2.31.2.14 #define RTERR_ENDOFFILE -17

Unexpected end-of-file error. This status code is returned when an unexpected end-of-file condition is detected on decode. It is similar to the ENDOFBUF error code described above except that in this case, decoding is being done from a file stream instead of from a memory buffer.

#### 2.31.2.15 #define RTERR_EXPIRED -40

Evaluation license expired. This error is returned from evaluation versions of the run-time library when the hard-coded evaluation period is expired.

#### 2.31.2.16 #define RTERR_EXPNAME -62

Expected name. This error code is returned when parsing a name/value pair and the name part is expected, but instead a value is encountered.

#### 2.31.2.17 #define RTERR_EXTRDATA -69

Extraneous data. This error is returned when after decoding is complete, additional undecoded data is still present in the message buffer.

#### 2.31.2.18 #define RTERR_FAILED -48

General failure. Low level call returned error.

#### 2.31.2.19 #define RTERR_FILNOTFOU -28

File not found. This status code is returned if an attempt is made to open a file input stream for decoding and the given file does not exist.

#### 2.31.2.20 #define RTERR_HOSTNOTFOU -37

Host name could not be resolved. This status code is returned from run-time socket functions when they are unable to connect to a given host computer.

#### 2.31.2.21 #define RTERR_HTTPERR -38

HTTP protocol error. This status code is returned by functions doing HTTP protocol operations such as SOAP functions. It is returned when a protocol error is detected. Details on the specific error can be obtained by calling rtxErrPrint.

#### 2.31.2.22 #define RTERR_IDNOTFOU -3

Expected identifier not found. This status is returned when the decoder is expecting a certain element to be present at the current position and instead something different is encountered. An example is decoding a sequence container type in which the declared elements are expected to be in the given order. If an element is encountered that is not the one expected, this error is raised.

#### 2.31.2.23 #define RTERR_INVATTR -33

Invalid attribute. This status code is returned by the decoder when an attribute is encountered in an XML instance that was not defined in the XML schema.

#### 2.31.2.24 #define RTERR_INVBASE64 -31

Invalid Base64 encoding. This status code is returned when an error is detected in decoding base64 data.

#### 2.31.2.25 #define RTERR_INVBOOL -64

Invalid boolean keyword. This error code is returned when an invalid boolean keyword in the format of the language being parsed is encountered. For example, 'true' or 'false' in all lowercase letters may be all that is acceptable.

#### 2.31.2.26 #define RTERR_INVCHAR -24

Invalid character. This status code is returned when a character is encountered that is not valid for a given data type. For example, if an integer value is being decoded and a non-numeric character is encountered, this error will be raised.

#### 2.31.2.27 #define RTERR_INVENUM -4

Invalid enumerated identifier. This status is returned when an enumerated value is being encoded or decoded and the given value is not in the set of values defined in the enumeration facet.

#### 2.31.2.28 #define RTERR_INVFORMAT -21

Invalid value format. This status code is returned when a value is received or passed into a function that is not in the expected format. For example, the time string parsing function expects a string in the form "nn:nn:nn" where n's are numbers. If not in this format, this error code is returned.

#### 2.31.2.29 #define RTERR_INVHEXS -11

Invalid hexadecimal string. This status code is returned when decoding a hexadecimal string value and a character is encountered in the string that is not in the valid hexadecimal character set ([0-9A-Fa-f] or whitespace).

#### 2.31.2.30 #define RTERR_INVLEN -66

Invalid length. This error code is returned when a length value is parsed that is not consistent with other lengths in a message. This typically happens when an inner length within a constructed type is larger than the outer length value.

#### 2.31.2.31 #define RTERR_INVMAC -70

Invalid Message Authentication Code. This error is returned when a given message's MAC is not the expected value.

#### 2.31.2.32 #define RTERR_INVMSGBUF -43

Invalid message buffer has been passed to decode or validate method. This status code is returned by decode or validate method when the used message buffer instance has type different from OSMessageBufferIF::XMLDecode.

#### 2.31.2.33 #define RTERR_INVNULL -65

Invalid null keyword. This error code is returned when an invalid null keyword in the format of the language being parsed is encountered. For example, 'null' in all lowercase letters may be all that is acceptable.

#### 2.31.2.34 #define RTERR_INVOCCUR -42

Invalid number of occurrences. This status code is returned by the decoder when an XML instance contains a number of occurrences of a repeating element that is outside the bounds (minOccurs/maxOccurs) defined for the element in the XML schema.

#### 2.31.2.35 #define RTERR_INVOPT -9

Invalid option in choice. This status code is returned when encoding or decoding an ASN.1 CHOICE or XSD xsd:choice construct. When encoding, it occurs when a value in the generated 't' member variable is outside the range of indexes of items in the content model group. It occurs on the decode side when an element is received that is not defined in the content model group.

#### 2.31.2.36 #define RTERR_INVPARAM -20

Invalid parameter passed to a function of method. This status code is returned by a function or method when it does an initial check on the values of parameters passed in. If a parameter is found to not have a value in the expected range, this error code is returned.

#### 2.31.2.37 #define RTERR_INVREAL -12

Invalid real number value. This status code is returned when decoding a numeric floating-point value and an invalid character is received (i.e. not numeric, decimal point, plus or minus sign, or exponent character).

#### 2.31.2.38 #define RTERR_INVSOCKET -32

Invalid socket. This status code is returned when an attempt is made to read or write from a scoket and the given socket handle is invalid. This may be the result of not having established a proper connection before trying to use the socket handle variable.

#### 2.31.2.39 #define RTERR_INVSOCKOPT -57

Invalid option. This status code is returned when an invalid option is passed to socket.

#### 2.31.2.40 #define RTERR_INVUTF8 -18

Invalid UTF-8 character encoding. This status code is returned by the decoder when an invalid sequence of bytes is detected in a UTF-8 character string.

#### 2.31.2.41 #define RTERR_MARKNOTSUP -59

This error is returned when an attempt is made to mark a stream position on a stream type that does not support it.

#### 2.31.2.42 #define RTERR_MULTIPLE -50

Multiple errors occurred during an encode or decode operation. See the error list within the context structure for a full list of all errors.

#### 2.31.2.43 #define RTERR_NOCONN -55

Not connected. This status code is returned when an operation is issued on an unconnected socket.

#### 2.31.2.44 #define RTERR_NOMEM -10

No dynamic memory available. This status code is returned when a dynamic memory allocation request is made and an insufficient amount of memory is available to satisfy the request.

#### 2.31.2.45 #define RTERR_NOSECPARAMS -71

No security parameters provided. This error is returned when a NAS message with either integrity protection or ciphering (or both) is received and the required security parameters needed to decrypt it or validate it have not been provided.

#### 2.31.2.46 #define RTERR_NOTALIGNED -68

Not aligned error. This is returned when an element is expected to start on a byte-aligned boundary and is found not to start on an unaligned boundary.

#### 2.31.2.47 #define RTERR_NOTINIT -22

Context not initialized. This status code is returned when the run-time context structure (OSCTXT) is attempted to be used without having been initialized. This can occur if rtxInitContext is not invoked to initialize a context variable before use in any other API call. It can also occur is there is a license violation (for example, evaluation license expired).

#### 2.31.2.48 #define RTERR_NOTINSET -7

Element not in set. This status code is returned when encoding or decoding an ASN.1 SET or XSD xsd:all construct. When encoding, it occurs when a value in the generated _order member variable is outside the range of indexes of items in the content model group. It occurs on the decode side when an element is received that is not defined in the content model group.

#### 2.31.2.49 #define RTERR_NOTSUPP -60

Feature is not supported. This status code is returned when a feature that is currently not supported is encountered. Support may be added in a future release.

#### 2.31.2.50 #define RTERR_NOTYPEINFO -51

This error is returned when decoding a derived type definition and no information exists as to what type of data is in the element content. When decoding XML, this normally means that an xsi:type attribute was not found identifying the type of content.

#### 2.31.2.51 #define RTERR_NULLPTR -47

Null pointer. This status code is returned when a null pointer is encountered in a place where it is expected that the pointer value is to be set.

#### 2.31.2.52 #define RTERR_OUTOFBND -19

Array index out-of-bounds. This status code is returned when an attempt is made to add something to an array and the given index is outside the defined bounds of the array.

#### 2.31.2.53 #define RTERR_PATMATCH -35

Pattern match error. This status code is returned by the decoder when a value in an XML instance does not match the pattern facet defined in the XML schema. It can also be returned by numeric encode functions that cannot format a numeric value to match the pattern specified for that value.

#### 2.31.2.54 #define RTERR_READERR -29

Read error. This status code if returned if a read I/O error is encountered when reading from an input stream associated with a physical device such as a file or socket.

#### 2.31.2.55 #define RTERR_REGEXP -34

Invalid regular expression. This status code is returned when a syntax error is detected in a regular expression value. Details of the syntax error can be obtained by invoking rtxErrPrint to print the details of the error contained within the context variable.

#### 2.31.2.56 #define RTERR_SEQORDER -27

Sequence order error. This status code is returned when decoding an ASN.1 SEQUENCE or XSD xsd:sequence construct. It is raised if the elements were received in an order different than that specified in the content model group definition.

#### 2.31.2.57 #define RTERR_SEQOVFLW -8

Sequence overflow. This status code is returned when decoding a repeating element (ASN.1 SEQUENCE OF or XSD element with min/maxOccurs $> 1$) and more instances of the element are received than were defined in the constraint.

#### 2.31.2.58 #define RTERR_SETDUPL -5

Duplicate element in set. This status code is returned when decoding an ASN.1 SET or XSD xsd:all construct. It is raised if a given element defined in the content model group occurs multiple times in the instance being decoded.

#### 2.31.2.59 #define RTERR_SETMISRQ -6

Missing required element in set. This status code is returned when decoding an ASN.1 SET or XSD xsd:all construct and all required elements in the content model group are not found to be present in the instance being decoded.

#### 2.31.2.60 #define RTERR_SOAPERR -39

SOAP error. This status code when an error is detected when trying to execute a SOAP operation.

#### 2.31.2.61 #define RTERR_SOAPFAULT -58

This error is returned when decoded SOAP envelope is fault message

#### 2.31.2.62 #define RTERR_STRMINUSE -46

Stream in-use. This status code is returned by stream functions when an attempt is made to initialize a stream or create a reader or writer when an existing stream is open in the context. The existing stream must first be closed before initializaing a stream for a new operation.

#### 2.31.2.63 #define RTERR_STROVFLW -13

String overflow. This status code is returned when a fixed-sized field is being decoded as specified by a size constraint and the item contains more characters or bytes then this amount. It can occur when a run-time function is called with a fixed-sixed static buffer and whatever operation is being done causes the bounds of this buffer to be exceeded.

#### 2.31.2.64 #define RTERR_TOOBIG -23

Value will not fit in target variable. This status is returned by the decoder when a target variable is not large enough to hold a a decoded value. A typical case is an integer value that is too large to fit in the standard C integer type (typically a 32-bit value) on a given platform. If this occurs, it is usually necessary to use a configuration file setting to force the compiler to use a different data type for the item. For example, for integer, the <isBigInteger/> setting can be used to force use of a big integer type.

#### 2.31.2.65 #define RTERR_TOODEEP -15

Nesting level too deep. This status code is returned when a preconfigured maximum nesting level for elements within a content model group is exceeded.

#### 2.31.2.66 #define RTERR_UNBAL -61

Unbalanced structure. This error code is returned when parsing formatted text such as XML or JSON and a block is not properly terminated. For JSON, this occurs when a '{' or '[' character does not a corresponding '}' or ']' respectively. For XML, it occurs when an open element does not have a corresponding end element.

#### 2.31.2.67 #define RTERR_UNEXPELEM -41

Unexpected element encountered. This status code is returned when an element is encountered in a position where something else (for example, an attribute) was expected.

#### 2.31.2.68 #define RTERR_UNICODE -63

Invalid Unicode sequence. The sequence of characters received did not comprise a valid unicode character.

#### 2.31.2.69 #define RTERR_UNKNOWNIE -67

Unknown information element. This error code is returned when an unknown information element or extension is received and the protocol specification indicates the element must be understood.

#### 2.31.2.70 #define RTERR_UNREACHABLE -54

Network failure. This status code is returned when the network or host is down or otherwise unreachable.

#### 2.31.2.71 #define RTERR_WRITEERR -30

Write error. This status code if returned if a write I/O error is encountered when attempting to output data to an output stream associated with a physical device such as a file or socket.

#### 2.31.2.72 #define RTERR_XMLPARSE -26

XML parser error. This status code in returned when the underlying XML parser application (by default, this is Expat) returns an error code. The parser error code or text is returned as a parameter in the errInfo structure within the context structure.

#### 2.31.2.73 #define RTERR_XMLSTATE -25

XML state error. This status code is returned when the XML parser is not in the correct state to do a certain operation.

# Chapter 3

# Class Documentation

## 3.1 _OSRTSList Struct Reference

`#include <rtxSList.h>`

### Public Attributes

- OSUINT32 count
- OSRTSListNode * head
- OSRTSListNode * tail
- struct OSCTXT * **pctxt**

### 3.1.1 Detailed Description

This is the main list structure. It contains a count of the number of elements in the list and pointers to the list head and tail elements.

### 3.1.2 Member Data Documentation

#### 3.1.2.1 OSUINT32 _OSRTSList::count

Count of items in the list.

#### 3.1.2.2 OSRTSListNode* _OSRTSList::head

Pointer to first entry in list.

#### 3.1.2.3 OSRTSListNode* _OSRTSList::tail

Pointer to last entry in list.

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

- rtxSList.h

## 3.2 _OSRTSListNode Struct Reference

`#include <rtxSList.h>`

### Public Attributes

- void * data
- struct _OSRTSListNode * next

### 3.2.1 Detailed Description

This structure is used to hold a single data item within the list. It contains a void pointer to point at any type of data item and forward pointer to the next entry in the list.

### 3.2.2 Member Data Documentation

#### 3.2.2.1 void* _OSRTSListNode::data

Pointer to list data item.

#### 3.2.2.2 struct _OSRTSListNode* _OSRTSListNode::next

Pointer to next node in list.

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

- rtxSList.h

## 3.3 _OSRTStack Struct Reference

`#include <rtxStack.h>`

### Public Attributes

- OSCTXT * **pctxt**
- OSRTDList **dlist**

### 3.3.1 Detailed Description

This is the main stack structure. It uses a linked list structure.

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

- rtxStack.h

## 3.4 Asn116BitCharSet Struct Reference

`#include <asn1type.h>`

### Public Attributes

- Asn116BitCharString charSet
- OSUINT16 firstChar
- OSUINT16 **lastChar**
- unsigned unalignedBits
- unsigned alignedBits

### 3.4.1 Detailed Description

Describes an ASN.1 character set whose characters are 16-bits wide instead of 8-bits wide.

### 3.4.2 Member Data Documentation

#### 3.4.2.1 unsigned Asn116BitCharSet::alignedBits

The number of bits required by this set in aligned applications.

#### 3.4.2.2 Asn116BitCharString Asn116BitCharSet::charSet

A character string describing the whole character set.

#### 3.4.2.3 OSUINT16 Asn116BitCharSet::firstChar

16-bit integers describing the first and last characters in the set.

#### 3.4.2.4 unsigned Asn116BitCharSet::unalignedBits

The number of bits required by this set in unaligned applications.

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

- asn1type.h

## 3.5 Asn116BitCharString Struct Reference

`#include <asn1type.h>`

### Public Attributes

- OSUINT32 **nchars**
- OSUNICHAR * **data**

### 3.5.1 Detailed Description

A structure that holds a 16-bit ASN.1 character string. This contains the number of characters and a pointer to 16-bit characters that hold the string data.

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

- asn1type.h

## 3.6 Asn132BitCharSet Struct Reference

`#include <asn1type.h>`

### Public Attributes

- Asn132BitCharString charSet
- OSUINT32 firstChar
- OSUINT32 **lastChar**
- unsigned unalignedBits
- unsigned alignedBits

### 3.6.1 Detailed Description

Describes an ASN.1 character set whose characters are 32-bits wide instead of 8-bits wide.

### 3.6.2 Member Data Documentation

#### 3.6.2.1 unsigned Asn132BitCharSet::alignedBits

The number of bits required by this set in aligned applications.

#### 3.6.2.2 Asn132BitCharString Asn132BitCharSet::charSet

A character string describing the whole character set.

#### 3.6.2.3 OSUINT32 Asn132BitCharSet::firstChar

32-bit integers describing the first and last characters in the set.

#### 3.6.2.4 unsigned Asn132BitCharSet::unalignedBits

The number of bits required by this set in unaligned applications.

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

- asn1type.h

## 3.7 Asn132BitCharString Struct Reference

`#include <asn1type.h>`

### Public Attributes

- OSUINT32 **nchars**
- OS32BITCHAR * **data**

### 3.7.1 Detailed Description

A structure that holds a 32-bit ASN.1 character string. This contains the number of characters and a pointer to 32-bit characters that hold the string data.

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

- asn1type.h

## 3.8 ASN1BigInt Struct Reference

`#include <asn1type.h>`

### Public Attributes

- size_t numocts
- OSOCTET * mag
- int sign
- size_t allocated
- OSBOOL dynamic

### 3.8.1 Detailed Description

A structure used to define an ASN.1 big integer. This structure is rarely, if ever, used by client code, and will instead be used by generated code to facilitate encoding and decoding integer values that cannot fit in normal C/C++ integer types.

### 3.8.2 Member Data Documentation

#### 3.8.2.1 size_t ASN1BigInt::allocated

The number of octets allocated for the magnitude.

#### 3.8.2.2 OSBOOL ASN1BigInt::dynamic

A flag that tells whether the buffer is dynamically allocated.

#### 3.8.2.3 OSOCTET* ASN1BigInt::mag

The magnitude value.

#### 3.8.2.4 size_t ASN1BigInt::numocts

The number of octets used in the magnitude.

#### 3.8.2.5 int ASN1BigInt::sign

The sign: either -1, 0, or 1.

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

- asn1type.h

## 3.9 ASN1BitStr32 Struct Reference

`#include <asn1type.h>`

### Public Attributes

- OSUINT32 **numbits**
- OSOCTET **data** [4]

### 3.9.1 Detailed Description

fixed-size bit string that can hold up to 32 bits

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

- asn1type.h

## 3.10 ASN1CCB Struct Reference

`#include <asn1type.h>`

### Public Attributes

- OSOCTET * ptr
- long len
- int seqx
- OSUINT16 mask [ASN1_K_CCBMaskSize]
- size_t bytes
- int stat

### 3.10.1 Detailed Description

The ASN.1 Context Control Block.

This structure is used to help facilitate decoding in SEQUENCE and SET structures. It is rarely (if ever) used directly by client code, and will instead by used in generated code.

### 3.10.2 Member Data Documentation

#### 3.10.2.1 size_t ASN1CCB::bytes

The bytes processed by the block, used for streaming.

#### 3.10.2.2 long ASN1CCB::len

The constructor length.

#### 3.10.2.3 OSUINT16 ASN1CCB::mask[ASN1_K_CCBMaskSize]

The set mask value.

#### 3.10.2.4 OSOCTET* ASN1CCB::ptr

The contructor pointer.

#### 3.10.2.5 int ASN1CCB::seqx

The sequence element index.

#### 3.10.2.6 int ASN1CCB::stat

The status, as returned by BS_CHKEND.

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

- asn1type.h

## 3.11 Asn1CharArray Struct Reference

`#include <asn1type.h>`

### Public Attributes

- int **nchars**
- char **data** [255]

### 3.11.1 Detailed Description

A generic character array. The array data holds up to 255 characters, with the actual number of characters being provided by the nchars member.

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

- asn1type.h

## 3.12 Asn1CharSet Struct Reference

`#include <asn1type.h>`

### Public Attributes

- Asn1CharArray charSet
- const char * canonicalSet
- int canonicalSetSize
- unsigned canonicalSetBits
- unsigned charSetUnalignedBits
- unsigned charSetAlignedBits

### 3.12.1 Detailed Description

Describes an ASN.1 character set, use primarily for PER encodings whose alphabet constraints can be used to encode more compact representations of their strings.

### 3.12.2 Member Data Documentation

#### 3.12.2.1 const char* Asn1CharSet::canonicalSet

A character string describing the canonical set of characters.

#### 3.12.2.2 unsigned Asn1CharSet::canonicalSetBits

The number of bits taken up by the canonical set.

#### 3.12.2.3 int Asn1CharSet::canonicalSetSize

The size of the canonical character set.

#### 3.12.2.4 Asn1CharArray Asn1CharSet::charSet

The array of characters that comprise this particular character set; at most this takes up 255 characters.

#### 3.12.2.5 unsigned Asn1CharSet::charSetAlignedBits

The number of bits reuqired in aligned applications for this character set.

#### 3.12.2.6 unsigned Asn1CharSet::charSetUnalignedBits

The number of bits required in unaligned applications for this character set.

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

- asn1type.h

## 3.13 ASN1DynBitStr Struct Reference

`#include <asn1type.h>`

### Public Attributes

- OSUINT32 **numbits**
- const OSOCTET * **data**

### 3.13.1 Detailed Description

generic bit string structure (dynamic)

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

- asn1type.h

## 3.14 Asn1Object Struct Reference

`#include <asn1type.h>`

### Public Attributes

- ASN1OpenType **encoded**
- void * **decoded**
- OSINT32 **index**

### 3.14.1 Detailed Description

A generic table constraint value holder. This structure contains the encoded open type value, a pointer-to-void that holds the decoded content, and the integer-valued index of the table constraint value.

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

- asn1type.h

## 3.15 ASN1OBJID Struct Reference

`#include <asn1type.h>`

### Public Attributes

- OSUINT32 numids
- OSUINT32 subid [ASN_K_MAXSUBIDS]

### 3.15.1 Detailed Description

This structure describes an object identifier with 32-bit arcs.

### 3.15.2 Member Data Documentation

#### 3.15.2.1 OSUINT32 ASN1OBJID::numids

The number of sub-identifiers in the OID.

#### 3.15.2.2 OSUINT32 ASN1OBJID::subid[ASN_K_MAXSUBIDS]

An array of sub-identifiers.

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

- asn1type.h

## 3.16 ASN1OctStr Struct Reference

`#include <asn1type.h>`

### Public Attributes

- OSUINT32 **numocts**
- OSOCTET **data** [1]

### 3.16.1 Detailed Description

A generic octet string structure. This contains the number of octest and a data array with a size of one.

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

- asn1type.h

## 3.17 ASN1OID64 Struct Reference

`#include <asn1type.h>`

### Public Attributes

- OSUINT32 numids
- OSUINT64 subid [ASN_K_MAXSUBIDS]

### 3.17.1 Detailed Description

This structure describes an object identifier with 64-bit arcs.

### 3.17.2 Member Data Documentation

#### 3.17.2.1 OSUINT32 ASN1OID64::numids

The number of sub-identifiers in the OID.

#### 3.17.2.2 OSUINT64 ASN1OID64::subid[ASN_K_MAXSUBIDS]

An array of sub-identifiers.

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

- asn1type.h

## 3.18 ASN1OpenType Struct Reference

`#include <asn1type.h>`

### Public Attributes

- OSUINT32 **numocts**
- const OSOCTET * **data**

### 3.18.1 Detailed Description

A generic open type structure. This structure contains a number of octets and the data that compose the open type information.

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

- asn1type.h

## 3.19 ASN1SeqOf Struct Reference

`#include <asn1type.h>`

### Public Attributes

- OSUINT32 n
- void * elem

### 3.19.1 Detailed Description

A generic SEQUENCE OF structure that contains a number of elements and a pointer-to-void that contains the contents.

### 3.19.2 Member Data Documentation

#### 3.19.2.1 void* ASN1SeqOf::elem

The pointer-to-void that contains the elements.

#### 3.19.2.2 OSUINT32 ASN1SeqOf::n

The number of elements.

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

- asn1type.h

## 3.20 ASN1SeqOfOctStr Struct Reference

`#include <asn1type.h>`

### Public Attributes

- OSUINT32 n
- OSDynOctStr * elem

### 3.20.1 Detailed Description

A generic SEQUENCE OF dynamic OCTET STRING.

This structure is used to hold a sequence of octet strings. The elements are pointers-to-OSDynOctStr.

### 3.20.2 Member Data Documentation

#### 3.20.2.1 OSDynOctStr* ASN1SeqOfOctStr::elem

A pointer-to-OSDynOctStr that contains the elements.

#### 3.20.2.2 OSUINT32 ASN1SeqOfOctStr::n

The number of elements.

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

- asn1type.h

## 3.21 DirBufDesc Struct Reference

### Public Attributes

- OSCTXT * **pctxt**
- OSRTSTREAM * **pUnderStream**
- size_t **savedIndex**

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

- rtxStreamMemory.h

## 3.22 OSBigInt Struct Reference

### Public Attributes

- OSSIZE **numocts**
- OSOCTET * **mag**
- int **sign**
- OSSIZE **allocated**
- OSBOOL **dynamic**

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

- rtxBigInt.h

## 3.23 OSCTXT Struct Reference

`#include <rtxContext.h>`

### Public Attributes

- void * **pMemHeap**
- OSRTBuffer **buffer**
- OSRTBufSave **savedInfo**
- OSRTErrInfoList **errInfo**
- OSUINT32 **initCode**
- OSRTFLAGS **flags**
- OSOCTET **level**
- OSOCTET **state**
- OSOCTET **diagLevel**
- OSOCTET **lastChar**
- struct OSRTSTREAM * **pStream**
- struct OSRTPrintStream * **pPrintStrm**
- OSRTDList **elemNameStack**
- OSRTDList **regExpCache**
- const OSOCTET * **key**
- OSSIZE **keylen**
- OSVoidPtr **pXMLInfo**
- OSVoidPtr **pASN1Info**
- OSVoidPtr **pEXIInfo**
- OSVoidPtr **pUserData**
- OSVoidPtr **pGlobalData**
- struct OS3GPPSecParams * **p3gppSec**
- OSFreeCtxtGlobalPtr **gblFreeFunc**
- OSVoidPtr **ssl**
- struct OSRTDiagBitFieldList * **pBitFldList**
- OSUINT16 **indent**
- OSUINT16 **version**

### 3.23.1 Detailed Description

Run-time context structure

This structure is a container structure that holds all working variables involved in encoding or decoding a message.

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

- rtxContext.h

## 3.24 OSRTBuffer Struct Reference

`#include <rtxContext.h>`

### Public Attributes

- OSOCTET * **data**
- OSSIZE **byteIndex**
- OSSIZE **size**
- OSINT16 **bitOffset**
- OSBOOL **dynamic**
- OSBOOL **aligned**

### 3.24.1 Detailed Description

Run-time message buffer structure

This structure holds encoded message data. For an encode operation, it is where the message being built is stored. For decode, it holds a copy of the message that is being decoded.

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

- rtxContext.h

## 3.25 OSRTBufSave Struct Reference

`#include <rtxContext.h>`

### Public Attributes

- OSSIZE **byteIndex**
- OSINT16 **bitOffset**
- OSRTFLAGS **flags**

### 3.25.1 Detailed Description

Structure to save the current message buffer state

This structure is used to save the current state of the buffer.

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

- rtxContext.h

## 3.26 OSRTDList Struct Reference

`#include <rtxDList.h>`

### Public Attributes

- OSSIZE count
- OSRTDListNode * head
- OSRTDListNode * tail

### 3.26.1 Detailed Description

This is the main list structure. It contains a count of the number of elements in the list and pointers to the list head and tail elements.

### 3.26.2 Member Data Documentation

#### 3.26.2.1 OSSIZE OSRTDList::count

Count of items in the list.

#### 3.26.2.2 OSRTDListNode* OSRTDList::head

Pointer to first entry in list.

#### 3.26.2.3 OSRTDListNode* OSRTDList::tail

Pointer to last entry in list.

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

- rtxDList.h

## 3.27 OSRTDListBuf Struct Reference

### Public Attributes

- OSSIZE **n**
- OSSIZE **nMax**
- OSSIZE **nAll**
- OSSIZE **firstSegSz**
- OSSIZE **elemSize**
- OSRTDList **tmplist**
- void ** **dataArray**

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

- rtxDList.h

## 3.28 OSRTDListNode Struct Reference

`#include <rtxDList.h>`

### Public Attributes

- void * data
- struct OSRTDListNode * next
- struct OSRTDListNode * prev

### 3.28.1 Detailed Description

This structure is used to hold a single data item within the list. It contains a void pointer to point at any type of data item and forward and backward pointers to the next and previous entries in the list.

### 3.28.2 Member Data Documentation

#### 3.28.2.1 void* OSRTDListNode::data

Pointer to list data item.

#### 3.28.2.2 struct OSRTDListNode* OSRTDListNode::next

Pointer to next node in list.

#### 3.28.2.3 struct OSRTDListNode* OSRTDListNode::prev

Pointer to previous node in list.

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

- rtxDList.h

## 3.29 OSRTDListUTF8StrNode Struct Reference

### Public Attributes

- OSRTDListNode **node**
- OSUTF8CHAR **utf8chars** [1]

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

- rtxDList.h

## 3.30 OSRTErrInfo Struct Reference

`#include <rtxContext.h>`

### Public Attributes

- OSRTErrLocn **stack** [OSRTERRSTKSIZ]
- OSINT16 **status**
- OSUINT8 **stkx**
- OSUINT8 **parmcnt**
- OSUTF8CHAR * **parms** [OSRTMAXERRPRM]
- OSUTF8CHAR * **elemName**

### 3.30.1 Detailed Description

Run-time error information structure

This structure is a container structure that holds information on run-time errors. The stack variable holds the trace stack information that shows where the error occurred in the source code. The parms variable holds error parameters that are substituted into the message that is returned to the user.

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

- rtxContext.h

## 3.31 OSRTErrInfoList Struct Reference

### Public Attributes

- OSRTDList **list**
- OSRTErrInfo **reserved**
- OSRTDListNode **reservedNode**

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

- rtxContext.h

## 3.32 OSRTErrLocn Struct Reference

`#include <rtxContext.h>`

### Public Attributes

- const OSUTF8CHAR * **module**
- OSINT32 **lineno**

### 3.32.1 Detailed Description

Run-time error location structure

This structure is a container structure that holds information on the location within a C source file where a run-time error occurred.

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

- rtxContext.h

## 3.33 OSRTMEMBUF Struct Reference

### Public Attributes

- OSCTXT * **pctxt**
- OSSIZE **segsize**
- OSSIZE **startidx**
- OSSIZE **usedcnt**
- OSSIZE **bufsize**
- OSSIZE **bitOffset**
- OSUINT32 **userState**
- OSOCTET * **buffer**
- OSBOOL **isDynamic**
- OSBOOL **isExpandable**
- OSBOOL **useSysMem**

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

- rtxMemBuf.h

## 3.34 OSRTPrintStream Struct Reference

`#include <rtxPrintStream.h>`

### Public Attributes

- rtxPrintCallback **pfPrintFunc**
- void * **pPrntStrmInfo**

### 3.34.1 Detailed Description

Structure to hold information about a global PrintStream.

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

- rtxPrintStream.h

## 3.35 OSRTSTREAM Struct Reference

`#include <rtxStream.h>`

### Public Attributes

- OSRTStreamReadProc read
- OSRTStreamBlockingReadProc blockingRead
- OSRTStreamWriteProc write
- OSRTStreamFlushProc flush
- OSRTStreamCloseProc close
- OSRTStreamSkipProc skip
- OSRTStreamMarkProc mark
- OSRTStreamResetProc reset
- OSRTStreamGetPosProc getPos
- OSRTStreamSetPosProc setPos
- void * extra
- size_t bufsize
- size_t readAheadLimit
- size_t bytesProcessed
- size_t markedBytesProcessed
- size_t ioBytes
- size_t nextMarkOffset
- size_t segsize
- OSUINT32 id
- OSRTMEMBUF * pCaptureBuf
- OSUINT16 flags

### 3.35.1 Detailed Description

The stream control block. A user may implement a customized stream by defining read, skip, close functions for input streams and write, flush, close for output streams.

### 3.35.2 Member Data Documentation

#### 3.35.2.1 OSRTStreamBlockingReadProc OSRTSTREAM::blockingRead

pointer to blockingRead function

#### 3.35.2.2 size_t OSRTSTREAM::bufsize

physical size of pctxt->buffer.data buffer

#### 3.35.2.3 size_t OSRTSTREAM::bytesProcessed

the number of bytes processed by the application program

#### 3.35.2.4 OSRTStreamCloseProc OSRTSTREAM::close

pointer to close function

#### 3.35.2.5 void* OSRTSTREAM::extra

pointer to stream-specific data

#### 3.35.2.6 OSUINT16 OSRTSTREAM::flags

flags (see OSRTSTRMF_* macros

#### 3.35.2.7 OSRTStreamFlushProc OSRTSTREAM::flush

pointer to flush function

#### 3.35.2.8 OSRTStreamGetPosProc OSRTSTREAM::getPos

pointer to getPos function

#### 3.35.2.9 OSUINT32 OSRTSTREAM::id

id of stream (see OSRTSTRMID_* macros)

#### 3.35.2.10 size_t OSRTSTREAM::ioBytes

the actual number of bytes read from or written to the stream

#### 3.35.2.11 OSRTStreamMarkProc OSRTSTREAM::mark

pointer to mark function

#### 3.35.2.12 size_t OSRTSTREAM::markedBytesProcessed

the marked number of bytes already processed

#### 3.35.2.13 size_t OSRTSTREAM::nextMarkOffset

offset of next appropriate mark position

#### 3.35.2.14 OSRTMEMBUF* OSRTSTREAM::pCaptureBuf

Buffer into which data read from stream can be captured for debugging purposes.

#### 3.35.2.15 OSRTStreamReadProc OSRTSTREAM::read

pointer to read function

#### 3.35.2.16 size_t OSRTSTREAM::readAheadLimit

read ahead limit (used by rtxStreamMark/rtxStreamReset

#### 3.35.2.17 OSRTStreamResetProc OSRTSTREAM::reset

pointer to reset function

#### 3.35.2.18 size_t OSRTSTREAM::segsize

size of decoded segment

#### 3.35.2.19 OSRTStreamSetPosProc OSRTSTREAM::setPos

pointer to setPos function

#### 3.35.2.20 OSRTStreamSkipProc OSRTSTREAM::skip

pointer to skip function

#### 3.35.2.21 OSRTStreamWriteProc OSRTSTREAM::write

pointer to write function

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

- rtxStream.h

# Chapter 4

# File Documentation

## 4.1 asn1type.h File Reference

```
#include <limits.h>
#include <stdio.h>
#include <string.h>
#include <stddef.h>
#include <setjmp.h>
#include <stdlib.h>
#include <time.h>
#include <wchar.h>
#include "rtxsrc/rtxExternDefs.h"
#include "rtxsrc/rtxSList.h"
#include "rtxsrc/rtxStack.h"
#include "rtxsrc/rtxUTF8.h"
#include "rtsrc/asn1tag.h"
#include "rtsrc/asn1ErrCodes.h"
#include "rtsrc/asn1version.h"
#include "rtsrc/rtExternDefs.h"
#include <float.h>
#include "rtxsrc/rtxBitString.h"
#include "rtsrc/rtContext.h"
#include "rtxsrc/rtxCommonDefs.h"
#include "rtxsrc/rtxError.h"
#include "rtxsrc/rtxMemory.h"
```

## Classes

- struct ASN1OBJID
- struct ASN1OID64
- struct ASN1OctStr
- struct ASN1DynBitStr
- struct ASN1BitStr32
- struct ASN1SeqOf
- struct ASN1SeqOfOctStr
- struct ASN1OpenType
- struct Asn1Object
- struct Asn116BitCharString
- struct Asn132BitCharString
- struct Asn1CharArray
- struct Asn1CharSet
- struct Asn116BitCharSet
- struct Asn132BitCharSet
- struct ASN1BigInt
- struct ASN1CCB

## Defines

- #define XM_SEEK 0x01
- #define XM_ADVANCE 0x02
- #define XM_DYNAMIC 0x04
- #define XM_SKIP 0x08
- #define XM_OPTIONAL 0x10
- #define ASN_K_MAXDEPTH 32
- #define ASN_K_MAXENUM 100
- #define ASN_K_MAXERRP 5
- #define ASN_K_MAXERRSTK 8
- #define ASN_K_ENCBUFSIZ 16∗1024
- #define ASN_K_MEMBUFSEG 1024
- #define OSRTINDENTSPACES 3
- #define ASN1_K_PLUS_INFINITY 0x40
- #define ASN1_K_MINUS_INFINITY 0x41
- #define **REAL_BINARY** 0x80
- #define **REAL_SIGN** 0x40
- #define **REAL_EXPLEN_MASK** 0x03
- #define **REAL_EXPLEN_1** 0x00
- #define **REAL_EXPLEN_2** 0x01
- #define **REAL_EXPLEN_3** 0x02
- #define **REAL_EXPLEN_LONG** 0x03
- #define **REAL_FACTOR_MASK** 0x0c
- #define **REAL_BASE_MASK** 0x30
- #define **REAL_BASE_2** 0x00
- #define **REAL_BASE_8** 0x10
- #define **REAL_BASE_16** 0x20
- #define **REAL_ISO6093_MASK** 0x3F
- #define **ASN1REALMAX** (OSREAL)DBL_MAX

- #define **ASN1REALMIN** (OSREAL)-DBL_MAX
- #define **ASN_K_MAXSUBIDS** 128
- #define ASN1DynOctStr OSDynOctStr
- #define OSSETBIT(bitStr, bitIndex) rtxSetBit (bitStr.data, bitStr.numbits, bitIndex)
- #define OSSETBITP(pBitStr, bitIndex) rtxSetBit ((pBitStr)->data, (pBitStr)->numbits, bitIndex)
- #define OSCLEARBIT(bitStr, bitIndex) rtxClearBit (bitStr.data, bitStr.numbits, bitIndex)
- #define OSCLEARBITP(pBitStr, bitIndex) rtxClearBit ((pBitStr)->data, (pBitStr)->numbits, bitIndex)
- #define OSTESTBIT(bitStr, bitIndex) rtxTestBit (bitStr.data, bitStr.numbits, bitIndex)
- #define OSTESTBITP(pBitStr, bitIndex) rtxTestBit ((pBitStr)->data, (pBitStr)->numbits, bitIndex)
- #define ASN1_K_CCBMaskSize 32
- #define ASN1_K_NumBitsPerMask 16
- #define ASN1_K_MaxSetElements (ASN1_K_CCBMaskSize*ASN1_K_NumBitsPerMask)
- #define **ASN1NUMOCTS**(nbits) ((nbits>0)?(((nbits-1)/8)+1):0)

  - #define ALLOC_ASN1ARRAY(pctxt, pseqof, type)
  - #define ALLOC_ASN1ARRAY1(pctxt, pseqof, type)

## Typedefs

- typedef void * **ASN1ANY**
- typedef Asn1Object **ASN1Object**
- typedef OSUNICHAR **ASN116BITCHAR**
- 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 * **ASN1VideotexString**
- typedef const char * **ASN1VisibleString**
- typedef const OSUTF8CHAR * **ASN1UTF8String**
- typedef Asn116BitCharString **ASN1BMPString**
- typedef Asn132BitCharString **ASN1UniversalString**
- typedef struct ASN1BigInt ASN1BigInt
- typedef int(* ASN1DumpCbFunc )(const char *text_p, void *cbArg_p)

## Enumerations

- enum ASN1StrType { **ASN1HEX**, **ASN1BIN**, **ASN1CHR** }
- enum ASN1ActionType { **ASN1ENCODE**, **ASN1DECODE** }

## Functions

- void rtSetOID (ASN1OBJID *ptarget, ASN1OBJID *psource)
- void rtAddOID (ASN1OBJID *ptarget, ASN1OBJID *psource)
- OSBOOL rtOIDsEqual (const ASN1OBJID *pOID1, const ASN1OBJID *pOID2)
- int rtOIDParseDottedNumberString (const char *oidstr, OSSIZE oidstrlen, ASN1OBJID *pvalue)
- OSBOOL rtOIDIsValid (const ASN1OBJID *pvalue)
- int rtMakeGeneralizedTime (OSCTXT *pctxt, const OSNumDateTime *dateTime, char **outdata, size_t outdataSize)
- int rtMakeUTCTime (OSCTXT *pctxt, const OSNumDateTime *dateTime, char **outdata, size_t outdataSize)
- int rtParseGeneralizedTime (OSCTXT *pctxt, const char *value, OSNumDateTime *dateTime)
- int rtParseUTCTime (OSCTXT *pctxt, const char *value, OSNumDateTime *dateTime)
- void normalizeTimeZone (OSNumDateTime *pvalue)
- int rtValidateStr (ASN1TAG tag, const char *pdata)
- int rtValidateChars (ASN1TAG tag, const char *pdata, size_t nchars)
- const char * rtBMPToCString (ASN1BMPString *pBMPString, char *cstring, OSUINT32 cstrsize)
- const char * rtBMPToNewCString (ASN1BMPString *pBMPString)
- const char * rtBMPToNewCStringEx (OSCTXT *pctxt, ASN1BMPString *pBMPString)
- ASN1BMPString * rtCToBMPString (OSCTXT *pctxt, const char *cstring, ASN1BMPString *pBMPString, Asn116BitCharSet *pCharSet)
- OSBOOL rtIsIn16BitCharSet (OSUNICHAR ch, Asn116BitCharSet *pCharSet)
- const char * rtUCSToCString (ASN1UniversalString *pUCSString, char *cstring, OSUINT32 cstrsize)
- const char * rtUCSToNewCString (ASN1UniversalString *pUCSString)
- const char * rtUCSToNewCStringEx (OSCTXT *pctxt, ASN1UniversalString *pUCSString)
- ASN1UniversalString * rtCToUCSString (OSCTXT *pctxt, const char *cstring, ASN1UniversalString *pUCSString, Asn132BitCharSet *pCharSet)
- OSBOOL rtIsIn32BitCharSet (OS32BITCHAR ch, Asn132BitCharSet *pCharSet)
- wchar_t * rtUCSToWCSString (ASN1UniversalString *pUCSString, wchar_t *wcstring, OSUINT32 wcstrsize)
- ASN1UniversalString * rtWCSToUCSString (OSCTXT *pctxt, wchar_t *wcstring, ASN1UniversalString *pUCSString, Asn132BitCharSet *pCharSet)
- int rtUnivStrToUTF8 (OSCTXT *pctxt, const ASN1UniversalString *pUnivStr, OSOCTET *outbuf, size_t outbufsiz)
- int rtUTF8StrToASN1DynBitStr (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, ASN1DynBitStr *pvalue)
- int rtUTF8StrnToASN1DynBitStr (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, size_t nbytes, ASN1DynBitStr *pvalue)

### 4.1.1 Detailed Description

Common ASN.1 runtime constants, data structure definitions, and run-time functions to support the BER/DER/PER/XER as defined in the ITU-T standards.

## 4.2 rtBCD.h File Reference

```
#include "rtsrc/asn1type.h"
#include "rtxsrc/rtxTBCD.h"
```

### Defines

- #define rtQ825TBCDToString(numocts, data, buffer, bufsiz) rtxQ825TBCDToString(numocts, data, buffer, bufsiz)
- #define rtDecQ825TBCDString(pctxt, numocts, buffer, bufsiz) rtxDecQ825TBCDString(pctxt, numocts, buffer, bufsiz)
- #define rtEncQ825TBCDString(pctxt, str) rtxEncQ825TBCDString(pctxt, str)
- #define rtTBCDBinToChar(bcdDigit, pdigit) rtxTBCDBinToChar(bcdDigit, pdigit)
- #define rtTBCDCharToBin(digit, pbyte) rtxTBCDCharToBin(digit, pbyte)

### Functions

- const char * rtBCDToString (OSUINT32 numocts, const OSOCTET *data, char *buffer, size_t bufsiz, OSBOOL isTBCD)
- int rtStringToBCD (const char *str, OSOCTET *bcdStr, size_t bufsiz, OSBOOL isTBCD)
- int rtStringToDynBCD (OSCTXT *pctxt, const char *str, ASN1DynOctStr *poctstr)
- int rtStringToTBCD (const char *str, OSOCTET *bcdStr, size_t bufsiz)
- const char * rtTBCDToString (OSUINT32 numocts, const OSOCTET *data, char *buffer, size_t bufsiz)

#### 4.2.1 Detailed Description

Binary-decimal conversion functions.

## 4.3 rtCompare.h File Reference

```
#include "asn1type.h"
#include <limits.h>
#include <stdio.h>
#include <string.h>
#include <stddef.h>
#include <setjmp.h>
#include <stdlib.h>
#include <time.h>
#include <wchar.h>
#include "rtxsrc/rtxExternDefs.h"
#include "rtxsrc/rtxSList.h"
#include "rtxsrc/rtxStack.h"
#include "rtxsrc/rtxUTF8.h"
#include "rtsrc/asn1tag.h"
#include "rtsrc/asn1ErrCodes.h"
#include "rtsrc/asn1version.h"
#include "rtsrc/rtExternDefs.h"
#include <float.h>
#include "rtxsrc/rtxBitString.h"
#include "rtsrc/rtContext.h"
#include "rtxsrc/rtxCommonDefs.h"
#include "rtxsrc/rtxError.h"
#include "rtxsrc/rtxMemory.h"
#include "rtsrc/asn1type.h"
```

### Defines

- #define rtCmpOID rtCmpOIDValue
- #define rtCmpOID64 rtCmpOID64Value

### Functions

- OSBOOL rtCmpBoolean (const char *name, OSBOOL value, OSBOOL compValue, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpInt8 (const char *name, OSINT8 value, OSINT8 compValue, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpSInt (const char *name, OSINT16 value, OSINT16 compValue, char *errBuff, OSSIZE errBuffSize)

- OSBOOL rtCmpUInt8 (const char *name, OSUINT8 value, OSUINT8 compValue, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpUSInt (const char *name, OSUINT16 value, OSUINT16 compValue, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpInteger (const char *name, OSINT32 value, OSINT32 compValue, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpUnsigned (const char *name, OSUINT32 value, OSUINT32 compValue, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpInt64 (const char *name, OSINT64 value, OSINT64 compValue, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpUInt64 (const char *name, OSUINT64 value, OSUINT64 compValue, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpBitStr (const char *name, OSSIZE numbits, const OSOCTET *data, OSSIZE compNumbits, const OSOCTET *compData, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpOctStr (const char *name, OSSIZE numocts, const OSOCTET *data, OSSIZE compNumocts, const OSOCTET *compData, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpCharStr (const char *name, const char *cstring, const char *compCstring, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmp16BitCharStr (const char *name, Asn116BitCharString *bstring, Asn116BitCharString *compBstring, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmp32BitCharStr (const char *name, Asn132BitCharString *bstring, Asn132BitCharString *compBstring, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpReal (const char *name, OSREAL value, OSREAL compValue, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpOIDValue (const char *name, ASN1OBJID *pOID, ASN1OBJID *pcompOID, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpOID64Value (const char *name, ASN1OID64 *pOID, ASN1OID64 *pcompOID, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpOpenType (const char *name, OSSIZE numocts, const OSOCTET *data, OSSIZE compNumocts, const OSOCTET *compData, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpOpenTypeExt (const char *name, OSRTDList *pElemList, OSRTDList *pCompElemList, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpTag (const char *name, int tag, int compTag, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpSeqOfElements (const char *name, OSSIZE noOfElems, OSSIZE compNoOfElems, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpOptional (const char *name, unsigned presentBit, unsigned compPresentBit, char *errBuff, OSSIZE errBuffSize)
- OSBOOL rtCmpToStdoutBoolean (const char *name, OSBOOL value, OSBOOL compValue)
- OSBOOL rtCmpToStdoutInteger (const char *name, OSINT32 value, OSINT32 compValue)
- OSBOOL rtCmpToStdoutInt64 (const char *name, OSINT64 value, OSINT64 compValue)
- OSBOOL rtCmpToStdoutUnsigned (const char *name, OSUINT32 value, OSUINT32 compValue)
- OSBOOL rtCmpToStdoutUInt64 (const char *name, OSUINT64 value, OSUINT64 compValue)
- OSBOOL rtCmpToStdoutBitStr (const char *name, OSSIZE numbits, const OSOCTET *data, OSSIZE compNumbits, const OSOCTET *compData)
- OSBOOL rtCmpToStdoutOctStr (const char *name, OSSIZE numocts, const OSOCTET *data, OSSIZE compNumocts, const OSOCTET *compData)
- OSBOOL rtCmpToStdoutCharStr (const char *name, const char *cstring, const char *compCstring)
- OSBOOL rtCmpToStdout16BitCharStr (const char *name, Asn116BitCharString *bstring, Asn116BitCharString *compBstring)
- OSBOOL rtCmpToStdout32BitCharStr (const char *name, Asn132BitCharString *bstring, Asn132BitCharString *compBstring)
- OSBOOL rtCmpToStdoutReal (const char *name, OSREAL value, OSREAL compValue)

- OSBOOL rtCmpToStdoutOID (const char *name, ASN1OBJID *pOID, ASN1OBJID *pcompOID)
- OSBOOL rtCmpToStdoutOIDValue (const char *name, ASN1OBJID *pOID, ASN1OBJID *pcompOID)
- OSBOOL rtCmpToStdoutOID64 (const char *name, ASN1OID64 *pOID, ASN1OID64 *pcompOID)
- OSBOOL rtCmpToStdoutOID64Value (const char *name, ASN1OID64 *pOID, ASN1OID64 *pcompOID)
- OSBOOL rtCmpToStdoutOpenType (const char *name, OSSIZE numocts, const OSOCTET *data, OSSIZE compNumocts, const OSOCTET *compData)
- OSBOOL rtCmpToStdoutOpenTypeExt (const char *name, OSRTDList *pElemList, OSRTDList *pCompElemList)
- OSBOOL rtCmpToStdoutTag (const char *name, int tag, int compTag)
- OSBOOL rtCmpToStdoutSeqOfElements (const char *name, OSSIZE noOfElems, OSSIZE compNoOfElems)
- OSBOOL rtCmpToStdoutOptional (const char *name, unsigned presentBit, unsigned compPresentBit)

### 4.3.1 Detailed Description

Functions for comparing the values of primitive ASN.1 types.

## 4.4 rtCopy.h File Reference

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

### Defines

- #define RTCOPYCHARSTR(pctxt, src, dst) do { char* ptr; rtCopyCharStr (pctxt, src, &ptr); *dst = ptr; } while(0)

### Functions

- OSBOOL rtCopyBitStr (OSUINT32 srcNumbits, const OSOCTET *pSrcData, OSUINT32 *pDstNumbits, OSOCTET *pDstData)
- OSBOOL rtCopyDynBitStr (OSCTXT *pctxt, const ASN1DynBitStr *pSrcData, ASN1DynBitStr *pDstData)
- OSBOOL rtCopyOctStr (OSUINT32 srcNumocts, const OSOCTET *pSrcData, OSUINT32 *pDstNumocts, OSOCTET *pDstData)
- OSBOOL rtCopyDynOctStr (OSCTXT *pctxt, const ASN1DynOctStr *pSrcData, ASN1DynOctStr *pDstData)
- OSBOOL rtCopyCharStr (OSCTXT *pctxt, const char *srcStr, char **dstStr)
- OSBOOL rtCopy16BitCharStr (OSCTXT *pctxt, const Asn116BitCharString *srcStr, Asn116BitCharString *dstStr)
- OSBOOL rtCopy32BitCharStr (OSCTXT *pctxt, const Asn132BitCharString *srcStr, Asn132BitCharString *dstStr)
- OSBOOL rtCopyOID (const ASN1OBJID *srcOID, ASN1OBJID *dstOID)
- OSBOOL rtCopyOID64 (const ASN1OID64 *srcOID, ASN1OID64 *dstOID)
- OSBOOL rtCopyOpenType (OSCTXT *pctxt, const ASN1OpenType *srcOT, ASN1OpenType *dstOT)
- OSBOOL rtCopyOpenTypeExt (OSCTXT *pctxt, const OSRTDList *srcList, OSRTDList *dstList)

### 4.4.1 Detailed Description

Functions for copying values of primitive ASN.1 types.

### 4.4.2 Define Documentation

#### 4.4.2.1 #define RTCOPYCHARSTR(pctxt, src, dst) do { char* ptr; rtCopyCharStr (pctxt, src, &ptr); *dst = ptr; } while(0)

This macro copies the source string to the destination string by calling the common runtime function rtCopyCharStr. This function allocates memory on the managed heap to store the string that will be released when rtxMemFree or rtFreeContext are called.

**Parameters**

***pctxt*** A pointer to an OSCTXT data structure.

***src*** The source string.

***dst*** The destination string.

## 4.5 rtxBase64.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

### Functions

- long rtxBase64EncodeData (OSCTXT *pctxt, const char *pSrcData, size_t srcDataSize, OSOCTET **ppDstData)
- long rtxBase64DecodeData (OSCTXT *pctxt, const char *pSrcData, size_t srcDataSize, OSOCTET **ppDstData)
- long rtxBase64DecodeDataToFSB (OSCTXT *pctxt, const char *pSrcData, size_t srcDataSize, OSOCTET *buf, size_t bufsiz)
- long rtxBase64GetBinDataLen (const char *pSrcData, size_t srcDataSize)

### 4.5.1 Detailed Description

### 4.5.2 Function Documentation

#### 4.5.2.1 long rtxBase64DecodeData (OSCTXT * *pctxt*, const char * *pSrcData*, size_t *srcDataSize*, OSOCTET ** *ppDstData*)

Decode base64 string to binary form into a dynamic buffer.

**Parameters**

***pctxt*** Pointer to context structure.

***pSrcData*** Pointer to base64 string to decode.

***srcDataSize*** Length of the base64 string.

***ppDstData*** Pointer to pointer variable to hold address of dynamically allocated buffer to hold data.

**Returns**

Completion status of operation:

- number of binary bytes written
- negative return value is error.

#### 4.5.2.2 long rtxBase64DecodeDataToFSB (OSCTXT * *pctxt*, const char * *pSrcData*, size_t *srcDataSize*, OSOCTET * *buf*, size_t *bufsiz*)

Decode base64 string to binary form into a fixed-size buffer.

**Parameters**

***pctxt*** Pointer to context structure.

***pSrcData*** Pointer to base64 string to decode.

***srcDataSize*** Length of the base64 string.

***buf*** Address of buffer to receive decoded binary data.

***bufsiz*** Size of output buffer.

**Returns**

Completion status of operation:

- number of binary bytes written
- negative return value is error.

#### 4.5.2.3 long rtxBase64EncodeData (OSCTXT * *pctxt*, const char * *pSrcData*, size_t *srcDataSize*, OSOCTET ** *ppDstData*)

Encode binary data into base64 string form to a dynamic buffer.

**Parameters**

***pctxt*** Pointer to context structure.

***pSrcData*** Pointer to binary data to encode.

***srcDataSize*** Length of the binary data in octets.

***ppDstData*** Pointer to pointer variable to hold address of dynamically allocated buffer the encoded base64 string.

**Returns**

Completion status of operation:

- number of binary bytes written
- negative return value is error.

#### 4.5.2.4 long rtxBase64GetBinDataLen (const char * *pSrcData*, size_t *srcDataSize*)

Calculate number of byte required to hold a decoded base64 string in binary form.

**Parameters**

***pSrcData*** Pointer to base64 string to decode.

***srcDataSize*** Length of the base64 string.

**Returns**

Completion status of operation: If success, positive value is number of bytes, If failure, negative status code.

## 4.6 rtxBigInt.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

### Classes

- struct OSBigInt

### Typedefs

- typedef struct OSBigInt **OSBigInt**

### Functions

- void rtxBigIntInit (OSBigInt *pInt)
- int rtxBigIntSetStr (OSCTXT *pctxt, OSBigInt *pInt, const char *value, int radix)
- int rtxBigIntSetStrn (OSCTXT *pctxt, OSBigInt *pInt, const char *value, OSSIZE len, int radix)
- int rtxBigIntSetInt64 (OSCTXT *pctxt, OSBigInt *pInt, OSINT64 value)
- int rtxBigIntSetUInt64 (OSCTXT *pctxt, OSBigInt *pInt, OSUINT64 value)
- int rtxBigIntSetBytes (OSCTXT *pctxt, OSBigInt *pInt, OSOCTET *value, OSSIZE vallen)
- OSSIZE rtxBigIntGetDataLen (const OSBigInt *pInt)
- int rtxBigIntGetData (OSCTXT *pctxt, const OSBigInt *pInt, OSOCTET *buffer, OSSIZE bufSize)
- OSSIZE rtxBigIntDigitsNum (const OSBigInt *pInt, int radix)
- int rtxBigIntCopy (OSCTXT *pctxt, const OSBigInt *pSrc, OSBigInt *pDst)
- int rtxBigIntFastCopy (OSCTXT *pctxt, const OSBigInt *pSrc, OSBigInt *pDst)
- int rtxBigIntToString (OSCTXT *pctxt, const OSBigInt *pInt, int radix, char *str, OSSIZE strSize)
- int rtxBigIntPrint (const OSUTF8CHAR *name, const OSBigInt *bigint, int radix)
- int rtxBigIntCompare (const OSBigInt *arg1, const OSBigInt *arg2)
- int rtxBigIntStrCompare (OSCTXT *pctxt, const char *arg1, const char *arg2)
- void rtxBigIntFree (OSCTXT *pctxt, OSBigInt *pInt)
- int rtxBigIntAdd (OSCTXT *pctxt, OSBigInt *result, const OSBigInt *arg1, const OSBigInt *arg2)
- int rtxBigIntSubtract (OSCTXT *pctxt, OSBigInt *result, const OSBigInt *arg1, const OSBigInt *arg2)
- int rtxBigIntMultiply (OSCTXT *pctxt, OSBigInt *result, const OSBigInt *arg1, const OSBigInt *arg2)
- unsigned short **rtxBigIntBitsPerDigit** (int radix)
- short **rtxBigIntByteRadix** (int halfRadix)

### 4.6.1 Detailed Description

### 4.6.2 Function Documentation

#### 4.6.2.1 int rtxBigIntAdd (OSCTXT * *pctxt*, OSBigInt * *result*, const OSBigInt * *arg1*, const OSBigInt * *arg2*)

This function is used to add two big integer values.

**Parameters**

***pctxt*** Pointer to a context structure.

***result*** Pointer to big integer structure to receive result.

***arg1*** First big integer to add.

***arg2*** Second big integer to add.

**Returns**

Result of operation: 0 = success, negative error code if error.

#### 4.6.2.2 int rtxBigIntCompare (const OSBigInt * *arg1*, const OSBigInt * *arg2*)

This function is used to compare two big integer values.

**Parameters**

***arg1*** First big integer to compare.

***arg2*** Second big integer to compare.

**Returns**

Result of comparison: -1 = arg1 < arg2, 0 = arg1 == arg2, +1 = arg1 > arg2

#### 4.6.2.3 int rtxBigIntCopy (OSCTXT * *pctxt*, const OSBigInt * *pSrc*, OSBigInt * *pDst*)

This function is used to copy a big integer data value from one structure to another.

**Parameters**

***pctxt*** Pointer to a context structure.

***pSrc*** Pointer to source big integer structure.

***pDst*** Pointer to destination big integer structure.

**Returns**

Status of the operation, 0 = success, negative code if error.

#### 4.6.2.4 OSSIZE rtxBigIntDigitsNum (const OSBigInt * *pInt*, int *radix*)

This function is used to get the number of digits in the binary big integer data value based on radix.

**Parameters**

***pInt*** Pointer to big integer structure.

***radix*** Radix of the string value, Valid values are 2, 8, 10, or 16.

**Returns**

Number of digits in the binary data value.

#### 4.6.2.5 int rtxBigIntFastCopy (OSCTXT * *pctxt*, const OSBigInt * *pSrc*, OSBigInt * *pDst*)

This function is used to copy one BigInt to another. This function will not allocate memory for the byte buffer if the destination BigInt already has a large enough allocated array to hold the data. The destination BigInt must have been initialized using the rtxBigIntInit function.

**Parameters**

***pctxt*** Pointer to a context structure.

***pSrc*** Pointer to source big integer structure.

***pDst*** Pointer to destination big integer structure.

**Returns**

Status of the operation, 0 = success, negative code if error.

#### 4.6.2.6 void rtxBigIntFree (OSCTXT * *pctxt*, OSBigInt * *pInt*)

This function frees internal memory within the big integer structure.

**Parameters**

***pctxt*** Pointer to a context structure.

***pInt*** Pointer to big integer structure in which memory is to be freed.

#### 4.6.2.7 int rtxBigIntGetData (OSCTXT * *pctxt*, const OSBigInt * *pInt*, OSOCTET * *buffer*, OSSIZE *bufSize*)

This function is used to get the binary big integer data value in a byte array.

**Parameters**

***pctxt*** Pointer to a context structure.

***pInt*** Pointer to big integer structure.

***buffer*** Buffer into which binary big integer value is to be copied.

***bufSize*** Size of the data buffer.

**Returns**

If success, number of bytes in byte array; if error, negative error code.

#### 4.6.2.8 OSSIZE rtxBigIntGetDataLen (const OSBigInt * *pInt*)

This function is used to get the size in bytes of the binary big integer data value.

**Parameters**

***pInt*** Pointer to big integer structure.

**Returns**

Length in bytes of the binary data value.

#### 4.6.2.9 void rtxBigIntInit (OSBigInt * *pInt*)

This function initializes a big integer structure. It must be called prior to working with the structure.

**Parameters**

***pInt*** Pointer to big integer data structure.

#### 4.6.2.10 int rtxBigIntMultiply (OSCTXT * *pctxt*, OSBigInt * *result*, const OSBigInt * *arg1*, const OSBigInt * *arg2*)

This function is used to multiply two big integer values.

**Parameters**

***pctxt*** Pointer to a context structure.

***result*** Pointer to big integer structure to receive result.

***arg1*** First big integer to be multiplied.

***arg2*** Second big integer to be multiplied.

**Returns**

Result of operation: 0 = success, negative error code if error.

#### 4.6.2.11 int rtxBigIntPrint (const OSUTF8CHAR * *name*, const OSBigInt * *bigint*, int *radix*)

This function is used to print a big integer value to standard output.

**Parameters**

***name*** Name to print in "name=value" format.

***bigint*** Pointer to big integer value to be printed.

***radix*** Radix of the string value, Valid values are 2, 8, 10, or 16.

**Returns**

Status of the operation, 0 = success, negative code if error.

#### 4.6.2.12 int rtxBigIntSetBytes (OSCTXT * *pctxt*, OSBigInt * *pInt*, OSOCTET * *value*, OSSIZE *vallen*)

This function sets a big integer binary value from a byte array. The array is assumed to hold the value in binary form.

**Parameters**

***pctxt*** Pointer to a context structure.

***pInt*** Pointer to big integer structure to receive converted value.

***value*** Buffer containing binary integer value.

***vallen*** Number of byte in the value buffer.

**Returns**

Status of the operation, 0 = success, negative code if error.

#### 4.6.2.13 int rtxBigIntSetInt64 (OSCTXT * *pctxt*, OSBigInt * *pInt*, OSINT64 *value*)

This function sets a big integer binary value from a signed 64-bit integer value.

**Parameters**

***pctxt*** Pointer to a context structure.

***pInt*** Pointer to big integer structure to receive converted value.

***value*** 64-bit integer value to convert.

**Returns**

Status of the operation, 0 = success, negative code if error.

#### 4.6.2.14 int rtxBigIntSetStr (OSCTXT * *pctxt*, OSBigInt * *pInt*, const char * *value*, int *radix*)

This function sets a big integer binary value from a null-terminated string.

**Parameters**

***pctxt*** Pointer to a context structure.

***pInt*** Pointer to big integer structure to receive converted value.

***value*** Numeric string to convert.

***radix*** Radix of the string value, Valid values are 2, 8, 10, or 16.

**Returns**

Status of the operation, 0 = success, negative code if error.

#### 4.6.2.15 int rtxBigIntSetStrn (OSCTXT * *pctxt*, OSBigInt * *pInt*, const char * *value*, OSSIZE *len*, int *radix*)

This function sets a big integer binary value from a character string using the given number of characters.

**Parameters**

***pctxt*** Pointer to a context structure.

***pInt*** Pointer to big integer structure to receive converted value.

***value*** Numeric string to convert.

***len*** Number of bytes from character string to use.

***radix*** Radix of the string value, Valid values are 2, 8, 10, or 16.

**Returns**

Status of the operation, 0 = success, negative code if error.

#### 4.6.2.16 int rtxBigIntSetUInt64 (OSCTXT * *pctxt*, OSBigInt * *pInt*, OSUINT64 *value*)

This function sets a big integer binary value from an unsigned 64-bit integer value.

**Parameters**

***pctxt*** Pointer to a context structure.

***pInt*** Pointer to big integer structure to receive converted value.

***value*** 64-bit integer value to convert.

**Returns**

Status of the operation, 0 = success, negative code if error.

#### 4.6.2.17 int rtxBigIntStrCompare (OSCTXT * *pctxt*, const char * *arg1*, const char * *arg2*)

This function is used to compare two big integer numeric strings.

**Parameters**

***pctxt*** Pointer to a context structure.

***arg1*** First big integer string to compare.

***arg2*** Second big integer string to compare.

**Returns**

Result of comparison: -1 = arg1 $<$ arg2, 0 = arg1 == arg2, +1 = arg1 $>$ arg2

#### 4.6.2.18 int rtxBigIntSubtract (OSCTXT * *pctxt*, OSBigInt * *result*, const OSBigInt * *arg1*, const OSBigInt * *arg2*)

This function is used to subtract one big integer value from another.

**Parameters**

***pctxt*** Pointer to a context structure.

***result*** Pointer to big integer structure to receive result.

***arg1*** Big integer value that arg2 is subtracted from (minuend).

***arg2*** Big integer to be subtracted from arg1 (subtrahend).

**Returns**

Result of operation: 0 = success, negative error code if error.

#### 4.6.2.19 int rtxBigIntToString (OSCTXT * *pctxt*, const OSBigInt * *pInt*, int *radix*, char * *str*, OSSIZE *strSize*)

This function is used to convert a binary big integer value to a string.

**Parameters**

***pctxt*** Pointer to a context structure.

***pInt*** Pointer to big integer structure to convert.

***radix*** Radix of the string value, Valid values are 2, 8, 10, or 16.

***str*** Character string buffer to receive converted value.

***strSize*** Size, in bytes, of the character string buffer.

**Returns**

Status of the operation, 0 = success, negative code if error.

## 4.7 rtxBitString.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

### Defines

- #define OSRTBYTEARRAYSIZE(numbits) ((numbits+7)/8)

### Functions

- OSUINT32 rtxGetBitCount (OSUINT32 value)
- int rtxSetBit (OSOCTET *pBits, OSSIZE numbits, OSSIZE bitIndex)
- OSUINT32 rtxSetBitFlags (OSUINT32 flags, OSUINT32 mask, OSBOOL action)
- int rtxClearBit (OSOCTET *pBits, OSSIZE numbits, OSSIZE bitIndex)
- OSBOOL rtxTestBit (const OSOCTET *pBits, OSSIZE numbits, OSSIZE bitIndex)
- OSSIZE rtxLastBitSet (const OSOCTET *pBits, OSSIZE numbits)
- int rtxCheckBitBounds (OSCTXT *pctxt, OSOCTET **ppBits, OSSIZE *pNumocts, OSSIZE minRequiredBits, OSSIZE preferredLimitBits)

### 4.7.1 Detailed Description

- Contains utility functions for setting, clearing, and testing bits at any position in an arbitrarily sized array of bytes.

## 4.8 rtxCharStr.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

### Functions

- int rtxStricmp (const char *str1, const char *str2)
- char * rtxStrcat (char *dest, size_t bufsiz, const char *src)
- char * rtxStrncat (char *dest, size_t bufsiz, const char *src, size_t nchars)
- char * rtxStrcpy (char *dest, size_t bufsiz, const char *src)
- char * rtxStrncpy (char *dest, size_t bufsiz, const char *src, size_t nchars)
- char * rtxStrdup (OSCTXT *pctxt, const char *src)
- const char * rtxStrJoin (char *dest, size_t bufsiz, const char *str1, const char *str2, const char *str3, const char *str4, const char *str5)
- const char * rtxStrDynJoin (OSCTXT *pctxt, const char *str1, const char *str2, const char *str3, const char *str4, const char *str5)
- int rtxIntToCharStr (OSINT32 value, char *dest, size_t bufsiz, char padchar)
- int rtxUIntToCharStr (OSUINT32 value, char *dest, size_t bufsiz, char padchar)
- int rtxInt64ToCharStr (OSINT64 value, char *dest, size_t bufsiz, char padchar)
- int rtxUInt64ToCharStr (OSUINT64 value, char *dest, size_t bufsiz, char padchar)
- int rtxSizeToCharStr (size_t value, char *dest, size_t bufsiz, char padchar)
- int rtxHexCharsToBinCount (const char *hexstr, size_t nchars)
- int rtxHexCharsToBin (const char *hexstr, size_t nchars, OSOCTET *binbuf, size_t bufsize)
- int rtxCharStrToInt (const char *cstr, OSINT32 *pvalue)
- int **rtxCharStrToInt8** (const char *cstr, OSINT8 *pvalue)
- int **rtxCharStrToInt16** (const char *cstr, OSINT16 *pvalue)
- int **rtxCharStrToUInt** (const char *cstr, OSUINT32 *pvalue)
- int **rtxCharStrToUInt8** (const char *cstr, OSUINT8 *pvalue)
- int **rtxCharStrToUInt16** (const char *cstr, OSUINT16 *pvalue)

### 4.8.1 Detailed Description

## 4.9 rtxCommon.h File Reference

```
#include "rtxsrc/osSysTypes.h"
#include "rtxsrc/osMacros.h"
#include "rtxsrc/rtxExternDefs.h"
#include "rtxsrc/rtxBigInt.h"
#include "rtxsrc/rtxBitString.h"
#include "rtxsrc/rtxBuffer.h"
#include "rtxsrc/rtxCharStr.h"
#include "rtxsrc/rtxCommonDefs.h"
#include "rtxsrc/rtxDateTime.h"
#include "rtxsrc/rtxDiag.h"
#include "rtxsrc/rtxEnum.h"
#include "rtxsrc/rtxError.h"
#include "rtxsrc/rtxFile.h"
#include "rtxsrc/rtxMemory.h"
#include "rtxsrc/rtxPattern.h"
#include "rtxsrc/rtxReal.h"
#include "rtxsrc/rtxUTF8.h"
#include "rtxsrc/rtxUtil.h"
```

### 4.9.1 Detailed Description

Common runtime constants, data structure definitions, and run-time functions to support various data encoding standards.

## 4.10 rtxContext.h File Reference

```
#include "rtxsrc/rtxDList.h"
```

### Classes

- struct OSRTErrLocn
- struct OSRTErrInfo
- struct OSRTErrInfoList
- struct OSRTBuffer
- struct OSRTBufSave
- struct OSCTXT

### Defines

- #define **OSRTENCBUFSIZ** 1024
- #define **OSRTERRSTKSIZ** 8
- #define **OSRTMAXERRPRM** 5
- #define **OSDIAG** 0x80000000
- #define **OSTRACE** 0x40000000
- #define **OSDISSTRM** 0x20000000
- #define **OSNOSTRMBACKOFF** 0x8000000
- #define **OS3GMOBORIG** 0x4000000
- #define **OSCDECL**
- #define **OSRT_GET_FIRST_ERROR_INFO**(pctxt)
- #define **OSRT_GET_LAST_ERROR_INFO**(pctxt)
- #define **OSRTISSTREAM**(pctxt) ((pctxt)->pStream != 0 && !((pctxt)->flags & OSDISSTRM))
- #define **OSRTBUFCUR**(pctxt) (pctxt)->buffer.data[(pctxt)->buffer.byteIndex]
- #define **OSRTBUFPTR**(pctxt) &(pctxt)->buffer.data[(pctxt)->buffer.byteIndex]
- #define **OSRTBUFFER**(pctxt) (pctxt)->buffer.data
- #define **OSRTBUFSIZE**(pctxt) (pctxt)->buffer.size
- #define **OSRTBUFSAVE**(pctxt)
- #define **OSRTBUFSAVE2**(pctxt, pSavedBuf)
- #define **OSRTBUFRESTORE**(pctxt)
- #define **OSRTBUFRESTORE2**(pctxt, pSavedBuf)
- #define **OSRTBYTEALIGNED**(pctxt) ((pctxt)->buffer.bitOffset == 8 || (pctxt)->buffer.bitOffset == 0)
- #define rtxCtxtGetMsgPtr(pctxt) (pctxt)->buffer.data
- #define rtxCtxtGetMsgLen(pctxt) (pctxt)->buffer.byteIndex
- #define rtxCtxtTestFlag(pctxt, mask) (((pctxt)->flags & mask) != 0)
- #define rtxCtxtPeekElemName(pctxt)
- #define rtxByteAlign(pctxt)
- #define rtxCtxtSetProtocolVersion(pctxt, value) (pctxt)->version = value
- #define **rtxMarkBitPos**(pctxt, ppos, pbitoff) (*(pbitoff) = (OSUINT8) (pctxt)->buffer.bitOffset, rtxMarkPos (pctxt, ppos))
- #define **rtxResetToBitPos**(pctxt, pos, bitoff) ((pctxt)->buffer.bitOffset = (OSUINT8) bitoff, rtxResetToPos (pctxt, pos))
- #define **RTXCTXTPUSHARRAYELEMNAME**(pctxt, name, idx) rtxCtxtPushArrayElemName(pctxt,OSUTF8(name),idx)
- #define **RTXCTXTPOPARRAYELEMNAME**(pctxt) rtxCtxtPopArrayElemName(pctxt)
- #define **RTXCTXTPUSHELEMNAME**(pctxt, name) rtxCtxtPushElemName(pctxt,OSUTF8(name))

- #define **RTXCTXTPOPELEMNAME**(pctxt) rtxCtxtPopElemName(pctxt)
- #define **RTXCTXTPUSHTYPENAME**(pctxt, name) rtxCtxtPushTypeName(pctxt,OSUTF8(name))
- #define **RTXCTXTPOPTYPENAME**(pctxt) rtxCtxtPopTypeName(pctxt)

## Typedefs

- typedef OSUINT32 **OSRTFLAGS**
- typedef int(∗ OSFreeCtxtAppInfoPtr )(struct OSCTXT ∗pctxt)
- typedef int(∗ OSResetCtxtAppInfoPtr )(struct OSCTXT ∗pctxt)
- typedef void(∗ OSFreeCtxtGlobalPtr )(struct OSCTXT ∗pctxt)
- typedef struct OSCTXT **OSCTXT**
- typedef void ∗OSCDECL ∗ **OSMallocFunc** (OSSIZE size)
- typedef void ∗OSCDECL ∗ **OSReallocFunc** (void ∗ptr, OSSIZE size)

## Functions

- typedef **void** (OSCDECL ∗OSFreeFunc)(void ∗ptr)
- int rtxInitContext (OSCTXT ∗pctxt)
- int rtxInitContextExt (OSCTXT ∗pctxt, OSMallocFunc malloc_func, OSReallocFunc realloc_func, OSFreeFunc free_func)
- int rtxInitThreadContext (OSCTXT ∗pctxt, const OSCTXT ∗pSrcCtxt)
- int rtxInitContextBuffer (OSCTXT ∗pctxt, OSOCTET ∗bufaddr, OSSIZE bufsiz)
- int rtxCtxtSetBufPtr (OSCTXT ∗pctxt, OSOCTET ∗bufaddr, OSSIZE bufsiz)
- OSSIZE rtxCtxtGetBitOffset (OSCTXT ∗pctxt)
- int rtxCtxtSetBitOffset (OSCTXT ∗pctxt, OSSIZE offset)
- OSSIZE rtxCtxtGetIOByteCount (OSCTXT ∗pctxt)
- int rtxCheckContext (OSCTXT ∗pctxt)
- void rtxFreeContext (OSCTXT ∗pctxt)
- void rtxCopyContext (OSCTXT ∗pdest, OSCTXT ∗psrc)
- void rtxCtxtSetFlag (OSCTXT ∗pctxt, OSUINT32 mask)
- void rtxCtxtClearFlag (OSCTXT ∗pctxt, OSUINT32 mask)
- int rtxCtxtPushArrayElemName (OSCTXT ∗pctxt, const OSUTF8CHAR ∗elemName, OSSIZE idx)
- int rtxCtxtPushElemName (OSCTXT ∗pctxt, const OSUTF8CHAR ∗elemName)
- int rtxCtxtPushTypeName (OSCTXT ∗pctxt, const OSUTF8CHAR ∗typeName)
- OSBOOL rtxCtxtPopArrayElemName (OSCTXT ∗pctxt)
- const OSUTF8CHAR ∗ rtxCtxtPopElemName (OSCTXT ∗pctxt)
- const OSUTF8CHAR ∗ rtxCtxtPopTypeName (OSCTXT ∗pctxt)
- int **rtxPreInitContext** (OSCTXT ∗pctxt)
- void rtxMemHeapSetFlags (OSCTXT ∗pctxt, OSUINT32 flags)
- void rtxMemHeapClearFlags (OSCTXT ∗pctxt, OSUINT32 flags)
- int rtxMarkPos (OSCTXT ∗pctxt, OSSIZE ∗ppos)
- int rtxResetToPos (OSCTXT ∗pctxt, OSSIZE pos)

### 4.10.1 Detailed Description

Common run-time context definitions.

## 4.10.2 Define Documentation

### 4.10.2.1 #define OSRTBUFRESTORE(pctxt)

**Value:**

```
{ \
(pctxt)->buffer.byteIndex = (pctxt)->savedInfo.byteIndex; \
(pctxt)->flags = (pctxt)->savedInfo.flags; }
```

### 4.10.2.2 #define OSRTBUFRESTORE2(pctxt, pSavedBuf)

**Value:**

```
{ \
(pctxt)->buffer.byteIndex = (pSavedBuf)->byteIndex; \
(pctxt)->buffer.bitOffset = (pSavedBuf)->bitOffset; \
(pctxt)->flags = (pSavedBuf)->flags; }
```

### 4.10.2.3 #define OSRTBUFSAVE(pctxt)

**Value:**

```
{ \
(pctxt)->savedInfo.byteIndex = (pctxt)->buffer.byteIndex; \
(pctxt)->savedInfo.flags = (pctxt)->flags; }
```

### 4.10.2.4 #define OSRTBUFSAVE2(pctxt, pSavedBuf)

**Value:**

```
{ \
(pSavedBuf)->byteIndex = (pctxt)->buffer.byteIndex; \
(pSavedBuf)->bitOffset = (pctxt)->buffer.bitOffset; \
(pSavedBuf)->flags = (pctxt)->flags; }
```

## 4.11 rtxCtype.h File Reference

### Defines

- #define **OS_ISASCII**(c) ((unsigned)(c) < 0x80)
- #define **OS_ISUPPER**(c) (c >= 'A' && c <= 'Z')
- #define **OS_ISLOWER**(c) (c >= 'a' && c <= 'z')
- #define **OS_ISDIGIT**(c) (c >= '0' && c <= '9')
- #define **OS_ISALPHA**(c) (OS_ISUPPER(c) || OS_ISLOWER(c))
- #define **OS_ISSPACE**(c) ((c >= 0x09 && c <= 0x0d) || (c == ' '))
- #define **OS_ISPUNCT**(c) (c >= 0 && c <= 0x20)
- #define **OS_ISALNUM**(c) (OS_ISALPHA(c) || OS_ISDIGIT(c))
- #define **OS_ISPRINT**(c) (c >= ' ' && c <= '~')
- #define **OS_ISGRAPH**(c) (c >= '!' && c <= '~')
- #define **OS_ISCNTRL**(c) ((c >= 0 && c <= 0x1F) || c == 0x7F)
- #define **OS_ISXDIGIT**(c) (OS_ISDIGIT(c) || (c >= 'A' && c <= 'F') || (c >= 'a' && c <= 'f'))
- #define **OS_TOLOWER**(c) (OS_ISUPPER(c) ? (c) - 'A' + 'a' : (c))
- #define **OS_TOUPPER**(c) (OS_ISLOWER(c) ? (c) - 'a' + 'A' : (c))

### 4.11.1 Detailed Description

## 4.12 rtxDateTime.h File Reference

```
#include <time.h>
#include "rtxsrc/rtxContext.h"
```

### Functions

- int rtxDateToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
- int rtxTimeToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
- int rtxDateTimeToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
- int rtxGYearToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
- int rtxGYearMonthToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
- int rtxGMonthToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
- int rtxGMonthDayToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
- int rtxGDayToString (const OSNumDateTime *pvalue, OSUTF8CHAR *buffer, size_t bufsize)
- int rtxGetCurrDateTime (OSNumDateTime *pvalue)
- int rtxCmpDate (const OSNumDateTime *pvalue1, const OSNumDateTime *pvalue2)
- int rtxCmpDate2 (const OSNumDateTime *pvalue, OSINT32 year, OSUINT8 mon, OSUINT8 day, OSBOOL tzflag, OSINT32 tzo)
- int rtxCmpTime (const OSNumDateTime *pvalue1, const OSNumDateTime *pvalue2)
- int rtxCmpTime2 (const OSNumDateTime *pvalue, OSUINT8 hour, OSUINT8 min, OSREAL sec, OSBOOL tzflag, OSINT32 tzo)
- int rtxCmpDateTime (const OSNumDateTime *pvalue1, const OSNumDateTime *pvalue2)
- int rtxCmpDateTime2 (const OSNumDateTime *pvalue, OSINT32 year, OSUINT8 mon, OSUINT8 day, OSUINT8 hour, OSUINT8 min, OSREAL sec, OSBOOL tzflag, OSINT32 tzo)
- int rtxParseDateString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
- int rtxParseTimeString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
- int rtxParseDateTimeString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
- int rtxParseGYearString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
- int rtxParseGYearMonthString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
- int rtxParseGMonthString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
- int rtxParseGMonthDayString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
- int rtxParseGDayString (const OSUTF8CHAR *inpdata, size_t inpdatalen, OSNumDateTime *pvalue)
- int rtxMSecsToDuration (OSINT32 msecs, OSUTF8CHAR *buf, OSUINT32 bufsize)
- int rtxDurationToMSecs (OSUTF8CHAR *buf, OSUINT32 bufsize, OSINT32 *msecs)
- int rtxSetDateTime (OSNumDateTime *pvalue, struct tm *timeStruct)
- int rtxSetLocalDateTime (OSNumDateTime *pvalue, time_t timeMs)
- int rtxSetUtcDateTime (OSNumDateTime *pvalue, time_t timeMs)
- int rtxGetDateTime (const OSNumDateTime *pvalue, time_t *timeMs)
- OSBOOL rtxDateIsValid (const OSNumDateTime *pvalue)
- OSBOOL rtxTimeIsValid (const OSNumDateTime *pvalue)
- OSBOOL rtxDateTimeIsValid (const OSNumDateTime *pvalue)

### 4.12.1 Detailed Description

Common runtime functions for converting to and from various standard date/time formats.

## 4.13 rtxDecimal.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

### Functions

- const char * **rtxNR3toDecimal** (OSCTXT *pctxt, const char *object_p)

### 4.13.1 Detailed Description

Common runtime functions for working with xsd:decimal numbers.

## 4.14 rtxDiag.h File Reference

```
#include <stdarg.h>
#include "rtxsrc/rtxContext.h"
```

### Defines

- #define **RTDIAG1**(pctxt, msg)
- #define **RTDIAG2**(pctxt, msg, a)
- #define **RTDIAG3**(pctxt, msg, a, b)
- #define **RTDIAG4**(pctxt, msg, a, b, c)
- #define **RTDIAG5**(pctxt, msg, a, b, c, d)
- #define **RTDIAGU**(pctxt, ucstr)
- #define **RTHEXDUMP**(pctxt, buffer, numocts)
- #define **RTDIAGCHARS**(pctxt, buf, nchars)
- #define **RTDIAGSTRM2**(pctxt, msg)
- #define **RTDIAGSTRM3**(pctxt, msg, a)
- #define **RTDIAGSTRM4**(pctxt, msg, a, b)
- #define **RTDIAGSTRM5**(pctxt, msg, a, b, c)
- #define **RTHEXDUMPSTRM**(pctxt, buffer, numocts)
- #define **RTDIAGSCHARS**(pctxt, buf, nchars)

### Enumerations

- enum **OSRTDiagTraceLevel** { **OSRTDiagError**, **OSRTDiagWarning**, **OSRTDiagInfo**, **OSRTDiagDebug** }

### Functions

- OSBOOL rtxDiagEnabled (OSCTXT *pctxt)
- OSBOOL rtxSetDiag (OSCTXT *pctxt, OSBOOL value)
- OSBOOL rtxSetGlobalDiag (OSBOOL value)
- void rtxDiagPrint (OSCTXT *pctxt, const char *fmtspec,...)
- void rtxDiagStream (OSCTXT *pctxt, const char *fmtspec,...)
- void rtxDiagHexDump (OSCTXT *pctxt, const OSOCTET *data, size_t numocts)
- void rtxDiagStreamHexDump (OSCTXT *pctxt, const OSOCTET *data, size_t numocts)
- void rtxDiagPrintChars (OSCTXT *pctxt, const char *data, size_t nchars)
- void rtxDiagStreamPrintChars (OSCTXT *pctxt, const char *data, size_t nchars)
- void rtxDiagStreamPrintBits (OSCTXT *pctxt, const char *descr, const OSOCTET *data, size_t bitIndex, size_t nbits)
- void rtxDiagSetTraceLevel (OSCTXT *pctxt, OSRTDiagTraceLevel level)
- OSBOOL rtxDiagTraceLevelEnabled (OSCTXT *pctxt, OSRTDiagTraceLevel level)

### 4.14.1 Detailed Description

Common runtime functions for diagnostic tracing and debugging.

## 4.15 rtxDList.h File Reference

```
#include "rtxsrc/osSysTypes.h"
#include "rtxsrc/rtxExternDefs.h"
#include "rtxsrc/rtxCommonDefs.h"
```

### Classes

- struct OSRTDListNode
- struct OSRTDList
- struct OSRTDListBuf
- struct OSRTDListUTF8StrNode

### Defines

- #define **OSRTDLISTNODESIZE** ((sizeof(OSRTDListNode)+7)&(∼7))
- #define **rtxDListAllocNodeAndData**(pctxt, type, ppnode, ppdata)
- #define **rtxDListAppendData**(pctxt, pList, pData)
- #define **rtxDListFastInit**(pList)
- #define **rtxDListFreeTailNode**(pctxt, pList) rtxDListFreeNode(pctxt,pList,(pList)->tail)
- #define **rtxDListFreeHeadNode**(pctxt, pList) rtxDListFreeNode(pctxt,pList,(pList)->head)

### Typedefs

- typedef struct OSRTDListNode **OSRTDListNode**
- typedef struct OSRTDList **OSRTDList**
- typedef struct OSRTDListBuf **OSRTDListBuf**
- typedef struct OSRTDListUTF8StrNode **OSRTDListUTF8StrNode**
- typedef int(∗ **PEqualsFunc** )(const void ∗a, const void ∗b, const void ∗sortCtxt)

### Functions

- void rtxDListInit (OSRTDList ∗pList)
- OSRTDListNode ∗ rtxDListAppend (struct OSCTXT ∗pctxt, OSRTDList ∗pList, void ∗pData)
- OSRTDListNode ∗ rtxDListAppendCharArray (struct OSCTXT ∗pctxt, OSRTDList ∗pList, size_t length, char ∗pData)
- OSRTDListNode ∗ rtxDListAppendNode (OSRTDList ∗pList, OSRTDListNode ∗pListNode)
- OSRTDListNode ∗ rtxDListInsert (struct OSCTXT ∗pctxt, OSRTDList ∗pList, OSSIZE idx, void ∗pData)
- OSRTDListNode ∗ **rtxDListInsertNode** (OSRTDList ∗pList, OSSIZE idx, OSRTDListNode ∗pListNode)
- OSRTDListNode ∗ rtxDListInsertBefore (struct OSCTXT ∗pctxt, OSRTDList ∗pList, OSRTDListNode ∗node, void ∗pData)
- OSRTDListNode ∗ rtxDListInsertAfter (struct OSCTXT ∗pctxt, OSRTDList ∗pList, OSRTDListNode ∗node, void ∗pData)
- OSRTDListNode ∗ rtxDListFindByIndex (const OSRTDList ∗pList, OSSIZE idx)
- OSRTDListNode ∗ rtxDListFindByData (const OSRTDList ∗pList, void ∗data)
- int rtxDListFindIndexByData (const OSRTDList ∗pList, void ∗data)
- void rtxDListFreeNode (struct OSCTXT ∗pctxt, OSRTDList ∗pList, OSRTDListNode ∗node)
- void rtxDListRemove (OSRTDList ∗pList, OSRTDListNode ∗node)

- void rtxDListFreeNodes (struct OSCTXT *pctxt, OSRTDList *pList)
- void rtxDListFreeAll (struct OSCTXT *pctxt, OSRTDList *pList)
- int rtxDListToArray (struct OSCTXT *pctxt, OSRTDList *pList, void **ppArray, OSSIZE *pElemCount, OSSIZE elemSize)
- int rtxDListAppendArray (struct OSCTXT *pctxt, OSRTDList *pList, void *pArray, OSSIZE numElements, OSSIZE elemSize)
- int rtxDListAppendArrayCopy (struct OSCTXT *pctxt, OSRTDList *pList, const void *pArray, OSSIZE numElements, OSSIZE elemSize)
- int rtxDListToUTF8Str (struct OSCTXT *pctxt, OSRTDList *pList, OSUTF8CHAR **ppstr, char sep)
- OSRTDListNode * **rtxDListInsertSorted** (struct OSCTXT *pctxt, OSRTDList *pList, void *pData, PEqualsFunc equalsFunc, void *sortCtxt)
- OSRTDListNode * **rtxDListInsertNodeSorted** (OSRTDList *pList, OSRTDListNode *pListNode, PEqualsFunc equalsFunc, void *sortCtxt)
- void **rtxDListBufInit** (OSRTDListBuf *pBuf, OSSIZE segSz, void **ppdata, OSSIZE elemSz)
- int **rtxDListBufExpand** (struct OSCTXT *pctxt, OSRTDListBuf *pBuf)
- int **rtxDListBufToArray** (struct OSCTXT *pctxt, OSRTDListBuf *pBuf)

## 4.15.1 Detailed Description

Doubly-Linked List Utility Functions.

## 4.15.2 Define Documentation

### 4.15.2.1 #define rtxDListAllocNodeAndData(pctxt, type, ppnode, ppdata)

**Value:**

```
do { \
*ppnode = (OSRTDListNode*) \
rtxMemAlloc (pctxt, sizeof(type)+OSRTDLISTNODESIZE); \
if (0 != *ppnode) { \
(*ppnode)->data = (void*)((char*)(*ppnode)+OSRTDLISTNODESIZE); \
*ppdata = (type*)((*ppnode)->data); \
} else { *ppdata = 0; } \
} while (0)
```

### 4.15.2.2 #define rtxDListAppendData(pctxt, pList, pData)

**Value:**

```
do { \
   rtxDListAppend(pctxt,pList,pData); \
} while(0);
```

### 4.15.2.3 #define rtxDListFastInit(pList)

**Value:**

```
do { \
if ((pList) != 0) { \
(pList)->head = (pList)->tail = (OSRTDListNode*) 0; \
(pList)->count = 0; } \
} while (0)
```

## 4.16 rtxErrCodes.h File Reference

### Defines

- #define RT_OK 0
- #define RT_OK_FRAG 2
- #define RTERR_BUFOVFLW -1
- #define RTERR_ENDOFBUF -2
- #define RTERR_IDNOTFOU -3
- #define RTERR_INVENUM -4
- #define RTERR_SETDUPL -5
- #define RTERR_SETMISRQ -6
- #define RTERR_NOTINSET -7
- #define RTERR_SEQOVFLW -8
- #define RTERR_INVOPT -9
- #define RTERR_NOMEM -10
- #define RTERR_INVHEXS -11
- #define RTERR_INVREAL -12
- #define RTERR_STROVFLW -13
- #define RTERR_BADVALUE -14
- #define RTERR_TOODEEP -15
- #define RTERR_CONSVIO -16
- #define RTERR_ENDOFFILE -17
- #define RTERR_INVUTF8 -18
- #define RTERR_OUTOFBND -19
- #define RTERR_INVPARAM -20
- #define RTERR_INVFORMAT -21
- #define RTERR_NOTINIT -22
- #define RTERR_TOOBIG -23
- #define RTERR_INVCHAR -24
- #define RTERR_XMLSTATE -25
- #define RTERR_XMLPARSE -26
- #define RTERR_SEQORDER -27
- #define RTERR_FILNOTFOU -28
- #define RTERR_READERR -29
- #define RTERR_WRITEERR -30
- #define RTERR_INVBASE64 -31
- #define RTERR_INVSOCKET -32
- #define RTERR_INVATTR -33
- #define RTERR_REGEXP -34
- #define RTERR_PATMATCH -35
- #define RTERR_ATTRMISRQ -36
- #define RTERR_HOSTNOTFOU -37
- #define RTERR_HTTPERR -38
- #define RTERR_SOAPERR -39
- #define RTERR_EXPIRED -40
- #define RTERR_UNEXPELEM -41
- #define RTERR_INVOCCUR -42
- #define RTERR_INVMSGBUF -43
- #define RTERR_DECELEMFAIL -44

- #define RTERR_DECATTRFAIL -45
- #define RTERR_STRMINUSE -46
- #define RTERR_NULLPTR -47
- #define RTERR_FAILED -48
- #define RTERR_ATTRFIXEDVAL -49
- #define RTERR_MULTIPLE -50
- #define RTERR_NOTYPEINFO -51
- #define RTERR_ADDRINUSE -52
- #define RTERR_CONNRESET -53
- #define RTERR_UNREACHABLE -54
- #define RTERR_NOCONN -55
- #define RTERR_CONNREFUSED -56
- #define RTERR_INVSOCKOPT -57
- #define RTERR_SOAPFAULT -58
- #define RTERR_MARKNOTSUP -59
- #define RTERR_NOTSUPP -60
- #define RTERR_UNBAL -61
- #define RTERR_EXPNAME -62
- #define RTERR_UNICODE -63
- #define RTERR_INVBOOL -64
- #define RTERR_INVNULL -65
- #define RTERR_INVLEN -66
- #define RTERR_UNKNOWNIE -67
- #define RTERR_NOTALIGNED -68
- #define RTERR_EXTRDATA -69
- #define RTERR_INVMAC -70
- #define RTERR_NOSECPARAMS -71

### 4.16.1 Detailed Description

List of numeric status codes that can be returned by common run-time functions and generated code.

## 4.17 rtxError.h File Reference

```
#include "rtxsrc/rtxContext.h"
#include "rtxsrc/rtxErrCodes.h"
```

### Defines

- #define LOG_RTERR(pctxt, stat) rtxErrSetData(pctxt,stat,__FILE__,__LINE__)
- #define **LOG_RTERRNEW**(pctxt, stat) rtxErrSetNewData(pctxt,stat,__FILE__,__LINE__)
- #define OSRTASSERT(condition) if (!(condition)) { rtxErrAssertionFailed(#condition,__LINE__,__FILE__); }
- #define OSRTCHECKPARAM(condition) if (condition) { /* do nothing */ }
- #define **LOG_RTERR1**(pctxt, stat, a) (a,LOG_RTERR (pctxt, stat),stat)
- #define **LOG_RTERRNEW1**(pctxt, stat, a) (a,LOG_RTERRNEW (pctxt, stat),stat)
- #define **LOG_RTERR2**(pctxt, stat, a, b) (a,b,LOG_RTERR (pctxt, stat),stat)
- #define **LOG_RTERRNEW2**(pctxt, stat, a, b) (a,b,LOG_RTERRNEW (pctxt, stat),stat)

### Typedefs

- typedef int(* **OSErrCbFunc** )(const char *ptext, void *cbArg_p)

### Functions

- OSBOOL rtxErrAddCtxtBufParm (OSCTXT *pctxt)
- OSBOOL rtxErrAddDoubleParm (OSCTXT *pctxt, double errParm)
- OSBOOL rtxErrAddErrorTableEntry (const char *const *ppStatusText, OSINT32 minErrCode, OSINT32 maxErrCode)
- OSBOOL rtxErrAddElemNameParm (OSCTXT *pctxt)
- OSBOOL rtxErrAddIntParm (OSCTXT *pctxt, int errParm)
- OSBOOL rtxErrAddInt64Parm (OSCTXT *pctxt, OSINT64 errParm)
- OSBOOL rtxErrAddStrParm (OSCTXT *pctxt, const char *pErrParm)
- OSBOOL rtxErrAddStrnParm (OSCTXT *pctxt, const char *pErrParm, size_t nchars)
- OSBOOL rtxErrAddUIntParm (OSCTXT *pctxt, unsigned int errParm)
- OSBOOL rtxErrAddUInt64Parm (OSCTXT *pctxt, OSUINT64 errParm)
- void rtxErrAssertionFailed (const char *conditionText, int lineNo, const char *fileName)
- const char * rtxErrFmtMsg (OSRTErrInfo *pErrInfo, char *bufp, size_t bufsiz)
- void rtxErrFreeParms (OSCTXT *pctxt)
- char * rtxErrGetText (OSCTXT *pctxt, char *pBuf, size_t *pBufSize)
- char * rtxErrGetTextBuf (OSCTXT *pctxt, char *pbuf, size_t bufsiz)
- OSRTErrInfo * rtxErrNewNode (OSCTXT *pctxt)
- void rtxErrInit ()
- int rtxErrReset (OSCTXT *pctxt)
- void rtxErrLogUsingCB (OSCTXT *pctxt, OSErrCbFunc cb, void *cbArg_p)
- void rtxErrPrint (OSCTXT *pctxt)
- void rtxErrPrintElement (OSRTErrInfo *pErrInfo)
- int rtxErrSetData (OSCTXT *pctxt, int status, const char *module, int lineno)
- int rtxErrSetNewData (OSCTXT *pctxt, int status, const char *module, int lineno)
- int rtxErrGetFirstError (const OSCTXT *pctxt)
- int rtxErrGetLastError (const OSCTXT *pctxt)

- OSSIZE rtxErrGetErrorCnt (const OSCTXT *pctxt)
- int rtxErrGetStatus (const OSCTXT *pctxt)
- int rtxErrResetLastErrors (OSCTXT *pctxt, int errorsToReset)
- int rtxErrCopy (OSCTXT *pDestCtxt, const OSCTXT *pSrcCtxt)
- int rtxErrAppend (OSCTXT *pDestCtxt, const OSCTXT *pSrcCtxt)
- int rtxErrInvUIntOpt (OSCTXT *pctxt, OSUINT32 ident)

### 4.17.1 Detailed Description

Error handling function and macro definitions.

## 4.18 rtxMemBuf.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

### Classes

- struct OSRTMEMBUF

### Defines

- #define **OSMBDFLTSEGSIZE** 1024
- #define **OSMEMBUFPTR**(pmb) ((pmb)->buffer + (pmb)->startidx)
- #define **OSMEMBUFENDPTR**(pmb) ((pmb)->buffer + (pmb)->startidx + (pmb)->usedcnt)
- #define **OSMEMBUFUSEDSIZE**(pmb) ((OSSIZE)(pmb)->usedcnt)
- #define **OSMBAPPENDSTR**(pmb, str)
- #define **OSMBAPPENDSTRL**(pmb, str) rtxMemBufAppend(pmb,(OSOCTET*)str,OSCRTLSTRLEN(str))
- #define **OSMBAPPENDUTF8**(pmb, str)

### Typedefs

- typedef struct OSRTMEMBUF **OSRTMEMBUF**

### Functions

- int rtxMemBufAppend (OSRTMEMBUF *pMemBuf, const OSOCTET *pdata, OSSIZE nbytes)
- int rtxMemBufCut (OSRTMEMBUF *pMemBuf, OSSIZE fromOffset, OSSIZE nbytes)
- void rtxMemBufFree (OSRTMEMBUF *pMemBuf)
- OSOCTET * rtxMemBufGetData (const OSRTMEMBUF *pMemBuf, int *length)
- OSOCTET * rtxMemBufGetDataExt (const OSRTMEMBUF *pMemBuf, OSSIZE *length)
- OSSIZE rtxMemBufGetDataLen (const OSRTMEMBUF *pMemBuf)
- void rtxMemBufInit (OSCTXT *pCtxt, OSRTMEMBUF *pMemBuf, OSSIZE segsize)
- void rtxMemBufInitBuffer (OSCTXT *pCtxt, OSRTMEMBUF *pMemBuf, OSOCTET *buf, OSSIZE bufsize, OSSIZE segsize)
- int rtxMemBufPreAllocate (OSRTMEMBUF *pMemBuf, OSSIZE nbytes)
- void rtxMemBufReset (OSRTMEMBUF *pMemBuf)
- int rtxMemBufSet (OSRTMEMBUF *pMemBuf, OSOCTET value, OSSIZE nbytes)
- OSBOOL rtxMemBufSetExpandable (OSRTMEMBUF *pMemBuf, OSBOOL isExpandable)
- OSBOOL rtxMemBufSetUseSysMem (OSRTMEMBUF *pMemBuf, OSBOOL value)
- OSSIZE rtxMemBufTrimW (OSRTMEMBUF *pMemBuf)

### 4.18.1 Detailed Description

## 4.19 rtxMemory.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

### Defines

- #define **RT_MH_DONTKEEPFREE** 0x1
- #define **RT_MH_VALIDATEPTR** 0x2
- #define **RT_MH_CHECKHEAP** 0x4
- #define **RT_MH_TRACE** 0x8
- #define **RT_MH_DIAG** 0x10
- #define **RT_MH_DIAG_DEBUG** 0x20
- #define **RT_MH_ZEROONFREE** 0x40
- #define **OSRTMH_PROPID_DEFBLKSIZE** 1
- #define **OSRTMH_PROPID_SETFLAGS** 2
- #define **OSRTMH_PROPID_CLEARFLAGS** 3
- #define **OSRTMH_PROPID_KEEPFREEUNITS** 4
- #define **OSRTMH_PROPID_USER** 10
- #define **OSRTXM_K_MEMBLKSIZ** (4∗1024)
- #define OSRTALLOCTYPE(pctxt, type) (type∗) rtxMemHeapAlloc (&(pctxt)->pMemHeap, sizeof(type))
- #define OSRTALLOCTYPEZ(pctxt, type) (type∗) rtxMemHeapAllocZ (&(pctxt)->pMemHeap, sizeof(type))
- #define OSRTREALLOCARRAY(pctxt, pseqof, type)
- #define **OSCRTMALLOC0**(nbytes) malloc(nbytes)
- #define **OSCRTFREE0**(ptr) free(ptr)
- #define **OSCRTMALLOC** rtxMemAlloc
- #define **OSCRTFREE** rtxMemFreePtr
- #define rtxMemAlloc(pctxt, nbytes) rtxMemHeapAlloc(&(pctxt)->pMemHeap,nbytes)
- #define rtxMemSysAlloc(pctxt, nbytes) rtxMemHeapSysAlloc(&(pctxt)->pMemHeap,nbytes)
- #define rtxMemAllocZ(pctxt, nbytes) rtxMemHeapAllocZ(&(pctxt)->pMemHeap,nbytes)
- #define rtxMemSysAllocZ(pctxt, nbytes) rtxMemHeapSysAllocZ(&(pctxt)->pMemHeap,nbytes)
- #define rtxMemRealloc(pctxt, mem_p, nbytes) rtxMemHeapRealloc(&(pctxt)->pMemHeap, (void∗)mem_p, nbytes)
- #define rtxMemSysRealloc(pctxt, mem_p, nbytes) rtxMemHeapSysRealloc(&(pctxt)->pMemHeap,(void∗)mem_p,nbytes)
- #define rtxMemFreePtr(pctxt, mem_p) rtxMemHeapFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)
- #define rtxMemSysFreePtr(pctxt, mem_p) rtxMemHeapSysFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)
- #define rtxMemAllocType(pctxt, ctype) (ctype∗)rtxMemHeapAlloc(&(pctxt)->pMemHeap,sizeof(ctype))
- #define rtxMemSysAllocType(pctxt, ctype) (ctype∗)rtxMemHeapSysAlloc(&(pctxt)->pMemHeap,sizeof(ctype))
- #define rtxMemAllocTypeZ(pctxt, ctype) (ctype∗)rtxMemHeapAllocZ(&(pctxt)->pMemHeap,sizeof(ctype))
- #define rtxMemSysAllocTypeZ(pctxt, ctype) (ctype∗)rtxMemHeapSysAllocZ(&(pctxt)->pMemHeap,sizeof(ctype))
- #define rtxMemFreeType(pctxt, mem_p) rtxMemHeapFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)
- #define rtxMemSysFreeType(pctxt, mem_p) rtxMemHeapSysFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)
- #define rtxMemAllocArray(pctxt, n, type) (type∗)rtxMemHeapAlloc (&(pctxt)->pMemHeap, sizeof(type)∗n)
- #define rtxMemSysAllocArray(pctxt, n, type) (type∗)rtxMemHeapSysAlloc (&(pctxt)->pMemHeap, sizeof(type)∗n)
- #define rtxMemAllocArrayZ(pctxt, n, type) (type∗)rtxMemHeapAllocZ (&(pctxt)->pMemHeap, sizeof(type)∗n)
- #define rtxMemFreeArray(pctxt, mem_p) rtxMemHeapFreePtr(&(pctxt)->pMemHeap, (void∗)mem_p)

- #define rtxMemSysFreeArray(pctxt, mem_p) rtxMemHeapSysFreePtr(&(pctxt)->pMemHeap, (void*)mem_p)
- #define rtxMemReallocArray(pctxt, mem_p, n, type) (type*)rtxMemHeapRealloc(&(pctxt)->pMemHeap, (void*)mem_p, sizeof(type)*n)
- #define rtxMemNewAutoPtr(pctxt, nbytes) rtxMemHeapAlloc(&(pctxt)->pMemHeap, nbytes)
- #define rtxMemAutoPtrRef(pctxt, ptr) rtxMemHeapAutoPtrRef(&(pctxt)->pMemHeap, (void*)(ptr))
- #define rtxMemAutoPtrUnref(pctxt, ptr) rtxMemHeapAutoPtrUnref(&(pctxt)->pMemHeap, (void*)(ptr))
- #define rtxMemAutoPtrGetRefCount(pctxt, ptr) rtxMemHeapAutoPtrGetRefCount(&(pctxt)->pMemHeap, (void*)(ptr))
- #define rtxMemCheckPtr(pctxt, mem_p) rtxMemHeapCheckPtr(&(pctxt)->pMemHeap, (void*)mem_p)
- #define rtxMemCheck(pctxt) rtxMemHeapCheck(&(pctxt)->pMemHeap, __FILE__, __LINE__)
- #define rtxMemPrint(pctxt) rtxMemHeapPrint(&(pctxt)->pMemHeap)
- #define rtxMemSetProperty(pctxt, propId, pProp) rtxMemHeapSetProperty (&(pctxt)->pMemHeap, propId, pProp)

## Functions

- void **rtxMemHeapAddRef** (void **ppvMemHeap)
- void * **rtxMemHeapAlloc** (void **ppvMemHeap, size_t nbytes)
- void * **rtxMemHeapAllocZ** (void **ppvMemHeap, size_t nbytes)
- void * **rtxMemHeapSysAlloc** (void **ppvMemHeap, size_t nbytes)
- void * **rtxMemHeapSysAllocZ** (void **ppvMemHeap, size_t nbytes)
- int **rtxMemHeapCheckPtr** (void **ppvMemHeap, const void *mem_p)
- int **rtxMemHeapCreate** (void **ppvMemHeap)
- int **rtxMemHeapCreateExt** (void **ppvMemHeap, OSMallocFunc malloc_func, OSReallocFunc realloc_func, OSFreeFunc free_func)
- void **rtxMemHeapFreeAll** (void **ppvMemHeap)
- void **rtxMemHeapFreePtr** (void **ppvMemHeap, void *mem_p)
- void **rtxMemHeapSysFreePtr** (void **ppvMemHeap, void *mem_p)
- void * **rtxMemHeapRealloc** (void **ppvMemHeap, void *mem_p, size_t nbytes_)
- void * **rtxMemHeapSysRealloc** (void **ppvMemHeap, void *mem_p, size_t nbytes_)
- void **rtxMemHeapRelease** (void **ppvMemHeap)
- void **rtxMemHeapReset** (void **ppvMemHeap)
- void **rtxMemHeapSetProperty** (void **ppvMemHeap, OSUINT32 propId, void *pProp)
- void * **rtxMemNewArray** (size_t nbytes)
- void * **rtxMemNewArrayZ** (size_t nbytes)
- void **rtxMemDeleteArray** (void *mem_p)
- void * **rtxMemHeapAutoPtrRef** (void **ppvMemHeap, void *ptr)
- int **rtxMemHeapAutoPtrUnref** (void **ppvMemHeap, void *ptr)
- int **rtxMemHeapAutoPtrGetRefCount** (void **ppvMemHeap, void *mem_p)
- void **rtxMemHeapInvalidPtrHook** (void **ppvMemHeap, const void *mem_p)
- void **rtxMemHeapCheck** (void **ppvMemHeap, const char *file, int line)
- void **rtxMemHeapPrint** (void **ppvMemHeap)
- void rtxMemSetAllocFuncs (OSMallocFunc malloc_func, OSReallocFunc realloc_func, OSFreeFunc free_func)
- void **rtxMemFreeOpenSeqExt** (OSCTXT *pctxt, struct OSRTDList *pElemList)
- OSUINT32 rtxMemHeapGetDefBlkSize (OSCTXT *pctxt)
- void rtxMemSetDefBlkSize (OSUINT32 blkSize)
- OSUINT32 rtxMemGetDefBlkSize ()
- OSBOOL rtxMemHeapIsEmpty (OSCTXT *pctxt)
- OSBOOL rtxMemIsZero (const void *pmem, size_t memsiz)
- void rtxMemFree (OSCTXT *pctxt)
- void rtxMemReset (OSCTXT *pctxt)

### 4.19.1 Detailed Description

Memory management function and macro definitions.

## 4.20 rtxPattern.h File Reference

```
#include "rtxsrc/osSysTypes.h"
#include "rtxsrc/rtxExternDefs.h"
#include "rtxsrc/rtxContext.h"
```

### Functions

- OSBOOL rtxMatchPattern (OSCTXT *pctxt, const OSUTF8CHAR *text, const OSUTF8CHAR *pattern)
- OSBOOL **rtxMatchPattern2** (OSCTXT *pctxt, const OSUTF8CHAR *pattern)
- void rtxFreeRegexpCache (OSCTXT *pctxt)

### 4.20.1 Detailed Description

Pattern matching functions.

## 4.21 rtxPrint.h File Reference

```
#include <stdio.h>
#include "rtxsrc/osSysTypes.h"
#include "rtxsrc/rtxExternDefs.h"
#include "rtxsrc/rtxDList.h"
```

### Defines

- #define **OSRTINDENTSPACES** 3

### Functions

- int rtxByteToHexChar (OSOCTET byte, char *buf, OSSIZE bufsize)
- void rtxPrintBoolean (const char *name, OSBOOL value)
- void rtxPrintDate (const char *name, const OSNumDateTime *pvalue)
- void rtxPrintTime (const char *name, const OSNumDateTime *pvalue)
- void rtxPrintDateTime (const char *name, const OSNumDateTime *pvalue)
- void **rtxPrintGYear** (const char *name, const OSNumDateTime *pvalue)
- void **rtxPrintGYearMonth** (const char *name, const OSNumDateTime *pvalue)
- void **rtxPrintGMonth** (const char *name, const OSNumDateTime *pvalue)
- void **rtxPrintGMonthDay** (const char *name, const OSNumDateTime *pvalue)
- void **rtxPrintGDay** (const char *name, const OSNumDateTime *pvalue)
- void rtxPrintInteger (const char *name, OSINT32 value)
- void rtxPrintInt64 (const char *name, OSINT64 value)
- void rtxPrintUnsigned (const char *name, OSUINT32 value)
- void rtxPrintUInt64 (const char *name, OSUINT64 value)
- void rtxPrintHexStr (const char *name, OSSIZE numocts, const OSOCTET *data)
- void rtxPrintHexBinary (const char *name, OSSIZE numocts, const OSOCTET *data)
- void rtxPrintCharStr (const char *name, const char *cstring)
- void rtxPrintUTF8CharStr (const char *name, const OSUTF8CHAR *cstring)
- void rtxPrintUnicodeCharStr (const char *name, const OSUNICHAR *str, int nchars)
- void rtxPrintReal (const char *name, OSREAL value)
- void rtxPrintNull (const char *name)
- void rtxPrintNVP (const char *name, const OSUTF8NVP *value)
- int rtxPrintFile (const char *filename)
- void rtxPrintIndent (void)
- void rtxPrintIncrIndent (void)
- void rtxPrintDecrIndent (void)
- void rtxPrintCloseBrace (void)
- void rtxPrintOpenBrace (const char *)
- void rtxHexDumpToNamedFile (const char *filename, const OSOCTET *data, OSSIZE numocts)
- void rtxHexDumpToFile (FILE *fp, const OSOCTET *data, OSSIZE numocts)
- void rtxHexDumpToFileEx (FILE *fp, const OSOCTET *data, OSSIZE numocts, int bytesPerUnit)
- void rtxHexDump (const OSOCTET *data, OSSIZE numocts)
- void rtxHexDumpEx (const OSOCTET *data, OSSIZE numocts, int bytesPerUnit)
- int rtxHexDumpToString (const OSOCTET *data, OSSIZE numocts, char *buffer, OSSIZE bufferIndex, OSSIZE bufferSize)

- int rtxHexDumpToStringEx (const OSOCTET *data, OSSIZE numocts, char *buffer, OSSIZE bufferIndex, OSSIZE bufferSize, int bytesPerUnit)
- void rtxHexDumpFileContents (const char *inFilePath)
- void rtxHexDumpFileContentsToFile (const char *inFilePath, const char *outFilePath)

### 4.21.1 Detailed Description

## 4.22 rtxPrintStream.h File Reference

```
#include <stdarg.h>
#include "rtxsrc/rtxContext.h"
```

### Classes

- struct OSRTPrintStream

### Typedefs

- typedef void(* rtxPrintCallback )(void *pPrntStrmInfo, const char *fmtspec, va_list arglist)
- typedef struct OSRTPrintStream OSRTPrintStream

### Functions

- int rtxSetPrintStream (OSCTXT *pctxt, rtxPrintCallback myCallback, void *pStrmInfo)
- int rtxSetGlobalPrintStream (rtxPrintCallback myCallback, void *pStrmInfo)
- int rtxPrintToStream (OSCTXT *pctxt, const char *fmtspec,...)
- int rtxDiagToStream (OSCTXT *pctxt, const char *fmtspec, va_list arglist)
- int rtxPrintStreamRelease (OSCTXT *pctxt)
- void rtxPrintStreamToStdoutCB (void *pPrntStrmInfo, const char *fmtspec, va_list arglist)
- void rtxPrintStreamToFileCB (void *pPrntStrmInfo, const char *fmtspec, va_list arglist)

### Variables

- OSRTPrintStream g_PrintStream

### 4.22.1 Detailed Description

Functions that allow printing diagnostic message to a stream using a callback function.

## 4.23 rtxPrintToStream.h File Reference

```
#include <stdio.h>
#include "rtxsrc/rtxContext.h"
```

### Defines

- #define **OSRTINDENTSPACES** 3

### Functions

- void rtxPrintToStreamBoolean (OSCTXT *pctxt, const char *name, OSBOOL value)
- void rtxPrintToStreamDate (OSCTXT *pctxt, const char *name, const OSNumDateTime *pvalue)
- void rtxPrintToStreamTime (OSCTXT *pctxt, const char *name, const OSNumDateTime *pvalue)
- void rtxPrintToStreamDateTime (OSCTXT *pctxt, const char *name, const OSNumDateTime *pvalue)
- void **rtxPrintToStreamGYear** (OSCTXT *pctxt, const char *name, const OSNumDateTime *pvalue)
- void **rtxPrintToStreamGYearMonth** (OSCTXT *pctxt, const char *name, const OSNumDateTime *pvalue)
- void **rtxPrintToStreamGMonth** (OSCTXT *pctxt, const char *name, const OSNumDateTime *pvalue)
- void **rtxPrintToStreamGMonthDay** (OSCTXT *pctxt, const char *name, const OSNumDateTime *pvalue)
- void **rtxPrintToStreamGDay** (OSCTXT *pctxt, const char *name, const OSNumDateTime *pvalue)
- void rtxPrintToStreamInteger (OSCTXT *pctxt, const char *name, OSINT32 value)
- void rtxPrintToStreamInt64 (OSCTXT *pctxt, const char *name, OSINT64 value)
- void rtxPrintToStreamUnsigned (OSCTXT *pctxt, const char *name, OSUINT32 value)
- void rtxPrintToStreamUInt64 (OSCTXT *pctxt, const char *name, OSUINT64 value)
- void rtxPrintToStreamHexStr (OSCTXT *pctxt, const char *name, size_t numocts, const OSOCTET *data)
- void rtxPrintToStreamHexBinary (OSCTXT *pctxt, const char *name, size_t numocts, const OSOCTET *data)
- void rtxPrintToStreamCharStr (OSCTXT *pctxt, const char *name, const char *cstring)
- void rtxPrintToStreamUTF8CharStr (OSCTXT *pctxt, const char *name, const OSUTF8CHAR *cstring)
- void rtxPrintToStreamUnicodeCharStr (OSCTXT *pctxt, const char *name, const OSUNICHAR *str, int nchars)
- void rtxPrintToStreamReal (OSCTXT *pctxt, const char *name, OSREAL value)
- void rtxPrintToStreamNull (OSCTXT *pctxt, const char *name)
- void rtxPrintToStreamNVP (OSCTXT *pctxt, const char *name, const OSUTF8NVP *value)
- int rtxPrintToStreamFile (OSCTXT *pctxt, const char *filename)
- void rtxPrintToStreamIndent (OSCTXT *pctxt)
- void rtxPrintToStreamIncrIndent (OSCTXT *pctxt)
- void rtxPrintToStreamDecrIndent (OSCTXT *pctxt)
- void rtxPrintToStreamCloseBrace (OSCTXT *pctxt)
- void rtxPrintToStreamOpenBrace (OSCTXT *pctxt, const char *)
- void rtxHexDumpToStream (OSCTXT *pctxt, const OSOCTET *data, size_t numocts)
- void rtxHexDumpToStreamEx (OSCTXT *pctxt, const OSOCTET *data, size_t numocts, int bytesPerUnit)

#### 4.23.1 Detailed Description

## 4.24 rtxReal.h File Reference

```
#include "rtxsrc/osSysTypes.h"
#include "rtxsrc/rtxExternDefs.h"
```

### Functions

- OSREAL rtxGetMinusInfinity (void)
- OSREAL rtxGetMinusZero (void)
- OSREAL rtxGetNaN (void)
- OSREAL rtxGetPlusInfinity (void)
- OSBOOL rtxIsMinusInfinity (OSREAL value)
- OSBOOL rtxIsMinusZero (OSREAL value)
- OSBOOL rtxIsNaN (OSREAL value)
- OSBOOL rtxIsPlusInfinity (OSREAL value)
- OSBOOL rtxIsApproximate (OSREAL a, OSREAL b, OSREAL delta)
- OSBOOL rtxIsApproximateAbs (OSREAL a, OSREAL b, OSREAL delta)

### 4.24.1 Detailed Description

Common runtime functions for working with floating-point numbers.

## 4.25 rtxSList.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

### Classes

- struct _OSRTSListNode
- struct _OSRTSList

### Defines

- #define **OSALLOCELEMSNODE**(pctxt, type)

### Typedefs

- typedef struct _OSRTSListNode **OSRTSListNode**
- typedef struct _OSRTSList **OSRTSList**

### Functions

- void rtxSListInit (OSRTSList *pList)
- void rtxSListInitEx (OSCTXT *pctxt, OSRTSList *pList)
- void rtxSListFree (OSRTSList *pList)
- void rtxSListFreeAll (OSRTSList *pList)
- OSRTSList * rtxSListCreate (void)
- OSRTSList * rtxSListCreateEx (OSCTXT *pctxt)
- OSRTSListNode * rtxSListAppend (OSRTSList *pList, void *pData)
- OSBOOL rtxSListFind (OSRTSList *pList, void *pData)
- void rtxSListRemove (OSRTSList *pList, void *pData)

### 4.25.1 Detailed Description

### 4.25.2 Define Documentation

#### 4.25.2.1 #define OSALLOCELEMSNODE(pctxt, type)

**Value:**

```
(type*) (((char*)rtxMemAllocZ (pctxt, sizeof(type) + \
sizeof(OSRTSListNode))) + sizeof(OSRTSListNode))
```

## 4.26 rtxSocket.h File Reference

```
#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 "rtxsrc/osSysTypes.h"
#include "rtxsrc/rtxExternDefs.h"
```

### Defines

- #define **OSRTSOCKET_INVALID** ((OSRTSOCKET)-1)
- #define **OSIPADDR_ANY** ((OSIPADDR)0)
- #define **OSIPADDR_LOCAL** ((OSIPADDR)0x7f000001UL)

### Typedefs

- typedef int OSRTSOCKET
- typedef unsigned long OSIPADDR

### Functions

- int rtxSocketAccept (OSRTSOCKET socket, OSRTSOCKET *pNewSocket, OSIPADDR *destAddr, int *destPort)
- int rtxSocketAddrToStr (OSIPADDR ipAddr, char *pbuf, size_t bufsize)
- int rtxSocketBind (OSRTSOCKET socket, OSIPADDR addr, int port)
- int rtxSocketClose (OSRTSOCKET socket)
- int rtxSocketConnect (OSRTSOCKET socket, const char *host, int port)
- int rtxSocketConnectTimed (OSRTSOCKET socket, const char *host, int port, int nsecs)
- int rtxSocketCreate (OSRTSOCKET *psocket)
- int **rtxSocketCreateUDP** (OSRTSOCKET *psocket)
- int rtxSocketGetHost (const char *host, struct in_addr *inaddr)
- int rtxSocketsInit ()
- int rtxSocketListen (OSRTSOCKET socket, int maxConnection)
- int rtxSocketParseURL (char *url, char **protocol, char **address, int *port)
- int rtxSocketRecv (OSRTSOCKET socket, OSOCTET *pbuf, size_t bufsize)
- int rtxSocketRecvTimed (OSRTSOCKET socket, OSOCTET *pbuf, size_t bufsize, OSUINT32 secs)
- int rtxSocketSelect (int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout)
- int rtxSocketSend (OSRTSOCKET socket, const OSOCTET *pdata, size_t size)
- int rtxSocketSetBlocking (OSRTSOCKET socket, OSBOOL value)
- int rtxSocketStrToAddr (const char *pIPAddrStr, OSIPADDR *pIPAddr)

### 4.26.1 Detailed Description

### 4.26.2 Typedef Documentation

#### 4.26.2.1 typedef int OSRTSOCKET

Socket handle type definition

## 4.27 rtxStack.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

### Classes

- struct _OSRTStack

### Defines

- #define rtxStackIsEmpty(stack) (OSBOOL)((stack).dlist.count == 0)

### Typedefs

- typedef struct _OSRTStack **OSRTStack**

### Functions

- OSRTStack * rtxStackCreate (OSCTXT *pctxt)
- void rtxStackInit (OSCTXT *pctxt, OSRTStack *pStack)
- void * rtxStackPop (OSRTStack *pStack)
- int rtxStackPush (OSRTStack *pStack, void *pData)
- void * rtxStackPeek (OSRTStack *pStack)

### 4.27.1 Detailed Description

Simple FIFO stack for storing void pointers to any type of data.

## 4.28 rtxStream.h File Reference

```
#include "rtxsrc/rtxContext.h"
#include "rtxsrc/rtxMemBuf.h"
```

### Classes

- struct OSRTSTREAM

### Defines

- #define **OSRTSTRMF_INPUT** 0x0001
- #define **OSRTSTRMF_OUTPUT** 0x0002
- #define **OSRTSTRMF_BUFFERED** 0x8000
- #define **OSRTSTRMF_UNBUFFERED** 0x4000
- #define **OSRTSTRMF_POSMARKED** 0x2000
- #define **OSRTSTRMF_FIXINMEM** 0x1000
- #define **OSRTSTRMF_BUF_INPUT** (OSRTSTRMF_INPUT|OSRTSTRMF_BUFFERED)
- #define **OSRTSTRMF_BUF_OUTPUT** (OSRTSTRMF_OUTPUT|OSRTSTRMF_BUFFERED)
- #define **OSRTSTRMID_FILE** 1
- #define **OSRTSTRMID_SOCKET** 2
- #define **OSRTSTRMID_MEMORY** 3
- #define **OSRTSTRMID_BUFFERED** 4
- #define **OSRTSTRMID_DIRECTBUF** 5
- #define **OSRTSTRMID_CTXTBUF** 6
- #define **OSRTSTRMID_ZLIB** 7
- #define **OSRTSTRMID_USER** 1000
- #define **OSRTSTRM_K_BUFSIZE** 1024
- #define **OSRTSTRM_K_INVALIDMARK** ((size_t)-1)
- #define **OSRTSTREAM_BYTEINDEX**(pctxt)
- #define **OSRTSTREAM_ID**(pctxt) ((pctxt)->pStream->id)
- #define **OSRTSTREAM_FLAGS**(pctxt) ((pctxt)->pStream->flags)

### Typedefs

- typedef long(* OSRTStreamReadProc )(struct OSRTSTREAM *pStream, OSOCTET *pbuffer, size_t bufSize)
- typedef long(* OSRTStreamBlockingReadProc )(struct OSRTSTREAM *pStream, OSOCTET *pbuffer, size_t toReadBytes)
- typedef long(* OSRTStreamWriteProc )(struct OSRTSTREAM *pStream, const OSOCTET *data, size_t numocts)
- typedef int(* OSRTStreamFlushProc )(struct OSRTSTREAM *pStream)
- typedef int(* OSRTStreamCloseProc )(struct OSRTSTREAM *pStream)
- typedef int(* OSRTStreamSkipProc )(struct OSRTSTREAM *pStream, size_t skipBytes)
- typedef int(* OSRTStreamMarkProc )(struct OSRTSTREAM *pStream, size_t readAheadLimit)
- typedef int(* OSRTStreamResetProc )(struct OSRTSTREAM *pStream)
- typedef int(* OSRTStreamGetPosProc )(struct OSRTSTREAM *pStream, size_t *ppos)
- typedef int(* OSRTStreamSetPosProc )(struct OSRTSTREAM *pStream, size_t pos)
- typedef struct OSRTSTREAM OSRTSTREAM

## Functions

- int rtxStreamClose (OSCTXT *pctxt)
- int rtxStreamFlush (OSCTXT *pctxt)
- int rtxStreamInit (OSCTXT *pctxt)
- long rtxStreamRead (OSCTXT *pctxt, OSOCTET *pbuffer, size_t bufSize)
- long rtxStreamBlockingRead (OSCTXT *pctxt, OSOCTET *pbuffer, size_t readBytes)
- int rtxStreamSkip (OSCTXT *pctxt, size_t skipBytes)
- long rtxStreamWrite (OSCTXT *pctxt, const OSOCTET *data, size_t numocts)
- int rtxStreamGetIOBytes (OSCTXT *pctxt, size_t *pPos)
- int rtxStreamMark (OSCTXT *pctxt, size_t readAheadLimit)
- int rtxStreamReset (OSCTXT *pctxt)
- OSBOOL rtxStreamMarkSupported (OSCTXT *pctxt)
- OSBOOL rtxStreamIsOpened (OSCTXT *pctxt)
- OSBOOL rtxStreamIsReadable (OSCTXT *pctxt)
- OSBOOL rtxStreamIsWritable (OSCTXT *pctxt)
- int rtxStreamRelease (OSCTXT *pctxt)
- void rtxStreamSetCapture (OSCTXT *pctxt, OSRTMEMBUF *pmembuf)
- OSRTMEMBUF * rtxStreamGetCapture (OSCTXT *pctxt)
- int rtxStreamGetPos (OSCTXT *pctxt, size_t *ppos)
- int rtxStreamSetPos (OSCTXT *pctxt, size_t pos)

### 4.28.1 Detailed Description

Input/output data stream type definitions and function prototypes.

## 4.29 rtxStreamBuffered.h File Reference

```
#include "rtxsrc/rtxStream.h"
```

### Defines

- #define **OSRTSTRMCM_RESTORE_UNDERLAYING_AFTER_RESET** 0x0001

### Functions

- int **rtxStreamBufferedCreate** (OSCTXT *pctxt, OSUINT32 mode)
- int **rtxStreamBufferedRelease** (OSCTXT *pctxt)
- int **rtxStreamBufferedSetPreReadBuf** (OSCTXT *pctxt, const OSOCTET *pdata, size_t datalen)
- int **rtxStreamBufferedPrependReadBuf** (OSCTXT *pctxt, const OSOCTET *pdata, size_t datalen)

### 4.29.1 Detailed Description

## 4.30 rtxStreamFile.h File Reference

```
#include <stdio.h>
#include "rtxsrc/rtxStream.h"
```

### Functions

- int rtxStreamFileAttach (OSCTXT *pctxt, FILE *pFile, OSUINT16 flags)
- int rtxStreamFileOpen (OSCTXT *pctxt, const char *pFilename, OSUINT16 flags)
- int rtxStreamFileCreateReader (OSCTXT *pctxt, const char *pFilename)
- int rtxStreamFileCreateWriter (OSCTXT *pctxt, const char *pFilename)

### 4.30.1 Detailed Description

## 4.31 rtxStreamHexText.h File Reference

```
#include "rtxsrc/osSysTypes.h"
#include "rtxsrc/rtxExternDefs.h"
#include "rtxsrc/rtxStream.h"
```

### Functions

- int rtxStreamHexTextAttach (OSCTXT *pctxt, OSUINT16 flags)

### 4.31.1 Detailed Description

### 4.31.2 Function Documentation

#### 4.31.2.1 int rtxStreamHexTextAttach (OSCTXT * *pctxt*, OSUINT16 *flags*)

This function initializes a hexText stream and attaches it to the existing stream defined within the context. This type of stream object can only be used with an existing stream. It acts as a filter to perform conversion to/from hex characters to binary data.

**Parameters**

***pctxt*** Pointer to context structure variable.

***flags*** Specifies the access mode for the stream:

- OSRTSTRMF_INPUT = input (reading) stream;
- OSRTSTRMF_OUTPUT = output (writing) stream.

**Returns**

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

## 4.32 rtxStreamMemory.h File Reference

```
#include "rtxsrc/rtxStream.h"
```

### Classes

- struct DirBufDesc

### Typedefs

- typedef struct DirBufDesc **DirBufDesc**

### Functions

- int rtxStreamMemoryCreate (OSCTXT *pctxt, OSUINT16 flags)
- int rtxStreamMemoryAttach (OSCTXT *pctxt, OSOCTET *pMemBuf, size_t bufSize, OSUINT16 flags)
- OSOCTET * rtxStreamMemoryGetBuffer (OSCTXT *pctxt, size_t *pSize)
- int rtxStreamMemoryCreateReader (OSCTXT *pctxt, OSOCTET *pMemBuf, size_t bufSize)
- int rtxStreamMemoryCreateWriter (OSCTXT *pctxt, OSOCTET *pMemBuf, size_t bufSize)
- int rtxStreamMemoryResetWriter (OSCTXT *pctxt)

#### 4.32.1 Detailed Description

## 4.33 rtxStreamSocket.h File Reference

```
#include "rtxsrc/rtxStream.h"
#include "rtxsrc/rtxSocket.h"
```

### Functions

- int rtxStreamSocketAttach (OSCTXT *pctxt, OSRTSOCKET socket, OSUINT16 flags)
- int rtxStreamSocketClose (OSCTXT *pctxt)
- int rtxStreamSocketCreateWriter (OSCTXT *pctxt, const char *host, int port)
- int rtxStreamSocketSetOwnership (OSCTXT *pctxt, OSBOOL ownSocket)
- int rtxStreamSocketSetReadTimeout (OSCTXT *pctxt, OSUINT32 nsecs)

### 4.33.1 Detailed Description

## 4.34 rtxUTF8.h File Reference

```
#include "rtxsrc/rtxContext.h"
```

### Defines

- #define **rtxUTF8StrToInt32** rtxUTF8StrToInt
- #define **rtxUTF8StrToUInt32** rtxUTF8StrToUInt
- #define RTUTF8STRCMPL(name, lstr) rtxUTF8Strcmp(name,(const OSUTF8CHAR*)lstr)
- #define **OSRTCHKUTF8LEN**(str, lower, upper, stat)

### Functions

- long rtxUTF8ToUnicode (OSCTXT *pctxt, const OSUTF8CHAR *inbuf, OSUNICHAR *outbuf, size_t outbufsiz)
- int rtxValidateUTF8 (OSCTXT *pctxt, const OSUTF8CHAR *inbuf)
- size_t rtxUTF8Len (const OSUTF8CHAR *inbuf)
- size_t **rtxCalcUTF8Len** (const OSUTF8CHAR *inbuf, size_t inbufBytes)
- size_t rtxUTF8LenBytes (const OSUTF8CHAR *inbuf)
- int rtxUTF8CharSize (OS32BITCHAR wc)
- int rtxUTF8EncodeChar (OS32BITCHAR wc, OSOCTET *buf, size_t bufsiz)
- int rtxUTF8DecodeChar (OSCTXT *pctxt, const OSUTF8CHAR *pinbuf, int *pInsize)
- OS32BITCHAR rtxUTF8CharToWC (const OSUTF8CHAR *buf, OSUINT32 *len)
- OSUTF8CHAR * rtxUTF8StrChr (OSUTF8CHAR *utf8str, OS32BITCHAR utf8char)
- OSUTF8CHAR * rtxUTF8Strdup (OSCTXT *pctxt, const OSUTF8CHAR *utf8str)
- OSUTF8CHAR * rtxUTF8Strndup (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, size_t nbytes)
- OSUTF8CHAR * rtxUTF8StrRefOrDup (OSCTXT *pctxt, const OSUTF8CHAR *utf8str)
- OSBOOL rtxUTF8StrEqual (const OSUTF8CHAR *utf8str1, const OSUTF8CHAR *utf8str2)
- OSBOOL rtxUTF8StrnEqual (const OSUTF8CHAR *utf8str1, const OSUTF8CHAR *utf8str2, size_t count)
- int rtxUTF8Strcmp (const OSUTF8CHAR *utf8str1, const OSUTF8CHAR *utf8str2)
- int rtxUTF8Strncmp (const OSUTF8CHAR *utf8str1, const OSUTF8CHAR *utf8str2, size_t count)
- OSUTF8CHAR * rtxUTF8Strcpy (OSUTF8CHAR *dest, size_t bufsiz, const OSUTF8CHAR *src)
- OSUTF8CHAR * rtxUTF8Strncpy (OSUTF8CHAR *dest, size_t bufsiz, const OSUTF8CHAR *src, size_t nchars)
- OSUINT32 rtxUTF8StrHash (const OSUTF8CHAR *str)
- const OSUTF8CHAR * rtxUTF8StrJoin (OSCTXT *pctxt, const OSUTF8CHAR *str1, const OSUTF8CHAR *str2, const OSUTF8CHAR *str3, const OSUTF8CHAR *str4, const OSUTF8CHAR *str5)
- int rtxUTF8StrToBool (const OSUTF8CHAR *utf8str, OSBOOL *pvalue)
- int rtxUTF8StrnToBool (const OSUTF8CHAR *utf8str, size_t nbytes, OSBOOL *pvalue)
- int rtxUTF8StrToDouble (const OSUTF8CHAR *utf8str, OSREAL *pvalue)
- int rtxUTF8StrnToDouble (const OSUTF8CHAR *utf8str, size_t nbytes, OSREAL *pvalue)
- int rtxUTF8StrToInt (const OSUTF8CHAR *utf8str, OSINT32 *pvalue)
- int rtxUTF8StrnToInt (const OSUTF8CHAR *utf8str, size_t nbytes, OSINT32 *pvalue)
- int rtxUTF8StrToUInt (const OSUTF8CHAR *utf8str, OSUINT32 *pvalue)
- int rtxUTF8StrnToUInt (const OSUTF8CHAR *utf8str, size_t nbytes, OSUINT32 *pvalue)
- int rtxUTF8StrToSize (const OSUTF8CHAR *utf8str, size_t *pvalue)
- int rtxUTF8StrnToSize (const OSUTF8CHAR *utf8str, size_t nbytes, size_t *pvalue)
- int rtxUTF8StrToInt64 (const OSUTF8CHAR *utf8str, OSINT64 *pvalue)
- int rtxUTF8StrnToInt64 (const OSUTF8CHAR *utf8str, size_t nbytes, OSINT64 *pvalue)
- int rtxUTF8StrToUInt64 (const OSUTF8CHAR *utf8str, OSUINT64 *pvalue)

- int rtxUTF8StrnToUInt64 (const OSUTF8CHAR *utf8str, size_t nbytes, OSUINT64 *pvalue)
- int rtxUTF8ToDynUniStr (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, const OSUNICHAR **ppdata, OSUINT32 *pnchars)
- int rtxUTF8RemoveWhiteSpace (const OSUTF8CHAR *utf8instr, size_t nbytes, const OSUTF8CHAR **putf8outstr)
- int rtxUTF8StrToDynHexStr (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, OSDynOctStr *pvalue)
- int rtxUTF8StrnToDynHexStr (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, size_t nbytes, OSDynOctStr *pvalue)
- int rtxUTF8StrToNamedBits (OSCTXT *pctxt, const OSUTF8CHAR *utf8str, const OSBitMapItem *pBitMap, OSOCTET *pvalue, OSUINT32 *pnbits, OSUINT32 bufsize)
- const OSUTF8CHAR * rtxUTF8StrNextTok (OSUTF8CHAR *utf8str, OSUTF8CHAR **ppNext)

### 4.34.1 Detailed Description

Utility functions for handling UTF-8 strings.

# Index